

# **Synopsys Synplify Pro<sup>®</sup> for GoWin**

## User Guide

---

September 2019

**SYNOPSYS<sup>®</sup>**

## Copyright Notice and Proprietary Information

© 2019 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](http://www.synopsys.com)

September 2019



# Contents

---

## Chapter 1: Introduction

Synopsys FPGA and Prototyping Products .....	14
FPGA Implementation Tools .....	15
Symphony Model Compiler .....	16
Rapid Prototyping .....	17
Starting the Synthesis Tool .....	18
Starting the Synthesis Tool in Interactive Mode .....	18
Starting the Tool in Batch Mode .....	18
Logic Synthesis Overview .....	19
Synthesizing Your Design .....	20
BEST Algorithms .....	23
Projects and Implementations .....	23
User Interface Overview .....	24

## Chapter 2: FPGA Synthesis Design Flows

Logic Synthesis Design Flow .....	28
-----------------------------------	----

## Chapter 3: Preparing the Input

Setting Up HDL Source Files .....	32
Creating HDL Source Files .....	32
Using the Context Help Editor .....	34
Checking HDL Source Files .....	36
Editing HDL Source Files with the Built-in Text Editor .....	37
Setting Editing Window Preferences .....	40
Using an External Text Editor .....	42
Using Library Extensions for Verilog Library Files .....	43
Using Mixed Language Source Files .....	46
Working with Constraint Files .....	51
When to Use Constraint Files over Source Code .....	51

Tcl Syntax Guidelines for Constraint Files .....	52
Checking Constraint Files .....	53
<b>Chapter 4: Setting Up a Logic Synthesis Project</b>	
Setting Up Project Files .....	56
Creating a Project File .....	56
Opening an Existing Project File .....	59
Making Changes to a Project .....	60
Setting Project View Display Preferences .....	61
Updating Verilog Include Paths in Older Project Files .....	64
Managing Project File Hierarchy .....	65
Creating Custom Folders .....	65
Manipulating Custom Project Folders .....	69
Manipulating Custom Files .....	70
Setting Up Implementations .....	72
Working with Multiple Implementations .....	72
Setting Logic Synthesis Implementation Options .....	74
Setting Device Options .....	74
Setting Optimization Options .....	76
Specifying Global Frequency and Constraint Files .....	78
Specifying Result Options .....	80
Specifying Timing Report Output .....	82
Setting Verilog and VHDL Options .....	83
Specifying Attributes and Directives .....	89
Specifying Attributes and Directives in VHDL .....	90
Specifying Attributes and Directives in Verilog .....	91
Specifying Attributes Using the SCOPE Editor .....	92
Specifying Attributes in the Constraints File .....	95
Handling Properties with Attributes or Directives .....	96
Searching Files .....	97
Identifying the Files to Search .....	98
Filtering the Files to Search .....	98
Initiating the Search .....	99
Search Results .....	99
Archiving Files and Projects .....	100
Archive a Project .....	100
Un-Archive a Project .....	103
Copy a Project .....	106
Support for Hierarchical Include Paths .....	110

## Chapter 5: Specifying Constraints

Using the SCOPE Editor . . . . .	114
Creating Constraints in the SCOPE Editor . . . . .	114
Creating Constraints With the FDC Template Command . . . . .	118
Specifying SCOPE Constraints . . . . .	121
Entering and Editing SCOPE Constraints . . . . .	121
Setting Clock and Path Constraints . . . . .	123
Defining Input and Output Constraints . . . . .	124
Specifying Standard I/O Pad Types . . . . .	126
Using the TCL View of SCOPE GUI . . . . .	126
Guidelines for Entering and Editing Constraints . . . . .	130
Specifying Timing Exceptions . . . . .	132
Defining From/To/Through Points for Timing Exceptions . . . . .	132
Defining Multicycle Paths . . . . .	136
Defining False Paths . . . . .	137
Finding Objects with Tcl find and expand . . . . .	138
Specifying Search Patterns for Tcl find . . . . .	138
Refining Tcl Find Results with -filter . . . . .	140
Using the Tcl Find Command to Define Collections . . . . .	143
Using the Tcl expand Command to Define Collections . . . . .	144
Checking Tcl find and expand Results . . . . .	145
Using Tcl find and expand in Batch Mode . . . . .	147
Using Collections . . . . .	147
Comparison of Methods for Defining Collections . . . . .	147
Creating and Using SCOPE Collections . . . . .	148
Creating Collections using Tcl Commands . . . . .	151
Viewing and Manipulating Collections with Tcl Commands . . . . .	154

## Chapter 6: Synthesizing and Analyzing the Results

Synthesizing Your Design . . . . .	160
Running Logic Synthesis . . . . .	160
Using Up-to-date Checking for Job Management . . . . .	161
Checking Log File Results . . . . .	166
Viewing and Working with the Log File . . . . .	166
Accessing Specific Reports Quickly . . . . .	170
Accessing Results Remotely . . . . .	173
Analyzing Results Using the Log File Reports . . . . .	176
Using the Watch Window . . . . .	177
Checking Resource Usage . . . . .	178

Querying Metrics for a Design .....	180
Handling Messages .....	182
Checking Results in the Message Viewer .....	182
Filtering Messages in the Message Viewer .....	184
Filtering Messages from the Command Line .....	186
Automating Message Filtering with a Tcl Script .....	187
Log File Message Controls .....	189
Working with Downgradable Errors and Critical Warnings .....	194

## **Chapter 7: Analyzing with HDL Analyst**

Working in the Schematic .....	198
Opening the Views .....	198
Cloning Schematics .....	201
Viewing Object Properties .....	202
Viewing Objects with Constant Values .....	205
Viewing Objects in a Source File .....	206
Selecting Objects in the Schematic .....	210
Grouping Objects in the Schematic .....	214
Moving Between Views in a Schematic Window .....	217
Setting Schematic Preferences .....	218
Exploring Design Hierarchy .....	221
Traversing Design Hierarchy with the Hierarchy Browser .....	221
Exploring Object Hierarchy with Push/Pop Commands .....	223
Finding Objects .....	227
Browsing to Find Objects in HDL Analyst Views .....	227
Using Wildcards with the Find Command .....	237
Crossprobing .....	238
Crossprobing within a View .....	238
Crossprobing from an HDL Analyst View .....	239
Crossprobing to the Source Code .....	240
Crossprobing from the Text Editor Window .....	242
Crossprobing from the Log File .....	243
Analyzing With the HDL Analyst Tool .....	245
Viewing Design Hierarchy and Context .....	245
Filtering Schematics .....	249
Expanding Pin and Net Logic .....	251
Dissolving and Partial Dissolving of Buses and Pins .....	255
Dissolving of Ports .....	258
Flattening Schematic Hierarchy .....	259
Using the FSM Viewer .....	261

Working in the Standard Schematic . . . . .	265
Differentiating Between the HDL Analyst Views . . . . .	266
Opening the Views . . . . .	266
Viewing Object Properties . . . . .	267
Selecting Objects in the RTL/Technology Views . . . . .	273
Working with Multisheet Schematics . . . . .	274
Moving Between Views in a Schematic Window . . . . .	275
Setting Schematic Preferences . . . . .	276
Managing Windows . . . . .	278
Exploring Design Hierarchy (Standard) . . . . .	280
Traversing Design Hierarchy with the Hierarchy Browser . . . . .	280
Exploring Object Hierarchy by Pushing/Popping . . . . .	281
Exploring Object Hierarchy of Transparent Instances . . . . .	286
Finding Objects (Standard) . . . . .	288
Browsing to Find Objects in HDL Analyst Views . . . . .	288
Using Find for Hierarchical and Restricted Searches . . . . .	291
Using Wildcards with the Find Command . . . . .	295
Combining Find with Filtering to Refine Searches . . . . .	300
Using Find to Search the Output Netlist . . . . .	300
Crossprobing (Standard) . . . . .	303
Crossprobing within an RTL/Technology View . . . . .	303
Crossprobing from the RTL/Technology View . . . . .	304
Crossprobing from the Text Editor Window . . . . .	306
Crossprobing from the Tcl Script Window . . . . .	309
Crossprobing from the FSM Viewer . . . . .	310
Analyzing With the Standard HDL Analyst Tool . . . . .	311
Viewing Design Hierarchy and Context . . . . .	312
Filtering Schematics . . . . .	315
Expanding Pin and Net Logic . . . . .	318
Expanding and Viewing Connections . . . . .	321
Flattening Schematic Hierarchy . . . . .	322
Minimizing Memory Usage While Analyzing Designs . . . . .	327
Using the FSM Viewer (Standard) . . . . .	329

## Chapter 8: Analyzing Timing

Analyzing Timing in Schematic Views . . . . .	336
Viewing Timing Information . . . . .	336
Annotating Timing Information in the Schematic Views . . . . .	337
Analyzing Clock Trees in the RTL View . . . . .	339
Viewing Critical Paths . . . . .	339

Handling Negative Slack .....	342
Generating Custom Timing Reports with STA .....	344
Using Analysis Design Constraints .....	347
Scenarios for Using Analysis Design Constraints .....	348
Creating an ADC File .....	349
Using Object Names Correctly in the adc File .....	353
Using Auto Constraints .....	354
Results of Auto Constraints .....	356

## **Chapter 9: Inferring High-Level Objects**

Defining Black Boxes for Synthesis .....	360
Instantiating Black Boxes and I/Os in Verilog .....	360
Instantiating Black Boxes and I/Os in VHDL .....	362
Adding Black Box Timing Constraints .....	365
Adding Other Black Box Attributes .....	368
Defining State Machines for Synthesis .....	370
Defining State Machines in Verilog .....	370
Defining State Machines in VHDL .....	371
Specifying FSMs with Attributes and Directives .....	372
Initializing RAMs .....	374
Initializing RAMs in Verilog .....	374
Initializing RAMs in VHDL .....	375

## **Chapter 10: Specifying Design-Level Optimizations**

Tips for Optimization .....	380
General Optimization Tips .....	380
Optimizing for Area .....	381
Optimizing for Timing .....	382
Pipelining .....	384
Prerequisites for Pipelining .....	384
Pipelining the Design .....	385
Retiming .....	388
Controlling Retiming .....	388
Retiming Example .....	391
Retiming Report .....	392
How Retiming Works .....	392
Preserving Objects from Being Optimized Away .....	396
Using syn_keep for Preservation or Replication .....	397

Controlling Hierarchy Flattening . . . . .	400
Preserving Hierarchy . . . . .	401
Optimizing Fanout . . . . .	402
Setting Fanout Limits . . . . .	402
Controlling Buffering and Replication . . . . .	404
Sharing Resources . . . . .	406
Inserting I/Os . . . . .	406
Optimizing State Machines . . . . .	407
Deciding when to Optimize State Machines . . . . .	408
Running the FSM Compiler . . . . .	408
Inserting Probes . . . . .	413
Specifying Probes in the Source Code . . . . .	413
Adding Probe Attributes Interactively . . . . .	415

## Chapter 11: Working with Compile Points

Compile Point Basics . . . . .	418
Advantages of Compile Point Design . . . . .	418
Manual Compile Points . . . . .	420
Nested Compile Points . . . . .	420
Compile Point Types . . . . .	422
Compile Point Synthesis Basics . . . . .	426
Compile Point Constraint Files . . . . .	426
Interface Logic Models . . . . .	428
Interface Timing for Compile Points . . . . .	429
Compile Point Synthesis . . . . .	432
Incremental Compile Point Synthesis . . . . .	434
Forward-annotation of Compile Point Timing Constraints . . . . .	435
Synthesizing Compile Points . . . . .	436
The Manual Compile Point Flow . . . . .	436
Creating a Top-Level Constraints File for Compile Points . . . . .	439
Defining Manual Compile Points . . . . .	441
Setting Constraints at the Compile Point Level . . . . .	444
Analyzing Compile Point Results . . . . .	446
Resynthesizing Incrementally . . . . .	449
Resynthesizing Compile Points Incrementally . . . . .	449
Working with Gated Clocks . . . . .	454

## Chapter 12: Clock Conversion

Obstacles to Conversion .....	456
Prerequisites for Gated Clock Conversion .....	457
Defining Clocks Properly .....	459
Synthesizing a Gated-Clock Design .....	464
Accessing the Clock Conversion Report .....	465
Analyzing the Clock Conversion Report .....	466
Interpreting Gated Clock Error Messages .....	469
Disabling Individual Gated Clock Conversions .....	472
Integrated Clock Gating Cells .....	472
Using Gated Clocks for Black Boxes .....	476
OR Gates Driving Latches .....	477
Obstructions to Optimization .....	478
Unsupported Constructs .....	480
Other Potential Workarounds to Solve Clock-Conversion Issues .....	480
Restrictions on Using Gated Clocks .....	480
 Optimizing Generated Clocks .....	484
Enabling Generated-Clock Optimization .....	484
Conditions for Generated-Clock Optimization .....	485
Generated-Clock Optimization Examples .....	485
 <b>Chapter 13: Optimizing for GoWin Designs</b>	
Instantiating GoWin Macros .....	489
Combinational Logic Support .....	490
Limitation .....	490
Handling Tristates .....	490
Handling I/Os and Buffers .....	491
Limitations .....	491
 <b>Chapter 14: Working with Synthesis Output</b>	
Passing Information to the P&R Tools .....	494
Generating Vendor-Specific Output .....	495
Targeting Output to Your Vendor .....	495
 <b>Chapter 15: Running Post-Synthesis Operations</b>	
Simulating with the VCS Tool .....	497

## CHAPTER 1

# Introduction

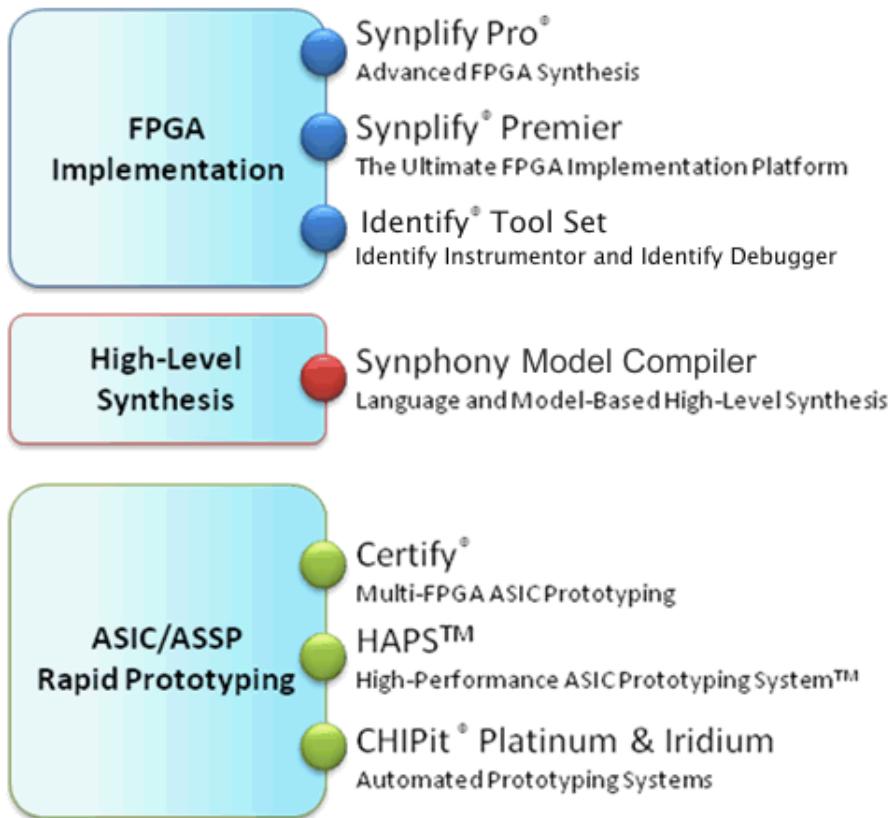
---

This introduction to the Synplify Pro® software describes the following:

- [Synopsys FPGA and Prototyping Products](#), on page 14
- [Starting the Synthesis Tool](#), on page 18
- [Logic Synthesis Overview](#), on page 19
- [User Interface Overview](#), on page 24

# Synopsys FPGA and Prototyping Products

The following figure displays the Synopsys FPGA and Prototyping family of products.



## FPGA Implementation Tools

The Synplify Pro and Premier products are RTL synthesis tools especially designed for FPGAs (field programmable gate arrays) and CPLDs (complex programmable logic devices).

### Synplify Pro Synthesis Software

The Synplify Pro FPGA synthesis software is the de facto industry standard for producing high-performance, cost-effective FPGA designs. Its unique Behavior Extracting Synthesis Technology® (B.E.S.T.™) algorithms, perform high-level optimizations before synthesizing the RTL code into specific FPGA logic. This approach allows for superior optimizations across the FPGA, fast runtimes, and the ability to handle very large designs. The software supports the latest VHDL and Verilog language constructs, including SystemVerilog and VHDL 2008. The tool is technology independent allowing quick and easy retargeting between FPGA devices and vendors from a single design project.

### Synplify Premier Synthesis Software

The Synplify Premier functionality is a superset of the Synplify Pro tool, providing the ultimate FPGA implementation and debug environment. It includes a comprehensive suite of tools and technologies for advanced FPGA designers, and also serves as the synthesis engine for ASIC prototypers targeting single FPGA-based prototypes.

The Synplify Premier product offers both FPGA designers and ASIC prototypers targeting single FPGAs with the most efficient method of design implementation and debug. On the design implementation side, it includes functionality for timing closure, logic verification, IP usage, ASIC compatibility, and DSP implementation, as well as a tight integration with FPGA vendor back-end tools. On the debug side, it provides for in-system verification of FPGAs which dramatically accelerates the debug process, and also includes a rapid and incremental method for finding elusive design problems.

The Synplify Premier product offers FPGA designers and ASIC prototypers, targeting single FPGA-based prototypes, with the most efficient method of design implementation and debug. The Synplify Premier software provides in-system verification of FPGAs, dramatically accelerates the debug process, and provides a rapid and incremental method for finding elusive design problems. Features exclusively supported in the Synplify Premier tool are the following:

- Design Planning (Optional)
- DesignWare Support
- Distributed Processing
- Unified Power Format (UPF)

## Identify Tool Set

The Identify® tool set allows you to debug an operating FPGA directly in the source RTL code. The Identify software is used to verify your design in hardware as you would in simulation, however much faster and with in-system stimulus. Designers and verification engineers are able to navigate the design graphically and instrument signals directly in RTL with which they are familiar, as probes or sample triggers. After synthesis, results are viewed embedded in the RTL source code or in a waveform. Design iterations are rapidly performed using incremental place and route. Identify software is closely integrated with synthesis and routing tools to create a seamless development environment.

## Symphony Model Compiler

Symphony Model Compiler is a language and model-based high-level synthesis technology that provides an efficient path from algorithm concept to silicon. Designers can construct high-level algorithm models from math languages and IP model libraries, then use the Symphony Model Compiler engine to synthesize optimized HDL implementations for FPGA and ASIC architectural exploration and rapid prototyping. In addition, Symphony Model Compiler generates high performance C-models for system validation and early software development in virtual platforms. Key features for this product include:

- MATLAB Language Synthesis
- Automated Fixed-point Conversion Tools
- Synthesizable Fixed-point High Level IP Model Library
- High Level Synthesis Optimizations and Transformations
- Integrated FPGA and ASIC Design Flows
- HDL Testbench Generation

- C-model Generation for Software Development and System Validation

## Rapid Prototyping

The Certify® and Identify products are tightly integrated with the HAPS™ and ChipIT® hardware tools.

### Certify Product

The Certify software is the leading implementation and partitioning tool for ASIC designers using FPGA-based prototypes to verify their designs. The tool provides a quick and easy method for partitioning large ASIC designs into multi-FPGA prototyping boards. Powerful features allow the tool to adapt easily to existing device flows, therefore, speeding up the verification process and helping with the time-to-market challenges. Key features include the following:

- Graphical User Interface (GUI) Flow Guide
- Manual Partitioning
- Synopsys Design Constraints Support for Timing Management
- Multi-core Parallel Processing Support for Faster Runtimes
- Support for Most Current FPGA Devices
- Industry Standard Synplify Premier Synthesis Support
- Compatible with HAPS Boards Including HSTDm

# 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
  - Start->Programs->Synopsys->*Synplify Pro version*
- To start the tool from a DOS command line, specify the executable:
  - *installDirectory\bin\synplify\_pro.exe*

The executable name is the name of the product followed by an exe file extension.

- To start the synthesis tool from a Linux platform, type the appropriate command at the system prompt:
  - *synplify\_pro*

For information about using the synthesis tool in batch mode, see [Starting the Tool in Batch Mode, on page 18](#).

## Starting the Tool in Batch Mode

The command to start the synthesis tool from the command line includes a number of command line options. These options control tool action on startup and, in many cases, can be combined on the same command line. To start the synthesis tool, use the following syntax:

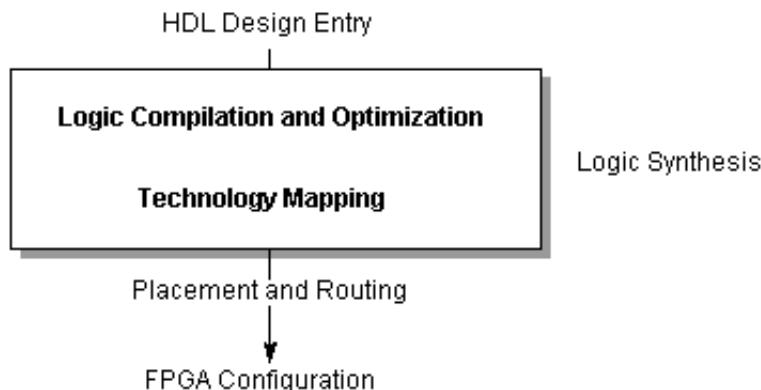
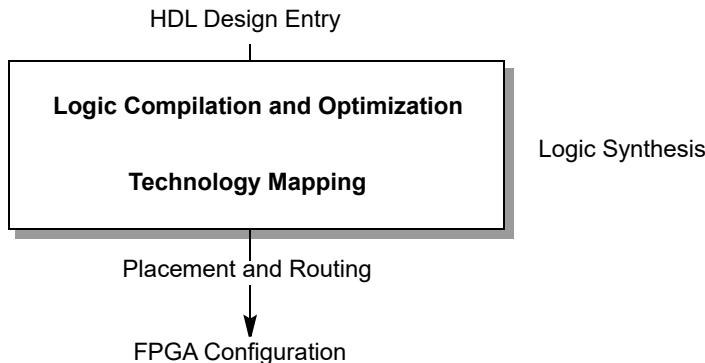
*toolName [-option ... ] [projectFile]*

In the syntax statement, *toolName* is *synplify\_pro*. For complete syntax details, refer to [synplify\\_pro, on page 113](#) in the *Command Reference*.

# Logic Synthesis Overview

When you run the synthesis tool, it performs *logic synthesis*. This consists of two stages:

- Logic compilation (HDL language synthesis) and optimization
- Technology mapping

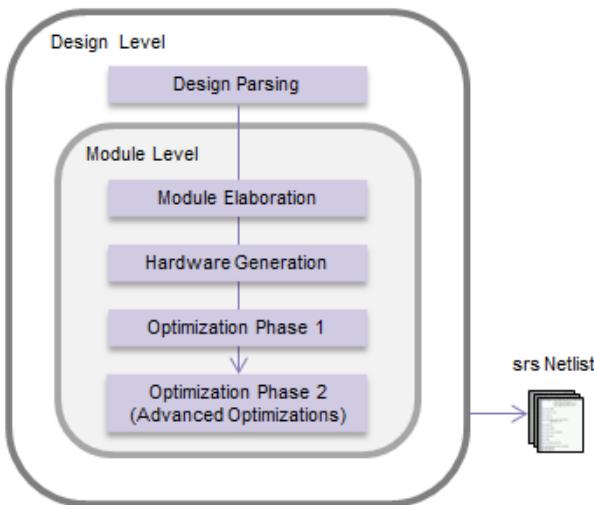


## Logic Compilation

The synthesis tool first compiles input HDL source code, which describes the design at a high level of abstraction, to known structural elements. Next, it optimizes the design in two phases, making it as small as possible to improve

circuit performance. These optimizations are technology independent. The final result is an srs database, which can be graphically represented in the RTL schematic view.

The following figure summarizes the stages of the standard compiler flow:



You can also run the compiler incrementally.

## Technology Mapping

During this stage, the tool optimizes the logic for the target technology, by mapping it to technology-specific components. It uses architecture-specific techniques to perform additional optimizations. Finally, it generates a design netlist for placement and routing.

## Synthesizing Your Design

The synthesis tool accepts high-level designs written in industry-standard hardware description languages (Verilog and VHDL) and uses Behavior Extracting Synthesis Technology® (BEST™) algorithms to keep the design at a high level of abstraction for better optimization. The tool can also write VHDL and Verilog netlists after synthesis, which you can simulate to verify functionality.

You perform the following actions to synthesize your design.

1. Access your design project: open an existing project or create a new one.
2. Specify the input source files to use. Right-click the project name in the Project view, then choose Add Source Files.

Select the desired Verilog, VHDL, or IP files, then click OK. (See the examples in the directory *installation\_dir/examples*, where *installation\_dir* is the directory where the product is installed.)

You can also add source files in the Project view by dragging and dropping them there from a Windows® Explorer folder (Microsoft® Windows® operating system only).

*Top-level file:* The last file compiled is the top-level file. You can designate a new top-level file by moving the desired file to the bottom of the source files list in the Project view, or by using the Implementation Options dialog box.

3. Add design constraints. Use the SCOPE spreadsheet to assign system-level and circuit-path timing constraints that can be forward-annotated.

See [SCOPE Tabs, on page 147](#), for details on the SCOPE spreadsheet.

4. Choose Project->Implementation Options, then define the following:
  - Target architecture and technology specifications
  - Optimization options and design constraints
  - Outputs

For an initial run, use the default options settings for the technology, and no timing goal (Frequency = 0 MHz).

5. Synthesize the design by clicking the Run button.

This step performs logic synthesis. While synthesizing, the synthesis tool displays the status (Compiling... or Mapping...). You can monitor messages by checking the log file (View->View Log File) or in the Tcl window (View->Tcl Window). The log file contains reports with information on timing, usage, and net buffering.

If synthesis is successful, you see the message Done! or Done (warnings). If processing stops because of syntax errors or other design problems, you see the message Errors! displayed, along with the error status in the log file of the Tcl window. If the tool displays Done (warnings), there might be potential design problems to investigate.

6. After synthesis, do one of the following:

- If there were no synthesis warnings or error messages (Done!), analyze your results in the RTL and Technology views. You can then resynthesize with different implementation options, or use the synthesis results to simulate or place-and-route your design.
- If there were synthesis warnings (Done (warnings)) or error messages (Errors!), check them in the log file. From the log file, you can jump to the corresponding source code or display information on the specific error or warning. Correct all errors and any relevant warnings and then rerun synthesis.

## BEST Algorithms

The Behavior Extracting Synthesis Technology (BEST™) feature is the underlying proprietary technology that the synthesis tools use to extract and implement your design structures.

During synthesis, the BEST algorithms recognize high-level abstract structures like RAMs, ROMs, finite state machines (FSMs), and arithmetic operators, and maintain them, instead of converting the design entirely to the gate level. The BEST algorithms automatically map these high-level structures to technology-specific resources using module generators. For example, the algorithms map RAMs to target-specific RAMs, and adders to carry chains. The BEST algorithms also optimize hierarchy automatically.

## Projects and Implementations

Projects and implementations features are available in the Synplify Pro tool.

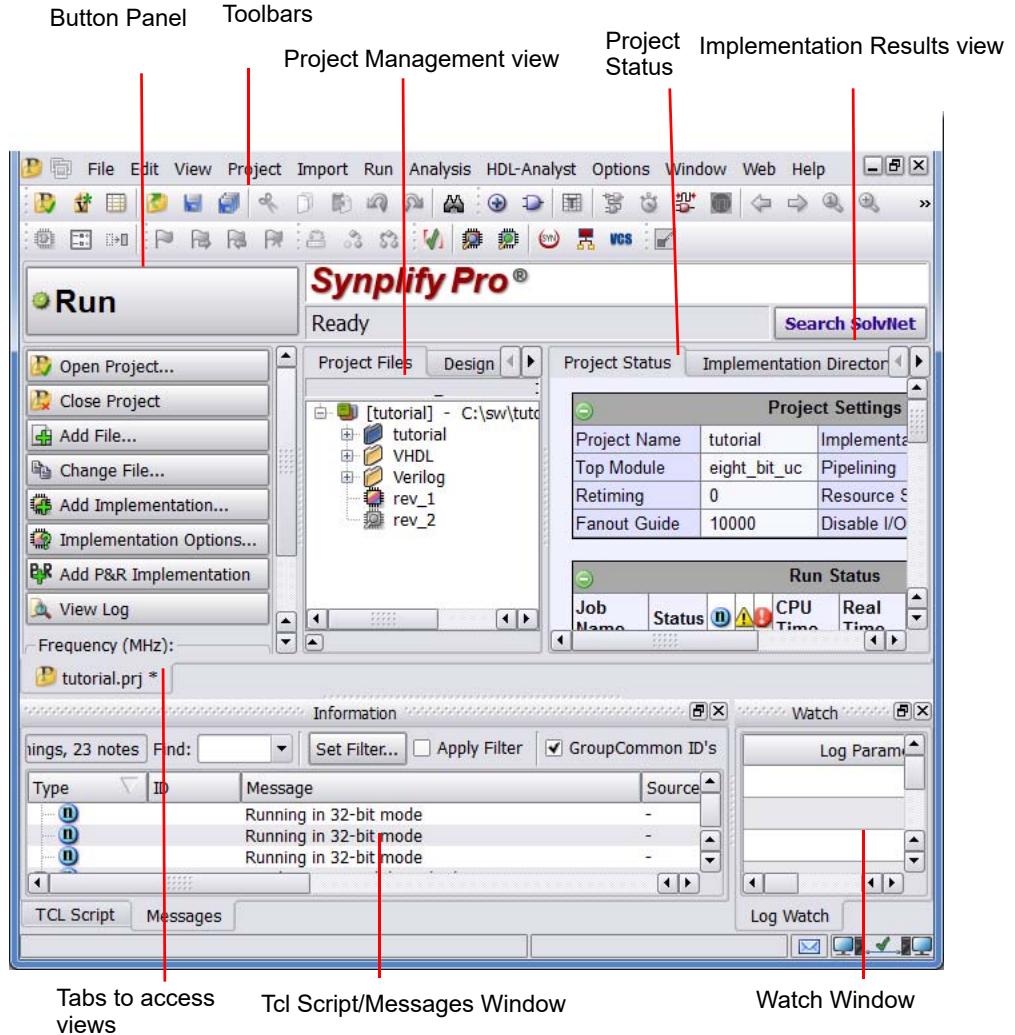
*Projects* contain information about the synthesis run, including the names of design files, constraint files (if used), and other options you have set. A *project file* (prj) is in Tcl format. It points to all the files you need for synthesis and contains the necessary optimization settings. In the Project view, a project appears as a folder.

An *implementation* is one version (also called a revision) of a project, run with certain parameter or option settings. You can synthesize again, with a different set of options, to get a different implementation. In the Project view, an implementation is shown in the folder of its project; the active implementation is highlighted. You can display multiple implementations in the same Project view. The output files generated for the active implementation are displayed in the Implementation Results view on the right.

# User Interface Overview

The graphical user interface (GUI) consists of a main window, called the Project view, and specialized windows or views for different tasks. For details about each of the features, see [Chapter 2, User Interface Overview](#) of the *Synopsys FPGA Synthesis Reference Manual*.

## Synplify Pro Interface





## CHAPTER 2

# FPGA Synthesis Design Flows

---

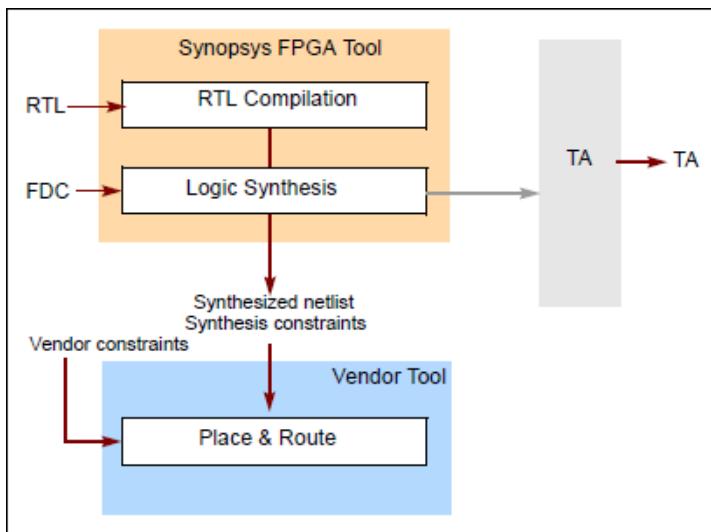
This describes the following tool flows:

- [Logic Synthesis Design Flow](#), on page 28

# Logic Synthesis Design Flow

The Synopsys FPGA tools synthesize logic by first compiling the RTL source into technology-independent logic structures, and then optimizing and mapping the logic to technology-specific resources. After logic synthesis, the tool generates a vendor-specific netlist and constraint file that you can use as inputs to the place-and-route (P&R) tool.

The following figure shows the phases and the tools used for logic synthesis and some of the major inputs and outputs. The interactive timing analysis step that is shown in gray is optional. Although the flow shows the vendor constraint files as direct inputs to the P&R tool, you should add these files to the synthesis project for timing black boxes.



## Logic Synthesis Procedure

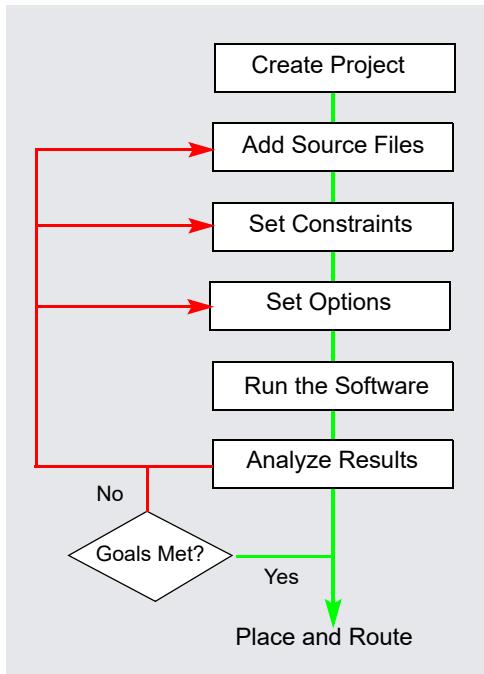
The following steps summarize the procedure for synthesizing the design, which is also illustrated in the figure that follows.

1. Create a project.
2. Add the source files to the project.
3. Set attributes and constraints for the design.

4. Set options for the implementation in the Implementation Options dialog box.
5. Click Run to run logic synthesis.
6. Analyze the results, using tools like the log file, the HDL Analyst schematic views, the Message window and the Watch Window.

After you have completed the design, you can use the output files to run place-and-route with the vendor tool and implement the FPGA.

The following figure lists the main steps in the flow:





## CHAPTER 3

# Preparing the Input

---

When you synthesize a design, you need to set up two kinds of files: HDL files that describe your design, and project files to manage the design. This chapter describes the procedures to set up these files and the project. It covers the following:

- [Setting Up HDL Source Files](#), on page 32
- [Using Mixed Language Source Files](#), on page 46
- [Working with Constraint Files](#), on page 51

# Setting Up HDL Source Files

This section describes how to set up your source files; project file setup is described in [Setting Up Project Files, on page 56](#). Source files can be in Verilog or VHDL. This section discusses the following topics:

- [Creating HDL Source Files, on page 32](#)
- [Using the Context Help Editor, on page 34](#)
- [Checking HDL Source Files, on page 36](#)
- [Editing HDL Source Files with the Built-in Text Editor, on page 37](#)
- [Setting Editing Window Preferences, on page 40](#)
- [Using an External Text Editor, on page 42](#)
- [Using Library Extensions for Verilog Library Files, on page 43](#)

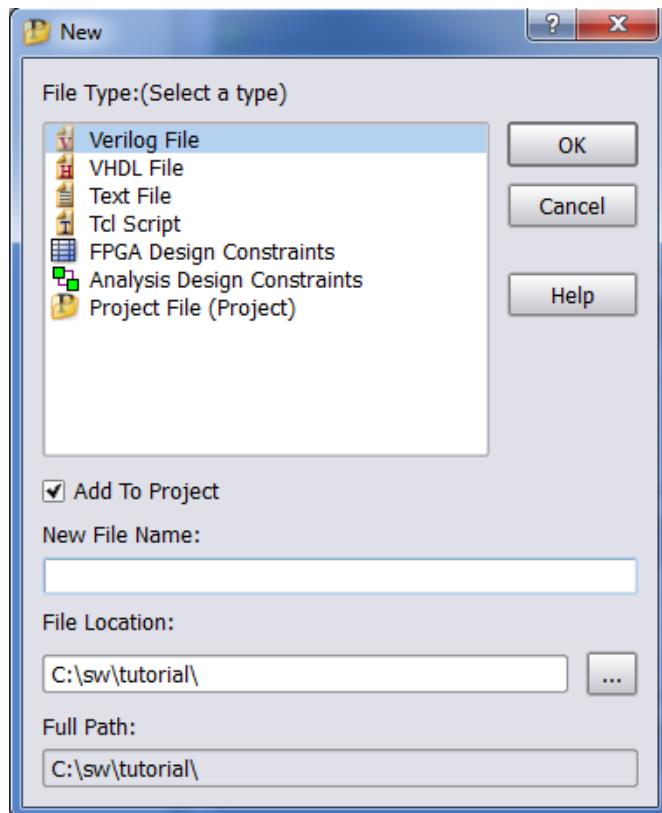
## Creating HDL Source Files

This section describes how to use the built-in text editor to create source files, but does not go into details of what the files contain. If you already have source files, you can use the text editor to check the syntax or edit the file (see [Checking HDL Source Files, on page 36](#) and [Editing HDL Source Files with the Built-in Text Editor, on page 37](#)).

You can use Verilog or VHDL for your source files. The files have v (Verilog) or vhd (VHDL) file extensions, respectively. With Synplify Pro, you can use Verilog and VHDL files in the same design. For information about using a mixture of Verilog and VHDL input files, see [Using Mixed Language Source Files, on page 46](#).

1. To create a new source file either click the HDL file icon () or do the following:
  - Select File->New or press Ctrl-n.
  - In the New dialog box, select the kind of source file you want to create, Verilog or VHDL.

If you are using Verilog 2001 format or SystemVerilog, make sure to enable the Verilog 2001 or System Verilog option before you run synthesis (Project->Implementation Options->Verilog tab). The default Verilog file format for new projects is SystemVerilog.



- Type a name and location for the file and Click OK. A blank editing window opens with line numbers on the left.

You can use the Context Help Editor for designs that contain Verilog, SystemVerilog, or VHDL constructs in the source file. For more information, see [Using the Context Help Editor, on page 34](#).

2. Type the source information in the window, or cut and paste it. See [Editing HDL Source Files with the Built-in Text Editor, on page 37](#) for more information on working in the Editing window.

For the best synthesis results, check the *Reference* manuals to ensure that you are using the available HDL constructs and vendor-specific attributes and directives effectively.

3. Save the file by selecting File->Save or the Save icon ().

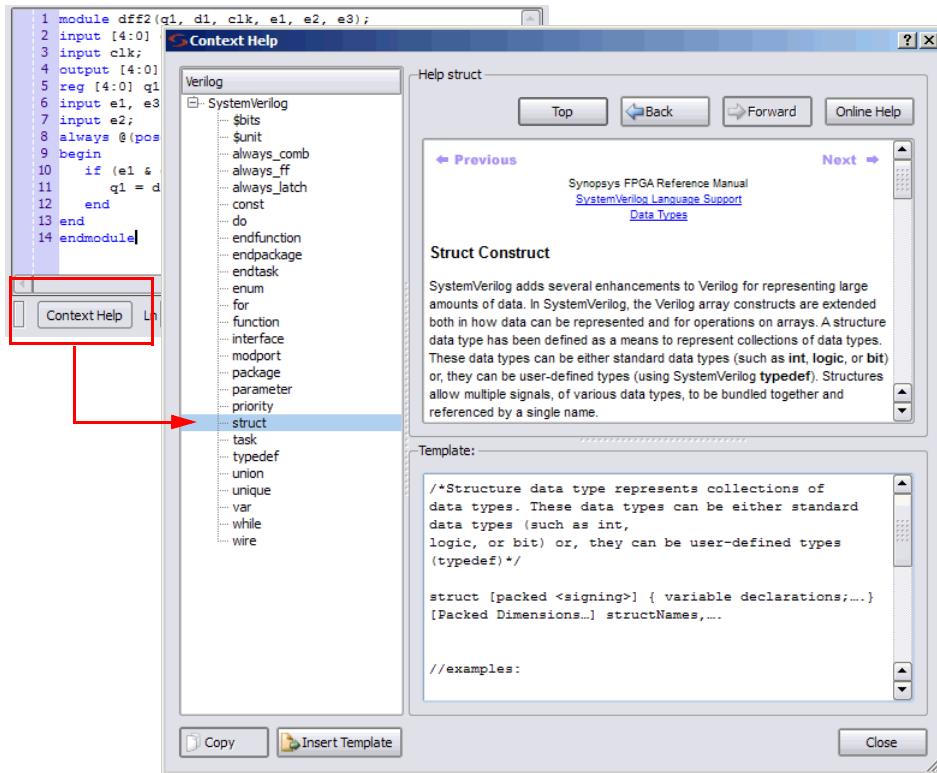
Once you have created a source file, you can check that you have the right syntax, as described in [Checking HDL Source Files, on page 36](#).

## Using the Context Help Editor

When you create or open a design file, use the Context Help button displayed at the bottom of the window to help you code with Verilog/SystemVerilog/VHDL constructs in the source file or Tcl constraint commands into your Tcl file.

To use the Context Help Editor:

1. Click the Context Help button to display this text editor.



2. When you select a construct in the left-side of the window, the online help description for the construct is displayed. If the selected construct has this feature enabled, the online help topic is displayed on the top of the window and a generic code or command template for that construct is displayed at the bottom.
3. The Insert Template button is also enabled. When you click the Insert Template button, the code or command shown in the template window is inserted into your file at the location of the cursor. This allows you to easily insert the code or command and modify it for the design that you are going to synthesize.

If you want to copy only parts of the template, select the code or command you want to insert and click Copy. You can then paste it into your file.

## Checking HDL Source Files

The software automatically checks your HDL source files when it compiles them, but if you want to check your source code before synthesis, use the following procedure. There are two kinds of checks you do in the synthesis software: syntax and synthesis.

1. Select the source files you want to check.
  - To check all the source files in a project, deselect all files in the project list, and make sure that none of the files are open in an active window. If you have an active source file, the software only checks the active file.
  - To check a single file, open the file with File->Open or double-click the file in the Project window. If you have more than one file open and want to check only one of them, put your cursor in the appropriate file window to make sure that it is the active window.
2. To check the syntax, select Run->Syntax Check or press Shift+F7.

The software detects syntax errors such as incorrect keywords and punctuation and reports any errors in a separate log file (syntax.log). If no errors are detected, a successful syntax check is reported at the bottom of this file.

3. To run a synthesis check, select Run->Synthesis Check or press Shift+F8.
4. Review the errors by opening the syntax.log file when prompted and use Find to locate the error message (search for @E). Double-click on the 5-character error code or click on the message text and push F1 to display online error message help.
5. Locate the portion of code responsible for the error by double-clicking on the message text in the syntax.log file. The Text Editor window opens the appropriate source file and highlights the code that caused the error.
6. Repeat steps 4 and 5 until all syntax and synthesis errors are corrected.

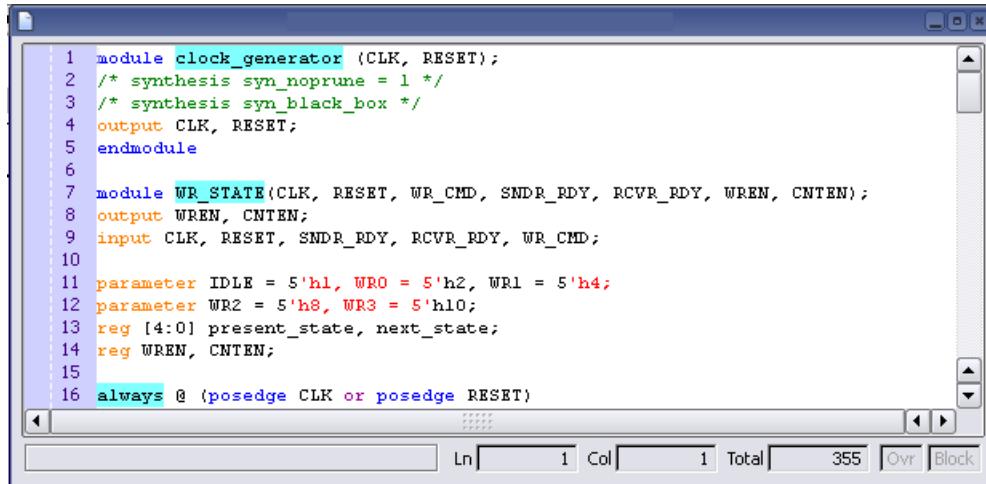
Messages can be categorized as errors, warnings, or notes. Review all messages and resolve any errors. Warnings are less serious than errors, but you must read through and understand them even if you do not resolve all of them. Notes are informative and do not need to be resolved.

## Editing HDL Source Files with the Built-in Text Editor

The built-in text editor makes it easy to create your HDL source code, view it, or edit it when you need to fix errors. If you want to use an external text editor, see [Using an External Text Editor, on page 42](#).

1. Do one of the following to open a source file for viewing or editing:
  - To automatically open the first file in the list with errors, press F5.
  - To open a specific file, double-click the file in the Project window or use File->Open (Ctrl-o) and specify the source file.

The Text Editor window opens and displays the source file. Lines are numbered. Keywords are in blue, and comments in green. String values are in red. If you want to change these colors, see [Setting Editing Window Preferences, on page 40](#).



A screenshot of the Synplify Pro GoWin Text Editor window. The window title is "Text Editor". The main area contains the following HDL source code:

```
1 module clock_generator (CLK, RESET);
2 /* synthesis syn_noprune = 1 */
3 /* synthesis syn_black_box */
4 output CLK, RESET;
5 endmodule
6
7 module WR_STATE(CLK, RESET, WR_CMD, SNDR_RDY, RCVR_RDY, WREN, CNTEN);
8 output WREN, CNTEN;
9 input CLK, RESET, SNDR_RDY, RCVR_RDY, WR_CMD;
10
11 parameter IDLE = 5'h1, WRO = 5'h2, WR1 = 5'h4;
12 parameter WR2 = 5'h8, WR3 = 5'h10;
13 reg [4:0] present_state, next_state;
14 reg WREN, CNTEN;
15
16 always @ (posedge CLK or posedge RESET)
```

The code uses color-coded syntax highlighting: blue for keywords like module, endmodule, output, and parameter; green for comments; and red for string values like 5'h1. The window has standard scroll bars and a status bar at the bottom with fields for Ln (line), Col (column), Total (355), Ovr (overshoot), and Block.

2. To edit a file, type directly in the window.

This table summarizes common editing operations you might use. You can also use the keyboard shortcuts instead of the commands.

To...	Do...
Cut, copy, and paste; undo, or redo an action	Select the command from the popup (hold down the right mouse button) or Edit menu.
Go to a specific line	Press Ctrl-g or select Edit->Go To, type the line number, and click OK.
Find text	Press Ctrl-f or select Edit ->Find. Type the text you want to find, and click OK.
Replace text	Press Ctrl-h or select Edit->Replace. Type the text you want to find, and the text you want to replace it with. Click OK.
Complete a keyword	Type enough characters to uniquely identify the keyword, and press Esc.
Indent text to the right	Select the block, and press Tab.
Indent text to the left	Select the block, and press Shift-Tab.
Change to upper case	Select the text, and then select Edit->Advanced ->Uppercase or press Ctrl-Shift-u.
Change to lower case	Select the text, and then select Edit->Advanced ->Lowercase or press Ctrl-u.
Add block comments	Put the cursor at the beginning of the comment text, and select Edit->Advanced->Comment Code or press Alt-c.
Edit columns	Press Alt, and use the left mouse button to select the column. On some platforms, you have to use the key to which the Alt functionality is mapped, like the Meta or diamond key.

3. To cut and paste a section of a PDF document, select the T-shaped Text Select icon, highlight the text you need and copy and paste it into your file. The Text Select icon lets you select parts of the document.
4. To create and work with bookmarks in your file, see the following table.

Bookmarks are a convenient way to navigate long files or to jump to points in the code that you refer to often. You can use the icons in the Edit toolbar for these operations. If you cannot see the Edit toolbar on the far right of your window, resize some of the other toolbars.

To...	Do...
Insert a bookmark	<p>Click anywhere in the line you want to bookmark.</p> <p>Select Edit-&gt;Toggle Bookmarks, press Ctrl-F2, or select the first icon in the Edit toolbar.</p> <p>The line number is highlighted to indicate that there is a bookmark at the beginning of that line.</p>
Delete a bookmark	<p>Click anywhere in the line with the bookmark.</p> <p>Select Edit-&gt;Toggle Bookmarks, press Ctrl-F2, or select the first icon in the Edit toolbar.</p> <p>The line number is no longer highlighted after the bookmark is deleted.</p>
Delete all bookmarks	<p>Select Edit-&gt;Delete all Bookmarks, press Ctrl-Shift-F2, or select the last icon in the Edit toolbar.</p> <p>The line numbers are no longer highlighted after the bookmarks are deleted.</p>
Navigate a file using bookmarks	<p>Use the Next Bookmark (F2) and Previous Bookmark (Shift-F2) commands from the Edit menu or the corresponding icons from the Edit toolbar to navigate to the bookmark you want.</p>

5. To fix errors or review warnings in the source code, do the following:
  - Open the HDL file with the error or warning by double-clicking the file in the project list.
  - Press F5 to go to the first error, warning, or note in the file. At the bottom of the Editing window, you see the message text.
  - To go to the next error, warning, or note, select Run->Next Error/Warning or press F5. If there are no more messages in the file, you see the message “No More Errors/Warnings/Notes” at the bottom of the Editing window. Select Run->Next Error/Warning or press F5 to go to the the error, warning, or note in the next file.
  - To navigate back to a previous error, warning, or note, select Run->Previous Error/Warning or press Shift-F5.

6. To bring up error message help for a full description of the error, warning, or note:
  - Open the text-format log file (click View Log) and either double click on the 5-character error code or click on the message text and press F1.
  - Open the HTML log file and click the 5-character error code.
  - In the Tcl window, click the Messages tab and click on the 5-character error code in the ID column.
7. To cross probe from the source code window to other views, open the view and select the piece of code. See [Crossprobing from the Text Editor Window, on page 306](#) for details.
8. When you have fixed all the errors, select File->Save or click the Save icon to save the file.

## Setting Editing Window Preferences

You can customize the fonts and colors used in a Text Editing window.

1. Select Options->Editor Options and either Synopsys Editor or External Editor. For more information about the external editor, see [Using an External Text Editor, on page 42](#).
2. Then depending on the type of file you open, you can set the background, syntax coloring, and font preferences to use with the text editor.

---

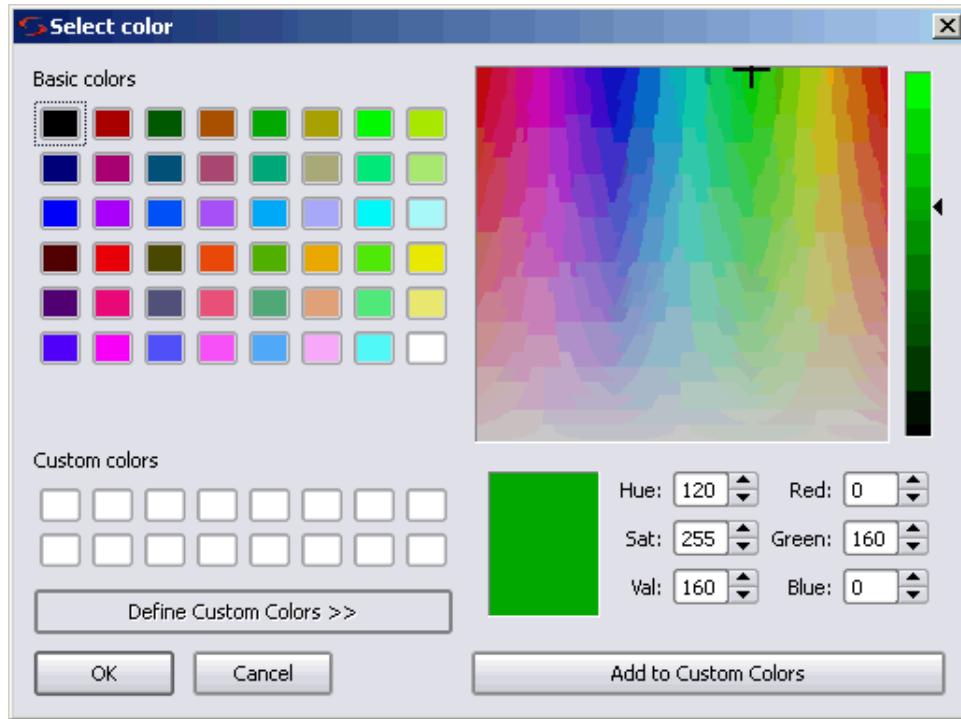
**Note:** Thereafter, text editing preferences you set for this file will apply to all files of this file type.

---

The Text Editing window can be used to set preferences for project files, source files (Verilog/VHDL), log files, Tcl files, constraint files, or other default files from the Editor Options dialog box.

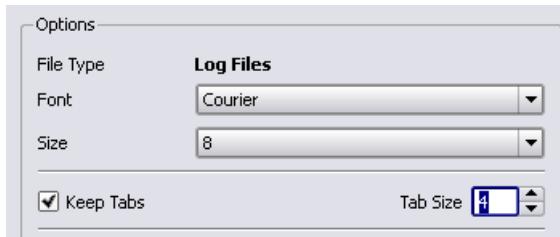
3. You can set syntax colors for some common syntax options, such as keywords, strings, and comments. For example in the log file, warnings and errors can be color-coded for easy recognition.

Click in the Foreground or Background field for the corresponding object in the Syntax Coloring field to display the color palette.



You can select basic colors or define custom colors and add them to your custom color palette. To select your desired color click OK.

4. To set font and font size for the text editor, use the pull-down menus.
5. Check Keep Tabs to enable tab settings, then set the tab spacing using the up or down arrow for Tab Size.

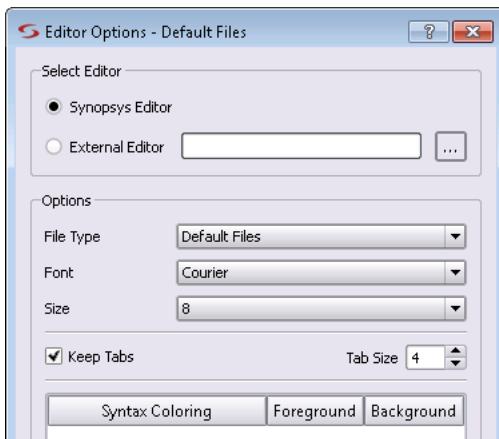


6. Click OK on the Editor Options form.

## Using an External Text Editor

You can use an external text editor like vi or emacs instead of the built-in text editor. Do the following to enable an external text editor. For information about using the built-in text editor, see [Editing HDL Source Files with the Built-in Text Editor, on page 37](#).

1. Select Options->Editor Options and turn on the External Editor option.
2. Select the external editor, using the method appropriate to your operating system.
  - If you are working on a Windows platform, click the ... (Browse) button and select the external text editor executable.
  - From a UNIX or Linux platform for a text editor that creates its own window, click the ... Browse button and select the external text editor executable.
  - From a UNIX platform for a text editor that does not create its own window, do not use the ... Browse button. Instead type xterm -e *editor*. The following figure shows VI specified as the external editor.



- From a Linux platform, for a text editor that does not create its own window, do not use the ... Browse button. Instead, type gnome-terminal -x *editor*. To use emacs for example, type gnome-terminal -x emacs.

The software has been tested with the emacs and vi text editors.

3. Click OK.

## Using Library Extensions for Verilog Library Files

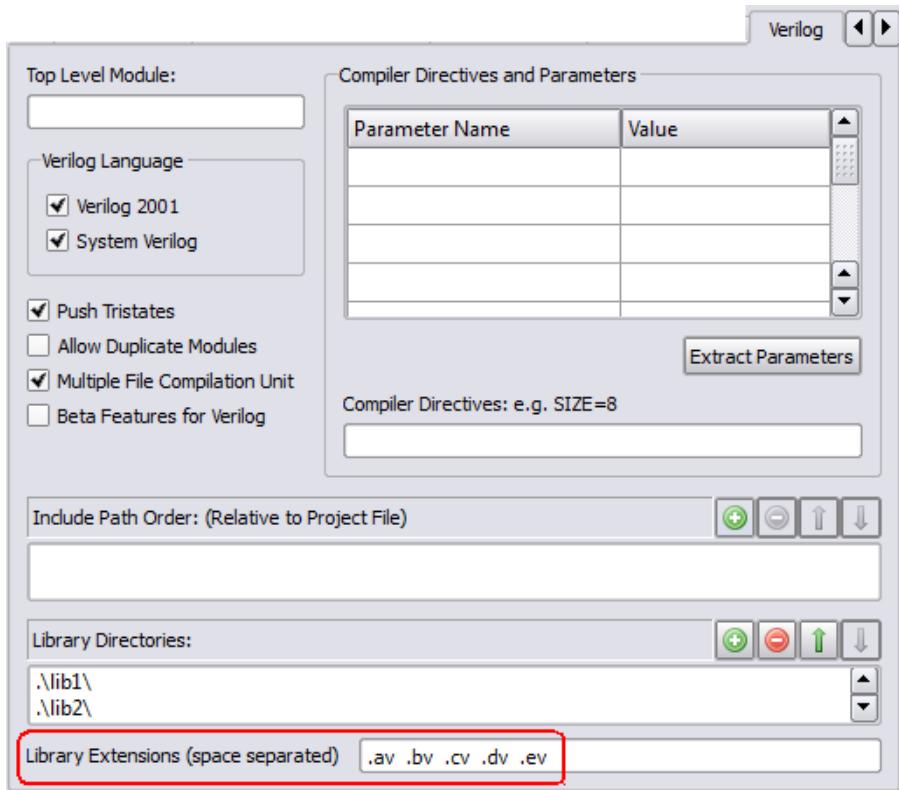
Library extensions can be added to Verilog library files included in your design for the project. When you provide search paths to the directories that contain the Verilog library files, you can specify these new library extensions as well as the Verilog and SystemVerilog (.v and .sv) file extensions.

To do this:

1. Select the Verilog tab of the Implementation Options panel.
2. Specify the locations of the Library Directories for the Verilog library files to be included in your design for the project.
3. Specify the Library Extensions.

Any library extensions can be specified, such as .av, .bv, .cv, .xxx, .va, .vas (separate library extensions with a space).

The following figure shows you where to enter the library extensions on the dialog box.



The Tcl equivalent for this example is the following command:

```
set_option -libext .av .bv .cv .dv .ev
```

For details, see [libext, on page 90](#) in the *Command Reference*.

- After you compile the design, you can verify in the log file that the library files with these extensions were loaded and read. For example:

```
@N: Running Verilog Compiler in SystemVerilog mode
@I::"C:\dir\top.v"
@N: CG1180 :"C:\dir\top.v":8:0:8:3 | Loading file
C:\dir\lib1\sub1.av from specified library directory C:\dir\lib1
@I::"C:\dir\lib1\sub1.av"
@N: CG1180 :"C:\dir\top.v":10:0:10:3 | Loading file
C:\dir\lib2\sub2.bv from specified library directory C:\dir\lib2
@I::"C:\dir\lib2\sub2.bv"
```

```
@N: CG1180 :"C:\dir\top.v":12:0:12:3 | Loading file  
C:\dir\lib3\sub3.cv from specified library directory C:\dir\lib3  
@I:"C:\dir\lib3\sub3.cv"  
@N: CG1180 :"C:\dir\top.v":14:0:14:3 | Loading file  
C:\dir\lib4\sub4.dv from specified library directory C:\dir\lib4  
@I:"C:\dir\lib4\sub4.dv"  
@N: CG1180 :"C:\dir\top.v":16:0:16:3 | Loading file  
C:\dir\lib5\sub5.ev from specified library directory C:\dir\lib5  
@I:"C:\dir\lib5\sub5.ev"  
Verilog syntax check successful!
```

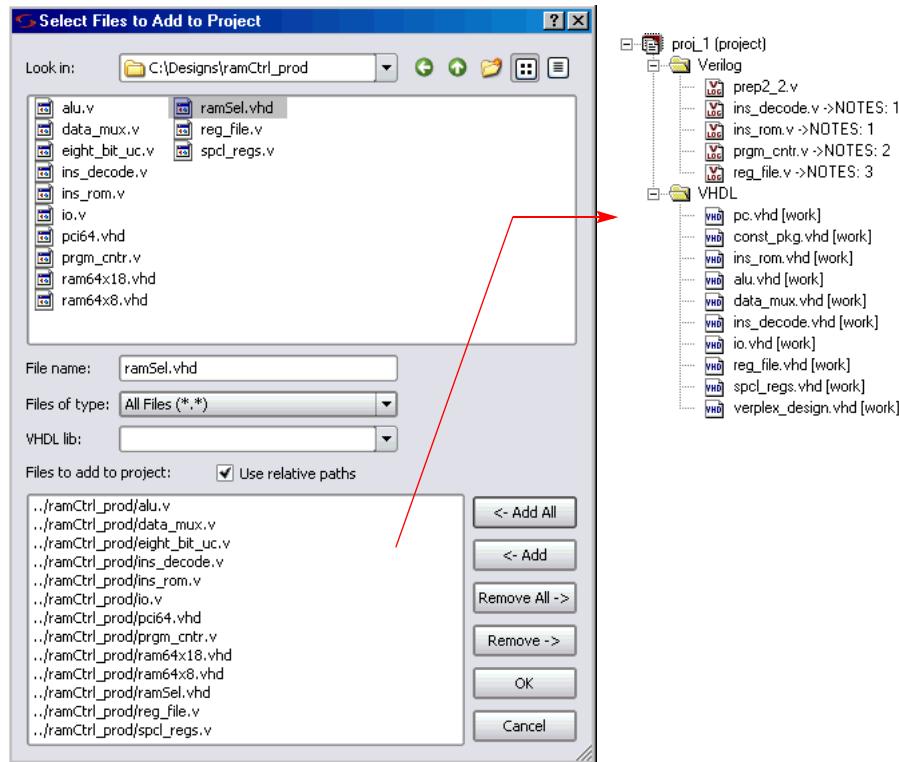
# Using Mixed Language Source Files

With the synthesis software, you can use a mixture of VHDL and Verilog input files in your project. For examples of the VHDL and Verilog files, see the *Reference Manual*. You cannot use Verilog and VHDL files together in the same design with the Synplify tool.

1. Remember that Verilog does not support unconstrained VHDL ports and set up the mixed language design files accordingly.
2. If you want to organize the Verilog and VHDL files in different folders, select Options->Project View Options and toggle on the View Project Files in Folders option.

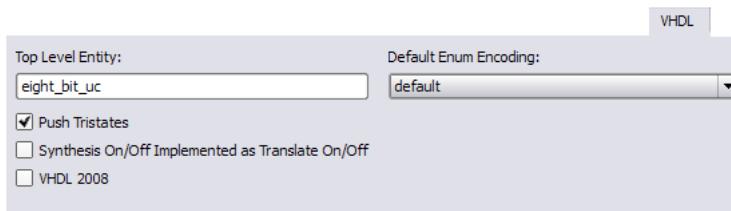
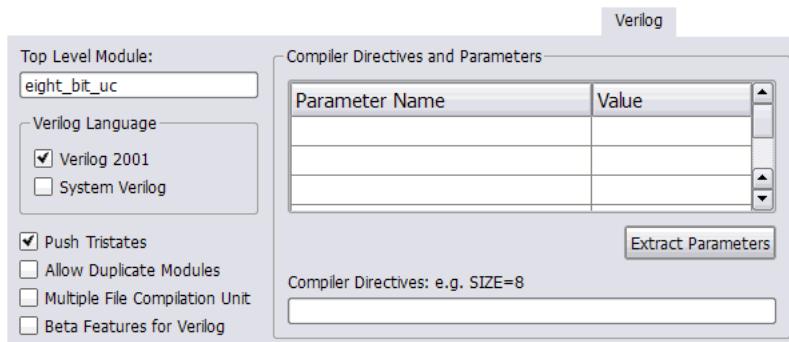
When you add the files to the project, the Verilog and VHDL files are in separate folders in the Project view.

3. When you open a project or create a new one, add the Verilog and VHDL files as follows:
  - Select the Project->Add Source File command or click the Add File button.
  - On the form, set Files of Type to HDL Files (\*.vhd, \*.vhdl, \*.v).
  - Select the Verilog and VHDL files you want and add them to your project. Click OK. For details about adding files to a project, see [Making Changes to a Project, on page 60](#).



The files you added are displayed in the Project view. This figure shows the files arranged in separate folders.

4. When you set device options (Implementation Options button), specify the top-level module. For more information about setting device options, see [Setting Logic Synthesis Implementation Options, on page 74](#).
  - If the top-level module is Verilog, click the Verilog tab and type the name of the top-level module.
  - If the top-level module is VHDL, click the VHDL tab and type the name of the top-level entity. If the top-level module is not located in the default work library, you must specify the library where the compiler can find the module. For information on how to do this, see [VHDL Panel, on page 323](#).



You must explicitly specify the top-level module, because it is the starting point from which the mapper generates a merged netlist.

5. Select the Implementation Results tab on the same form and select one output HDL format for the output files generated by the software. For more information about setting device options, see [Setting Logic Synthesis Implementation Options, on page 74](#).

- For a Verilog output netlist, select Write Verilog Netlist.
- For a VHDL output netlist, select Write VHDL Netlist.
- Set any other device options and click OK.

You can now synthesize your design. The software reads in the mixed formats of the source files and generates a single srs file that is used for synthesis.

6. If you run into problems, see [Troubleshooting Mixed Language Designs, on page 49](#) for additional information and tips.

## Troubleshooting Mixed Language Designs

This section provides tips on handling specific situations that might come up with mixed language designs.

### VHDL File Order

Correct file order is important, especially for VHDL files.

- Make sure the files are ordered correctly. To re-order files from the UI, drag files to their correct locations in the order. Alternatively, use the `add_file` command in the project file to add the input files in the correct sequence. See [Ordering Input Files, on page 63](#) for details about the order sequence for Verilog and VHDL.
- In the Project view, check that the last file in the Project view is the top-level source file. Alternatively, you can specify the top-level file when you set the device options.

An example of correct file order for a design that consists of one top module (`top.v`) and two sub module files (`module1.v` and `module2.v`):

```
add_file -verilog -lib work module1.v
add_file -verilog -lib work module2.v
add_file -verilog -lib work top.v
```

For more information about the file order, see [Ordering Input Files, on page 63](#).

### VHDL Global Signals

Currently, you cannot have VHDL global signals in mixed language designs, because the tool only implements these signals in VHDL-only designs.

### Passing VHDL Boolean Generics to Verilog Parameters

The tool infers a black box for a VHDL component with Boolean generics, if that component is instantiated in a Verilog design. This is because Verilog does not recognize Boolean data types, so the Boolean value must be represented correctly. If the value of the VHDL Boolean generic is TRUE and the Verilog literal is represented by a 1, the Verilog compiler interprets this as a black box.

To avoid inferring a black box, the Verilog literal for the VHDL Boolean generic set to TRUE must be 1'b1, not 1. Similarly, if the VHDL Boolean generic is FALSE, the corresponding Verilog literal must be 1'b0, not 0. The following example shows how to represent Boolean generics so that they correctly pass the VHDL-Verilog boundary, without inferring a black box.

VHDL Entity Declaration	Verilog Instantiation
Entity abc is Generic ( Number_Bits : integer := 0; Divide_Bit : boolean := False; ) ;	abc #( .Number_Bits (16), .Divide_Bit (1'b0) )

## Passing VHDL Generics Without Inferring a Black Box

In the case where a Verilog component parameter, (for example [0:0] RSR = 1'b0) does not match the size of the corresponding VHDL component generic (RSR : integer := 0), the tool infers a black box.

You can work around this by removing the bus width notation of [0:0] in the Verilog files. You must use a VHDL generic of type integer because the other types do not allow for the proper binding of the Verilog component.

# Working with Constraint Files

Constraint files are text files that are automatically generated by the SCOPE interface (see [Specifying SCOPE Constraints, on page 121](#)), or which you create manually with a text editor. They contain Tcl commands or attributes that constrain the synthesis run. Alternatively, you can set constraints in the source code, but this is not the preferred method.

This section contains information about

- [When to Use Constraint Files over Source Code, on page 51](#)
- [Tcl Syntax Guidelines for Constraint Files, on page 52](#)
- [Checking Constraint Files, on page 53](#)

## When to Use Constraint Files over Source Code

You can add constraints in constraint files (generated by the SCOPE interface or entered in a text editor) or in the source code. In general, it is better to use constraint files, because you do not have to recompile for the constraints to take effect. It also makes your source code more portable. See [Using the SCOPE Editor, on page 114](#) for more information.

However, if you have black box timing constraints like syn\_tco, syn\_tpd, and syn\_tsu, you must enter them as directives in the source code. Unlike attributes, directives can only be added to the source code, not to constraint files. See [Specifying Attributes and Directives, on page 89](#) for more information on adding directives to source code.

## Tcl Syntax Guidelines for Constraint Files

This section covers general guidelines for using Tcl for constraint files:

- Tcl is case-sensitive.
- For naming objects:
  - The object name must match the name in the HDL code.
  - Enclose instance and port names within curly braces {}.
  - Do not use spaces in names.
  - Use the dot (.) to separate hierarchical names.
  - In Verilog modules, use the following syntax for instance, port, and net names:

**v:cell|[prefix:]objectName**

Where *cell* is the name of the design entity, *prefix* is a prefix to identify objects with the same name, *objectName* is an instance path with the dot (.) separator. The prefix can be any of the following:

Prefix (Lower-case)	Object
i:	Instance names
p:	Port names (entire port)
b:	Bit slice of a port
n:	Net names

- In VHDL modules, use the following syntax for instance, port, and net names in VHDL modules:

**v:cell[.view] [prefix:]objectName**

Where v: identifies it as a view object, lib is the name of the library, *cell* is the name of the design entity, *view* is a name for the architecture, *prefix* is a prefix to identify objects with the same name, and *objectName* is an instance path with the dot (.) separator. *View* is only needed if there is more than one architecture for the design. See the table above for the prefixes of objects.

- Name matching wildcards are \* (asterisk matches any number of characters) and ? (question mark matches a single character). These characters do not match dots used as hierarchy separators. For example, the following string identifies all bits of the statereg instance in the statemod module:

```
i:statemod.statereg[*]
```

## Checking Constraint Files

You can check syntax and other pertinent information for your constraint files using the Constraint Check command. To generate a constraint report, do the following:

1. Create a constraint file and add it to your project.
2. Select Run->Constraint Check.

This command generates a report that checks the syntax and applicability of the timing constraints in the FPGA synthesis constraint files for your project. The report is written to the *projectName\_cck.rpt* file and lists the following information:

- Constraints that are not applied
- Constraints that are valid and applicable to the design
- Wildcard expansion on the constraints
- Constraints on objects that do not exist

For details on this report, see [Constraint Checking Report, on page 159](#) of the *Reference Manual*.



## CHAPTER 4

# Setting Up a Logic Synthesis Project

---

When you synthesize a design with the Synopsys FPGA synthesis tools, you must set up a project for your design. The following describe the procedures for setting up a project for logic synthesis:

- [Setting Up Project Files](#), on page 56
- [Managing Project File Hierarchy](#), on page 65
- [Setting Up Implementations](#), on page 72
- [Setting Logic Synthesis Implementation Options](#), on page 74
- [Specifying Attributes and Directives](#), on page 89
- [Searching Files](#), on page 97
- [Archiving Files and Projects](#), on page 100

# Setting Up Project Files

This section describes the basics of how to set up and manage a project file for your design, including the following information:

- [Creating a Project File](#), on page 56
- [Opening an Existing Project File](#), on page 59
- [Making Changes to a Project](#), on page 60
- [Setting Project View Display Preferences](#), on page 61
- [Updating Verilog Include Paths in Older Project Files](#), on page 64

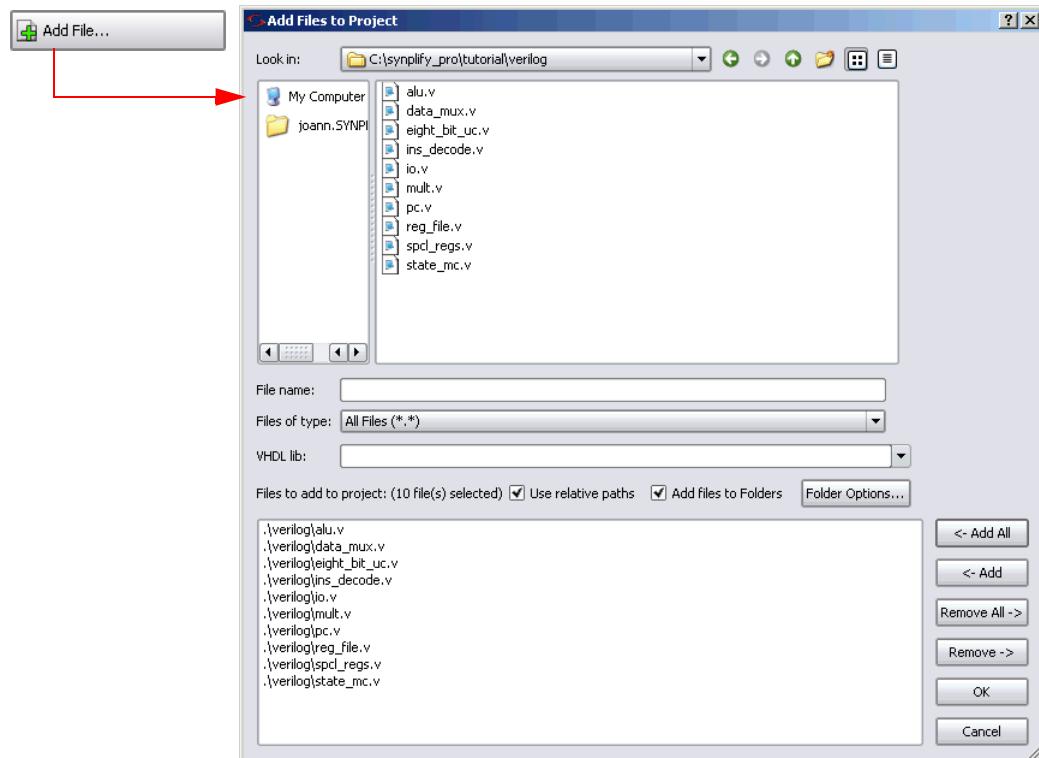
## Creating a Project File

You must set up a project file for each project. A project contains the data needed for a particular design: the list of source files, the synthesis results file, and your device option settings. The following procedure shows you how to set up a project file using individual commands.

1. Start by selecting one of the following: File->Build Project, File->Open Project, or the P icon. Click New Project.

The Project window shows a new project. Click the Add File button, press shift F4, or select the Project->Add Source File command. The Add Files to Project dialog box opens.

2. Add the source files to the project.
  - Make sure the Look in field at the top of the form points to the right directory. The files are listed in the box. If you do not see the files, check that the Files of Type field is set to display the correct file type. If you have mixed input files, follow the procedure described in [Using Mixed Language Source Files](#), on page 46.



- To add all the files in the directory at once, click the Add All button on the right side of the form. To add files individually, click on the file in the list and then click the Add button, or double-click the file name.

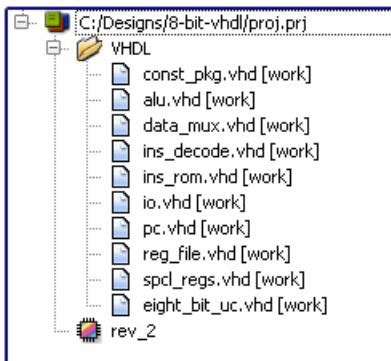
You can add all the files in the directory and then remove the ones you do not need with the Remove button.

If you are adding VHDL files, select the appropriate library from the VHDL Library popup menu. The library you select is applied to all VHDL files when you click OK in the dialog box.

Your project window displays a new project file. If you click on the plus sign next to the project and expand it, you see the following:

- A folder (two folders for mixed language designs) with the source files. If your files are not in a folder under the project directory, you can set this preference by selecting Options->Project View Options and checking the View Project Files in Type Folders box. This separates one kind of file from another in the Project view by putting them in separate folders.

- The implementation, named `rev_1` by default. Implementations are revisions of your design within the context of the synthesis software, and do not replace external source code control software and processes. Multiple implementations let you modify device and synthesis options to explore design options. Each implementation has its own synthesis and device options and its own project-related files.



3. Add any libraries you need, using the method described in the previous step to add the Verilog or VHDL library file.

- For GoWin libraries, add the appropriate library file to the project. Note that for some families, the libraries are loaded automatically and you do not need to explicitly add them to the project file.

To add a third-party VHDL package library, add the appropriate .vhd file to the design, as described in step 2. Right-click the file in the Project view and select File Options, or select Project-> Set VHDL library. Specify a library name that is compatible with the simulators. For example, MYLIB. Make sure that this package library is before the top-level design in the list of files in the Project view.

For information about setting Verilog and VHDL file options, see [Setting Verilog and VHDL Options, on page 83](#). You can also set these file options later, before running synthesis.

For additional information about using vendor macro libraries and black boxes, see [Optimizing for GoWin Designs, on page 489](#).

- For generic technology components, you can either add the technology-independent Verilog library supplied with the software (`install_dir/lib/generic_technology/gtech.v`) to your design, or add your

own generic component library. Do not use both together as there may be conflicts.

4. Check file order in the Project view. File order is important for all HDL files.
  - Make sure the files are ordered correctly. To re-order files from the UI, drag files to their correct locations in the order. Alternatively, use the `add_file` command in the project file to add the input files in the correct sequence. See [Ordering Input Files, on page 63](#) for details about the order sequence for Verilog and VHDL files.
  - In the Project view, check that the last file in the Project view is the top-level source file. Alternatively, you can specify the top-level file when you set the device options.

An example of correct file order for a design that consists of one top module (`top.v`) and two sub module files (`module1.v` and `module2.v`):

```
add_file -verilog -lib work module1.v
add_file -verilog -lib work module2.v
add_file -verilog -lib work top.v
```

5. Select **File->Save**, type a name for the project, and click **Save**. The Project window reflects your changes.
6. To close a project file, select the **Close Project** button or **File->Close Project**.

## Opening an Existing Project File

There are two ways to open a project file: the **Open Project** and the generic **File->Open** command.

1. If the project you want to open is one you worked on recently, you can select it directly: **File->Recent Projects-> projectName**.
2. Use one of the following methods to open any project file:

Open Project Command	File->Open Command
Select File->Open Project, click the Open Project button on the left side of the Project window, or click the P icon.  To open a recent project, double-click it from the list of recent projects.  Otherwise, click the Existing Project button to open the Open dialog box and select the project.	Select File->Open.  Specify the correct directory in the Look In: field.  Set File of Type to Project Files (*.prj). The box lists the project files.  Double-click the project you want to open.

The project opens in the Project window.

## Making Changes to a Project

Typically, you add, delete, or replace files.

1. To add source or constraint files to a project, select the Add Files button or Project->Add Source File to open the Select Files to Add to Project dialog box. See [Creating a Project File, on page 56](#) for details.
2. To delete a file from a project, click the file in the Project window, and press the Delete key.
3. To replace a file in a project,
  - Select the file you want to change in the Project window.
  - Click the Change File button, or select Project->Change File.
  - In the Source File dialog box that opens, set Look In to the directory where the new file is located. The new file must be of the same type as the file you want to replace.
  - If you do not see your file listed, select the type of file you need from the Files of Type field.
  - Double-click the file. The new file replaces the old one in the project list.
4. To specify how project files are saved in the project, right click on a file in the Project view and select File Options. Set the Save File option to either Relative to Project or Absolute Path.

5. To check the time stamp on a file, right click on a file in the Project view and select File Options. Check the time that the file was last modified. Click OK.

## Setting Project View Display Preferences

You can customize the organization and display of project files.

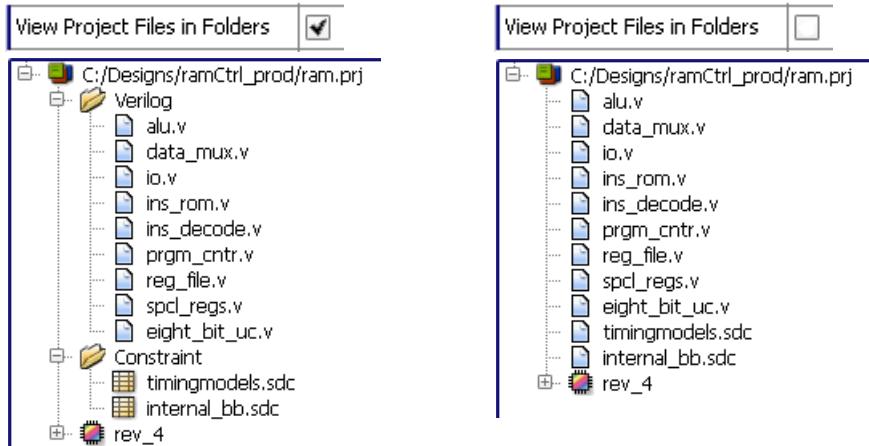
1. Select Options->Project View Options.

The Project View Options form opens.



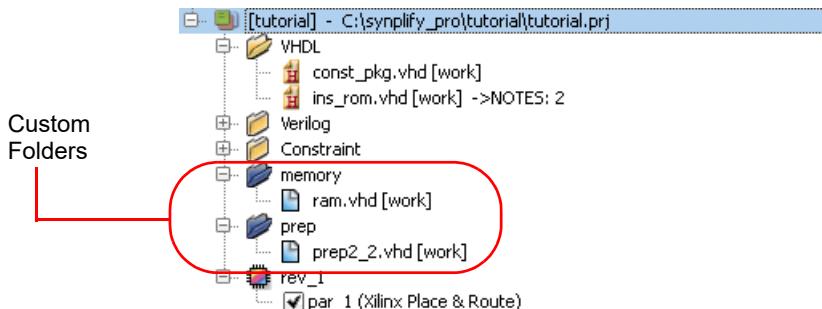
2. To organize different kinds of input files in separate folders, check View Project Files in Folders.

Checking this option creates separate folders in the Project view for constraint files and source files.

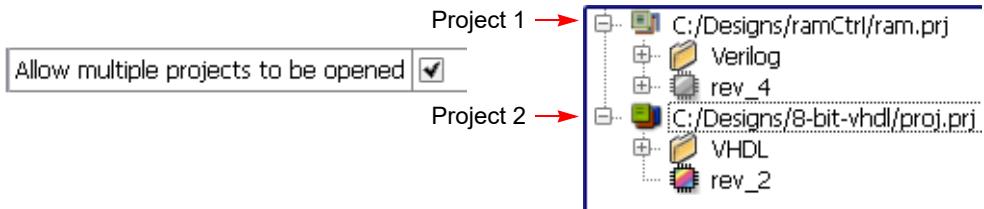


3. Control file display with the following:

- Automatically display all the files, by checking Show Project Library. If this is unchecked, the Project view does not display files until you click the plus symbol and expand the files in a folder.
  - Check one of the boxes in the Project File Name Display section of the form to determine how filenames are displayed. You can display just the filename, the relative path, or the absolute path.
4. To view project files in customized custom folders, check View Project Files in Custom Folders. For more information, see [Creating Custom Folders, on page 65](#). Type folders are only displayed if there are multiple types in a custom folder.



- To open more than one implementation in the same Project view, check Allow Multiple Projects to be Opened.



- Control the output file display with the following:
  - Check the Show all Files in Results Directory box to display all the output files generated after synthesis.
  - Change output file organization by clicking in one of the header bars in the Implementation Results view. You can group the files by type or sort them according to the date they were last modified.
- To view file information, select the file in the Project view, right-click, and select File Options. For example, you can check the date a file was modified.

## Ordering Input Files

Correct file order is important for all HDL designs, to ensure that the packages are compiled quickly. Incorrect file order causes unwanted errors/warnings and longer run times. The general rule is to instantiate macros or packages that are referenced by other files and list them before the files that instantiate them. The top-level file must be last on the list.

Do the following to correctly arrange VHDL files:

- To automatically arrange the VHDL files, select the Run->Arrange VHDL option from the menu. Run this command only once.
- To manually arrange the VHDL files, make sure to list the package files first because they must be compiled. Then follow this file order for big designs spread out over multiple FPGAs: entity file, then the architecture file and lastly the configuration file.

Follow this file order when specifying Verilog files for a design:

- For Verilog designs, list package files first because they are compiled before use. After that, list the corresponding HDL files, and list the top-level source file last.
- For SystemVerilog designs, ensure that package, macro, and component files are listed first. Next add the files that instantiate the packages and macros. For example:
  - Package1.sv (package file)
  - Test.sv (import Package1.sv)
  - Top.sv (instantiates Test.sv)

Do the following to arrange files for mixed designs:

- Arrange Verilog/System Verilog files are described above
- Use the Run->Arrange VHDL option from the menu for VHDL files.

## Updating Verilog Include Paths in Older Project Files

If you have a project file created with an older version of the software (prior to 8.1), the Verilog include paths in this file are relative to the results directory or the source file with the `include statements. In releases after 8.1, the project file `include paths are relative to the project file only. The GUI in the more recent releases does not automatically upgrade the older PRJ files to conform to the newer rules. To upgrade and use the old project file, do one of the following:

- Manually edit the PRJ file in a text editor and add the following on the line before each set\_option -include\_path:

```
set_option -project_relative_includes 1
```

- Start a new project with a newer version of the software and delete the old project. This will make the new PRJ file obey the new rule where includes are relative to the PRJ file.

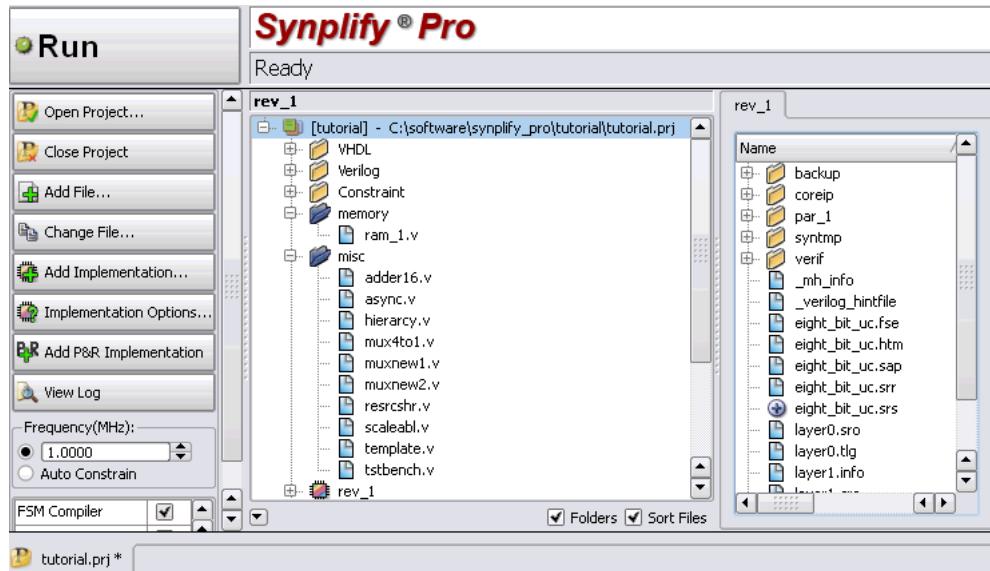
# Managing Project File Hierarchy

The following sections describe how you can create and manage customized folders and files in the Project view:

- [Creating Custom Folders](#)
- [Manipulating Custom Project Folders](#)
- [Manipulating Custom Files](#)

## Creating Custom Folders

You can create logical folders and customize files in various hierarchy groupings within your Project view. These folders can be specified with any name or hierarchy level. For example, you can arbitrarily match your operating system file structure or HDL logic hierarchy. Custom folders are distinguished by their blue color.



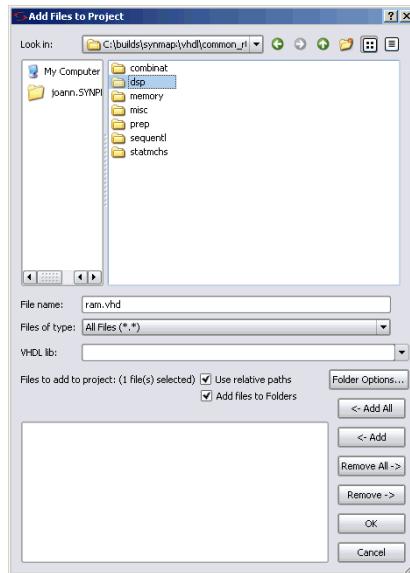
There are several ways to create custom folders and then add files to them in a project. Use one of the following methods:

1. Right-click on a project file or another custom folder and select Add Folder from the popup menu. Then perform any of the following file operations:
  - Right-click on a file or files and select Place in Folder. A sub-menu displays so that you can either select an existing folder or create a new folder.

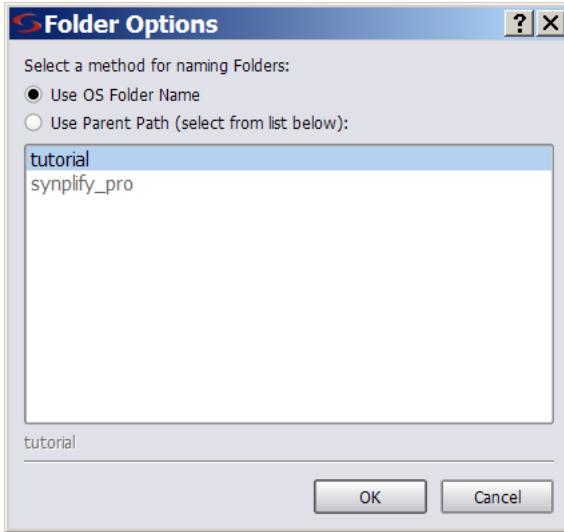


Note that you can arbitrarily name the folder, however do not use the character (/) because this is a hierarchy separator symbol.

- To rename a folder, right-click on the folder and select Rename from the popup menu. The Rename Folder dialog box appears; specify a new name.
2. Use the Add Files to Project dialog box to add the entire contents of a folder hierarchy, and optionally place files into custom folders corresponding to the OS folder hierarchies listed in the dialog box display.



- To do this, select the Add File button in the Project view.
- Select any requested folders such as dsp from the dialog box, then click the Add button. This places all the files from the dsp hierarchy into the custom folder you just created.
- To automatically place the files into custom folders corresponding to the OS folder hierarchy, check the option called Add Files to Custom Folders on the dialog box.
- By default, the custom folder name is the same name as the folder containing files or folder to be added to the project. However, you can modify how folders are named, by clicking on the Folders Option button. The following dialog box is displayed.



To use:

- Only the folder containing files for the folder name, click on Use OS Folder Name.
  - The path name to the selected folder to determine the level of hierarchy reflected for the custom folder path.
3. You can drag and drop files and folders from an OS Explorer application into the Project view. This feature is available on Windows and Linux desktops running KDE.
- When you drag and drop a file, it is immediately added to the project. If no project is open, the software creates a project.
  - When you drag and drop a file over a folder, it will be placed in that folder. Initially, the Add Files to Project dialog box is displayed asking you to confirm the files to be added to the project. You can click OK to accept the files. If you want to make changes, you can click the Remove All button and specify a new filter or option.

---

**Note:** To display custom folders in the Project view, select the Options->Project View Options menu, then enable/disable the check box for View Project Files in Custom Folders on the dialog box.

---

## Manipulating Custom Project Folders

The following procedure describes how you can remove files from folders, delete folders, and change the folder hierarchy.

1. To remove a file from a custom folder, either:
  - Drag and drop it into another folder or onto the project.
  - Highlight the file, right-click and select Remove from Folder from the popup menu.

Do not use the Delete (DEL) key, as this removes the file from the project.
2. To delete a custom folder, highlight it then right-click and select Delete from the popup menu or press the DEL key. When you delete a folder, make one of the following choices:
  - Click Yes to delete the folder and the files contained in the folder from the project.
  - Click No to just delete the folder.
3. To change the hierarchy of the custom folder:
  - Drag and drop the folder within another folder so that it is a sub-folder or over the project to move it to the top-level.
  - To remove the top-level hierarchy of a custom folder, drag and drop the desired sub-level of hierarchy over the project. Then delete the empty root directory for the folder.

For example, if the existing custom folder directory is:

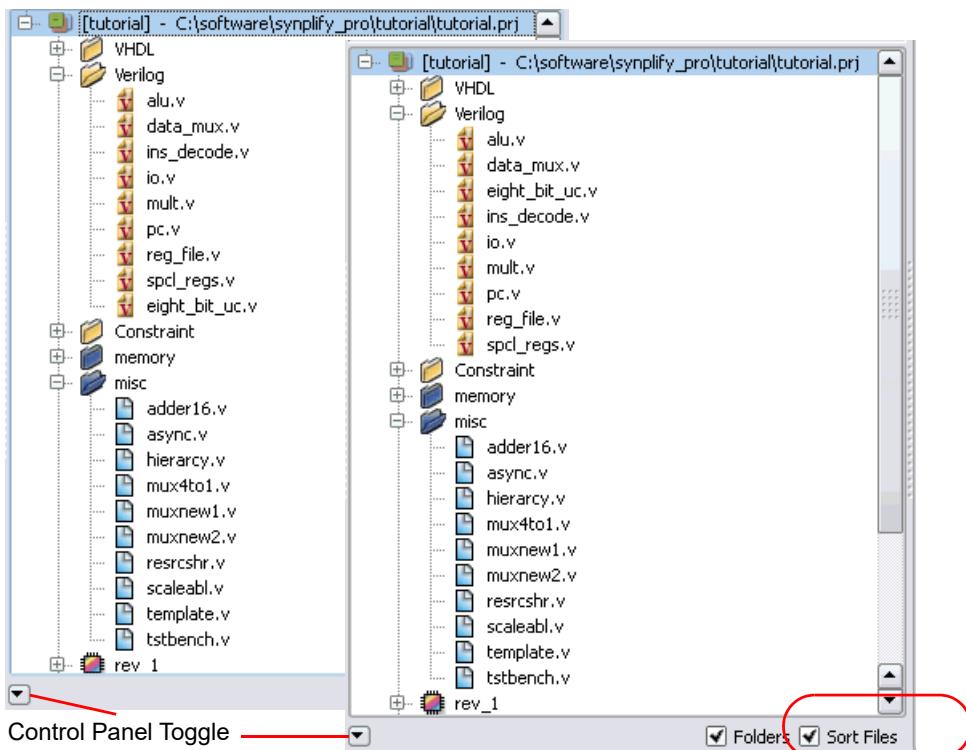
`/Examples/Verilog/RTL`

Suppose you want a single-level RTL hierarchy only, then drag and drop RTL over the project. Thereafter, you can delete the `/Examples/Verilog` directory.

## Manipulating Custom Files

Additionally, you can perform the following types of custom file operations:

1. To suppress the display of files in the Type folders, right-click in the Project view and select Project View Options or select Options->Project View Options. Disable the option View Project Files in Type Folders on the dialog box.
2. To display files in alphabetical order instead of project order, check the Sort Files button in the Project view control panel. Click the down arrow key in the bottom-left corner of the panel to toggle the control panel on and off.



3. To change the order of files in the project:
  - Make sure to disable custom folders and sorting files.
  - Drag and drop a file to the desired position in the list of files.
4. To change the file type, drag and drop it to the new type folder. The software will prompt you for verification.

# Setting Up Implementations

An implementation is a version of a project, implemented with a specific set of constraints and other settings. A project can contain multiple implementations, each one with its own settings.

## Working with Multiple Implementations

The synthesis tools let you create multiple implementations of the same design and then compare results. This lets you experiment with different settings for the same design. Implementations are revisions of your design within the context of the synthesis software, and do not replace external source code control software and processes.

1. Click the Add Implementation button or select Project->New Implementation and set new device options (Device tab), new options (Options tab), or a new constraint file (Constraints tab).

The software creates another implementation in the project view. The new implementation has the same name as the previous one, but with a different number suffix. The following figure shows two implementations, rev1 and rev2, with the current (active) implementation highlighted.



The new implementation uses the same source code files, but different device options and constraints. It copies some files from the previous implementation: the `t1g` log file, and the `srs` RTL netlist file. The software keeps a repeatable history of the synthesis runs.

2. Run synthesis again with the new settings.
  - To run the current implementation only, click Run.
  - To run all the implementations in a project, select Run->Run All Implementations.

You can use multiple implementations to try a different part or experiment with a different frequency. See [Setting Logic Synthesis Implementation Options, on page 74](#) for information about setting options.

The Project view shows all implementations with the active implementation highlighted and the corresponding output files generated for the active implementation displayed in the Implementation Results view on the right; changing the active implementation changes the output file display. The Watch window monitors the active implementation. If you configure this window to watch all implementations, the new implementation is automatically updated in the window.

3. Compare the results.
  - Use the Watch window to compare selected criteria. Make sure to set the implementations you want to compare with the Configure Watch command. See [Using the Watch Window, on page 177](#) for details.

Log Parameter	rev_1	rev_2
eight_bit_uc clock - Estimated Frequency	47.0 MHz	201.6 MHz
eight_bit_uc clock - Requested Frequency	55.3 MHz	237.1 MHz
eight_bit_uc clock - Slack	-3.191	-0.744

- To compare details, compare the log file results.
4. To rename an implementation, click the right mouse button on the implementation name in the project view, select Change Implementation Name from the popup menu, and type a new name.
- Note that the current UI overwrites the implementation; releases prior to 9.0 preserve the implementation to be renamed.
5. To copy an implementation, click the right mouse button on the implementation name in the project view, select Copy Implementation from the popup menu, and type a new name for the copy.
  6. To delete an implementation, click the right mouse button on the implementation name in the project view, and select Remove Implementation from the popup menu.

# Setting Logic Synthesis Implementation Options

You can set global options for your synthesis implementations, some of them technology-specific. This section describes how to set global options like device, optimization, and file options with the Implementation Options command. For information about setting constraints for the implementation, see [Specifying SCOPE Constraints, on page 121](#). For information about overriding global settings with individual attributes or directives, see [Specifying Attributes and Directives, on page 89](#).

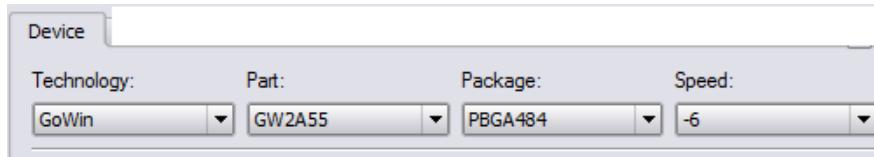
This section discusses the following topics:

- [Setting Device Options](#), on page 74
- [Setting Optimization Options](#), on page 76
- [Specifying Global Frequency and Constraint Files](#), on page 78
- [Specifying Result Options](#), on page 80
- [Specifying Timing Report Output](#), on page 82
- [Setting Verilog and VHDL Options](#), on page 83

## Setting Device Options

Device options are part of the global options you can set for the synthesis run. They include the part selection (technology, part and speed grade) and implementation options (I/O insertion and fanouts). The options and the implementation of these options can vary from technology to technology, so check the vendor chapters of the *Reference Manual* for information about your vendor options.

1. Open the Implementation Options form by clicking the Implementation Options button or selecting Project->Implementation Options, and click the Device tab at the top if it is not already selected.
2. Select the technology, part, package, and speed.



3. Set the device mapping options. The options vary, depending on the technology you choose.
  - If you are unsure of what an option means, click the option to see a description in the box below. For full descriptions of the options, click F1 or refer to the appropriate vendor chapter in the *Reference Manual*.
  - To set an option, type in the value or check the box to enable it.

For more information about setting fanout limits, pipelining, and retiming, see [Setting Fanout Limits, on page 402](#), [Pipelining, on page 384](#), and [Retiming, on page 388](#), respectively. For details about other vendor-specific options, refer to the appropriate vendor chapter and technology family in the *Reference Manual*.

Device Mapping Options	
Option	Value
Fanout Guide	10000
Disable I/O Insertion	<input type="checkbox"/>
Update Compile Point Timing Data	<input type="checkbox"/>
Read Write Check on RAM	<input checked="" type="checkbox"/>
Annotated Properties for Analyst	<input checked="" type="checkbox"/>
Resolve Mixed Drivers	<input type="checkbox"/>

4. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 74](#) for a list of choices). Click OK.

5. Click the Run button to synthesize the design. The software compiles and maps the design using the options you set.
6. To set device options with a script, use the `set_option` Tcl command. The following table contains an alphabetical list of the device options on the Device tab mapped to the equivalent Tcl commands. Because the options are technology- and family-based, all of the options listed in the table may not be available in the selected technology. All commands begin with `set_option`, followed by the syntax in the column as shown. Check the *Reference Manual* for the most comprehensive list of options for your vendor.

The following table shows a majority of the device options.

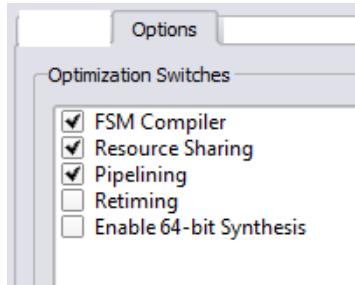
Option	Tcl Command ( <code>set_option...</code> )
Annotated Properties for Analyst	<code>-run_prop_extract {1 0}</code>
Disable I/O Insertion	<code>-disable_io_insertion {1 0}</code>
Disable Sequential Optimizations	<code>-no_sequential_opt {1 0}</code>
Fanout Guide	<code>-fanout_limit fanout_value</code>
Package	<code>-package pkg_name</code>
Part	<code>-part part_name</code>
Resolve Mixed Drivers	<code>-resolve_multiple_driver {1 0}</code>
Speed	<code>-speed_grade speed_grade</code>
Technology	<code>-technology keyword</code>
Update Compile Point Timing Data	<code>-update_models_cp {0 1}</code>

## Setting Optimization Options

Optimization options are part of the global options you can set for the implementation. This section tells you how to set options like frequency and global optimization options like resource sharing. You can also set some of these options with the appropriate buttons on the UI.

1. Open the Implementation Options form by clicking the Implementation Options button or selecting Project->Implementation Options, and click the Options tab at the top.

2. Click the optimization options you want, either on the form or in the Project view. Your choices vary, depending on the technology. If an option is not available for your technology, it is greyed out. Setting the option in one place automatically updates it in the other.



For details about using these optimizations refer to the following sections:

FSM Compiler	<a href="#">Optimizing State Machines, on page 407</a>
Resource Sharing	<a href="#">Sharing Resources, on page 406</a>
Pipelining	<a href="#">Pipelining, on page 384</a>
Retiming	<a href="#">Retiming, on page 388</a>

The equivalent Tcl `set_option` command options are as follows:

Option	set_option Tcl Command Option
FSM Compiler	<code>-symbolic_fsm_compiler {1 0}</code>
Resource Sharing	<code>-resource_sharing {1 0}</code>
Pipelining	<code>-pipe {0 1}</code>
Retiming	<code>-retiming {1 0}</code>

3. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 74](#) for a list of choices). Click OK.

4. Click the Run button to run synthesis.

The software compiles and maps the design using the options you set.

## Specifying Global Frequency and Constraint Files

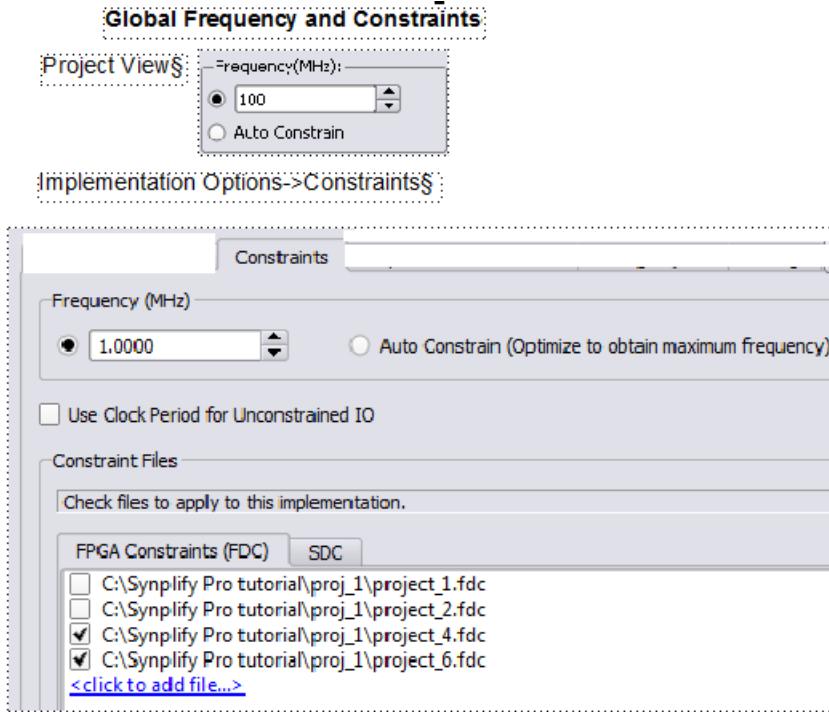
This procedure tells you how to set the global frequency and specify the constraint files for the implementation.

1. To set a global frequency, do one of the following:

- Type a global frequency in the Project view.
- Open the Implementation Options form by clicking the Implementation Options button or selecting Project->Implementation Options, and click the Constraints tab.

The equivalent Tcl `set_option` command is `-frequency frequencyValue`.

You can override the global frequency with local constraints, as described in [Specifying SCOPE Constraints, on page 121](#). In the Synplify Pro tool, you can automatically generate clock constraints for your design instead of setting a global frequency.



2. To specify constraint files for an implementation, do one of the following:
  - Select Project->Implementation Options->Constraints. Check the constraint files you want to use in the project.
  - From the Implementation Options->Constraints panel, you can also click to add a constraint file.
  - With the implementation you want to use selected, click Add File in the Project view, and add the constraint files you need.

To create constraint files, see [Specifying SCOPE Constraints, on page 121](#).

3. To remove constraint files from an implementation, do one of the following:
  - Select Project->Implementation Options->Constraints. Click off the checkbox next to the file name.
  - In the Project view, right-click the constraint file to be removed and select Remove from Project.

This removes the constraint file from the implementation, but does not delete it.

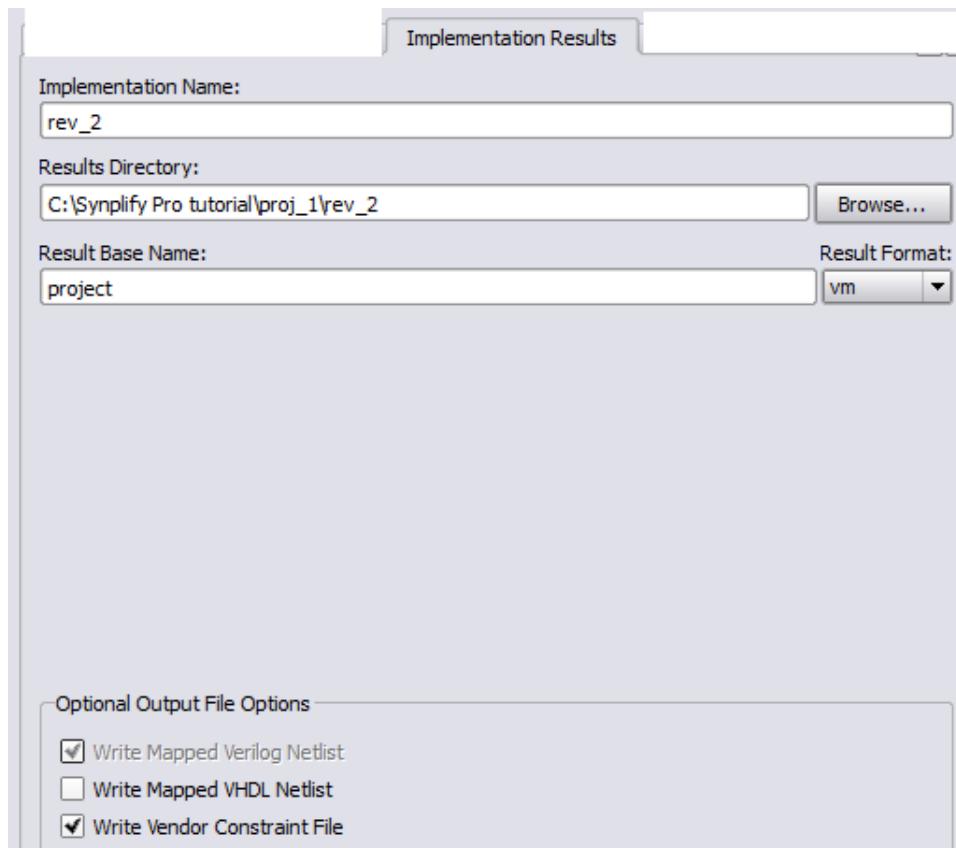
4. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 74](#) for a list of choices). Click OK.

When you synthesize the design, the software compiles and maps the design using the options you set.

## Specifying Result Options

This section shows you how to specify criteria for the output of the synthesis run.

1. Open the Implementation Options form by clicking the Implementation Options button or selecting Project->Implementation Options, and click the Implementation Results tab at the top.



2. Specify the output files you want to generate.
  - To generate mapped netlist files, click Write Mapped Verilog Netlist or Write Mapped VHDL Netlist.
  - To generate a vendor-specific constraint file for forward annotation, click Write Vendor Constraint File.
3. Set the directory to which you want to write the results.

4. Set the format for the output file. The equivalent Tcl command for scripting is `project -result_format format`.

You might also want to set attributes to control name-mapping. For details, refer to the appropriate vendor chapter in the *Reference Manual*.

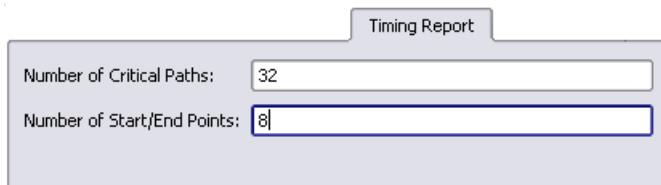
5. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 74](#) for a list of choices). Click OK.

When you synthesize the design, the software compiles and maps the design using the options you set.

## Specifying Timing Report Output

You can determine how much is reported in the timing report by setting the following options.

1. Selecting Project->Implementation Options, and click the Timing Report tab.
2. Set the number of critical paths you want the software to report.



3. Specify the number of start and end points you want to see reported in the critical path sections.
4. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 74](#) for a list of choices). Click OK.

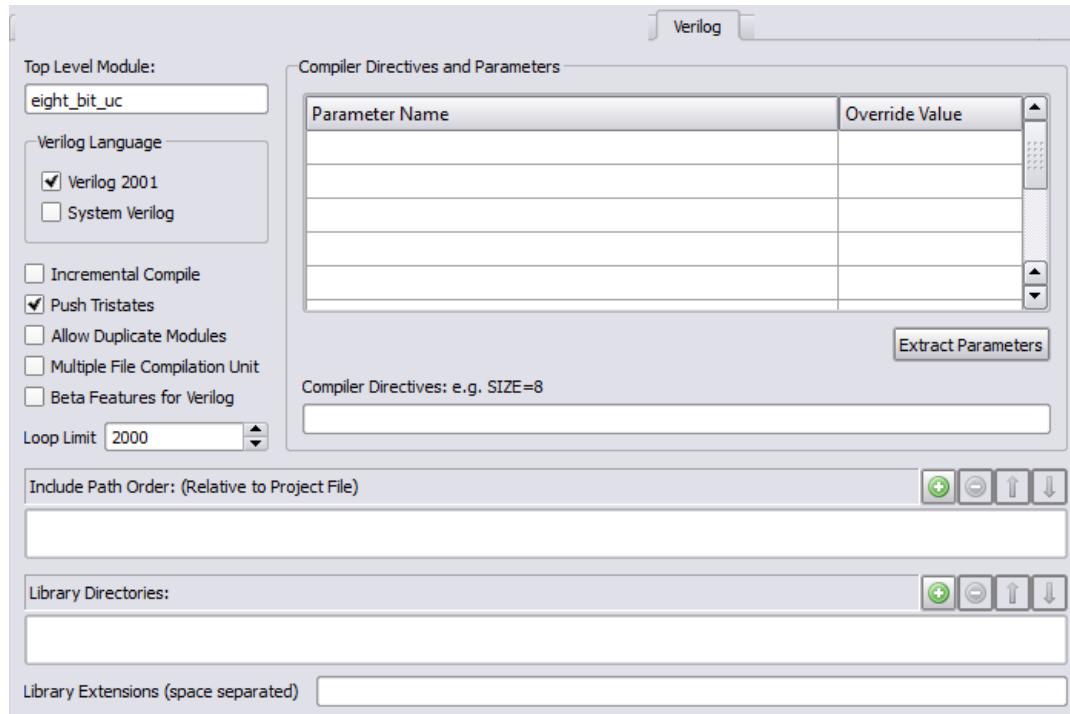
When you synthesize the design, the software compiles and maps the design using the options you set.

## Setting Verilog and VHDL Options

When you set up the Verilog and VHDL source files in your project, you can also specify certain compiler options.

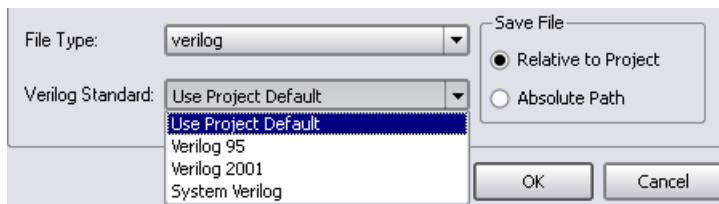
### Setting Verilog File Options

You set Verilog file options by selecting either Project->Implementation Options->Verilog, or Options->Configure Verilog Compiler.



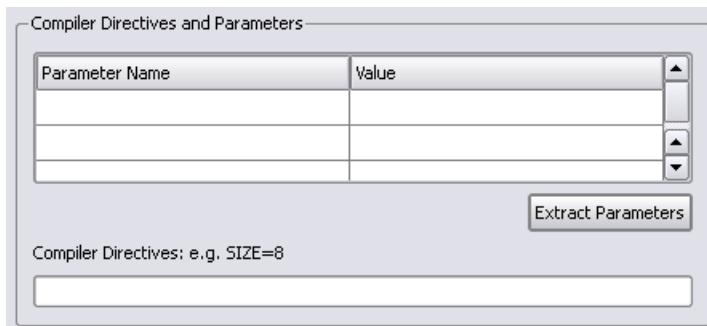
1. Specify the Verilog format to use.

- To set the compiler globally for all the files in the project, select Project->Implementation Options->Verilog. If you are using Verilog 2001 or SystemVerilog, check the *Command Reference Manual* for supported constructs.
- To specify the Verilog compiler on a per file basis, select the file in the Project view. Right-click and select File Options. Select the appropriate compiler. The default Verilog file format for new projects is SystemVerilog.



2. Specify the top-level module if you did not already do this in the Project view.
3. To extract parameters from the source code, do the following:
- Click Extract Parameters.
  - To override the default, enter a new value for a parameter.

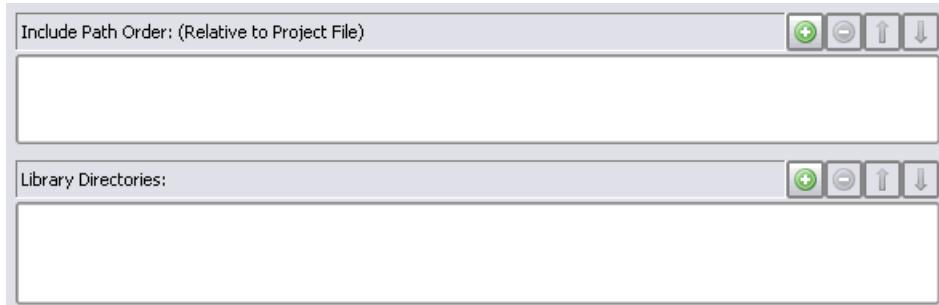
The software uses the new value for the current implementation only. Note that parameter extraction is not supported for mixed designs.



4. Type in the directive in Compiler Directives, using spaces to separate the statements.

You can type in directives you would normally enter with 'ifdef and 'define statements in the code. For example, ABC=30 results in the software writing the following statements to the project file:

```
set_option -hdl_define -set "ABC=30"
```

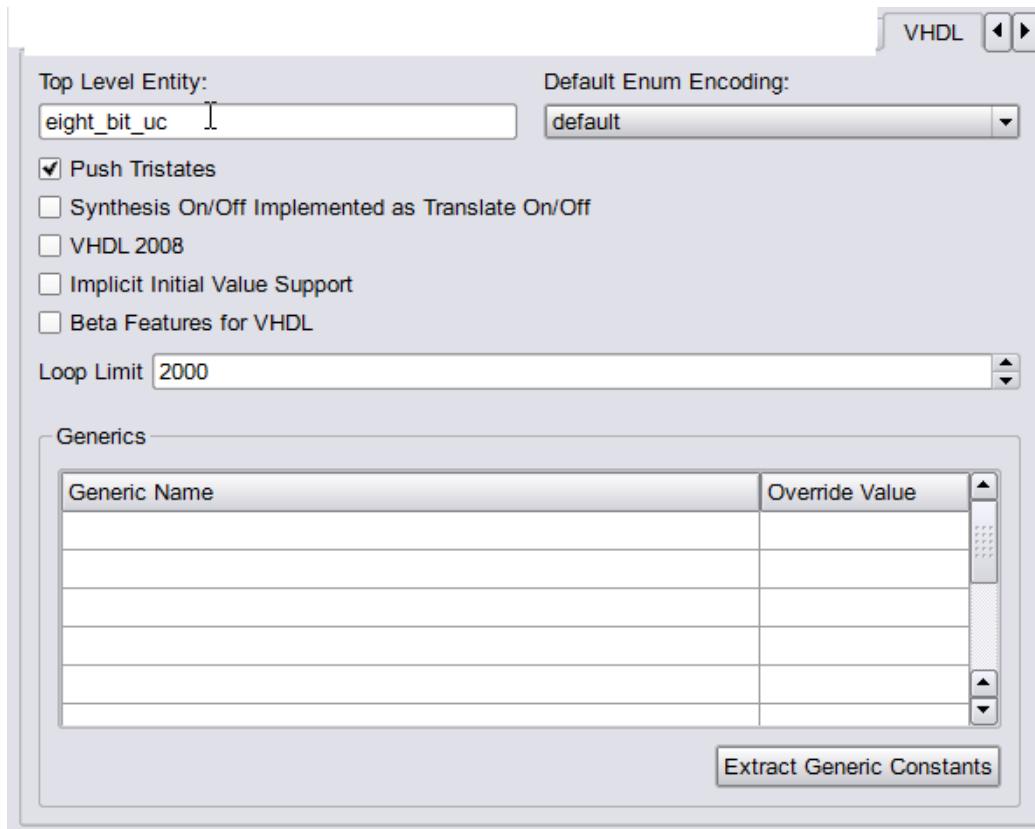


5. In the Include Path Order, specify the search paths for the include commands for the Verilog files that are in your project. Use the buttons in the upper right corner of the box to add, delete, or reorder the paths.
6. In the Library Directories or Files, specify the path to the directory which contains the library files for your project. Use the buttons in the upper right corner of the box to add, delete, or reorder the paths or files.
7. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 74](#) for a list of choices). Click OK.

When you synthesize the design, the software compiles and maps the design using the options you set.

## Setting VHDL File Options

You set VHDL file options by selecting either Project->Implementation Options->VHDL, or Options->Configure VHDL Compiler.



For VHDL source, you can specify the options described below.

1. Specify the top-level module if you did not already do this in the Project view. If the top-level module is not located in the default work library, you must specify the library where the compiler can find the module. For information on how to do this, see [VHDL Panel](#), on page 323.

You can also use this option for mixed language designs or when you want to specify a module that is not the actual top-level entity for HDL Analyst displaying and debugging in the schematic views.

2. For user-defined state machine encoding, do the following:

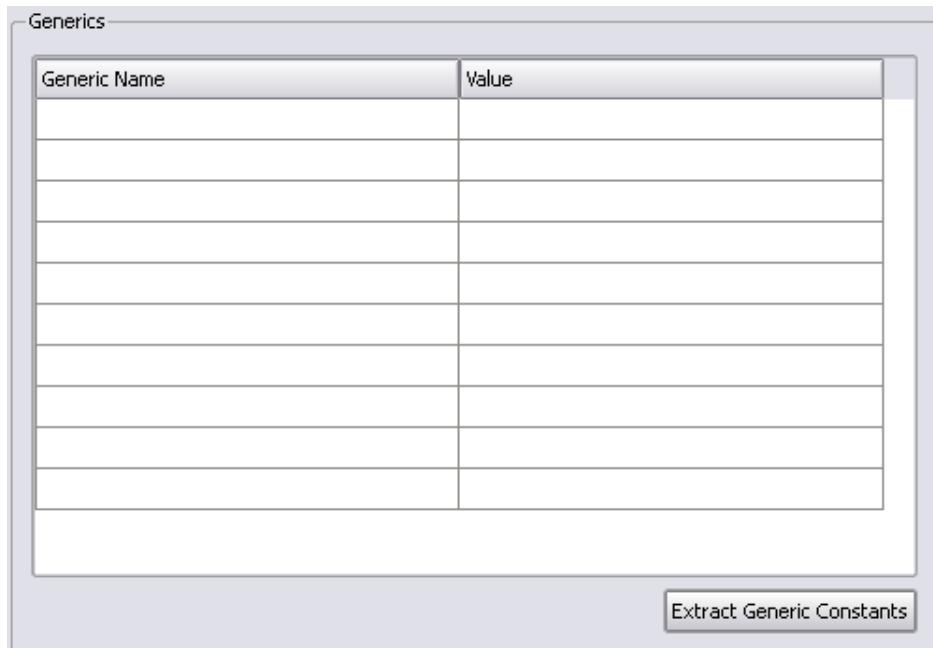
- Specify the kind of encoding you want to use.
- Disable the FSM compiler.

When you synthesize the design, the software uses the compiler directives you set here to encode the state machines and does not run the FSM compiler, which would override the compiler directives. Alternatively, you can define state machines with the `syn_encoding` attribute, as described in [Defining State Machines in VHDL, on page 371](#).

3. To extract generics from the source code, do this:

- Click Extract Generic Constants.
- To override the default, enter a new value for a generic.

The software uses the new value for the current implementation only. Note that you cannot extract generics if you have a mixed language design.



4. To push tristates across process/block boundaries, check that Push Tristates is enabled. For details, see [Push Tristates Option](#), on page 329 in the *Command Reference Manual*.
5. Determine the interpretation of the synthesis\_on and synthesis\_off directives:
  - To make the compiler interpret synthesis\_on and synthesis\_off directives like translate\_on/translate\_off, enable the Synthesis On/Off Implemented as Translate On/Off option.
  - To ignore the synthesis\_on and synthesis\_off directives, make sure that this option is not checked. See [translate\\_off/translate\\_on](#), on page 146 in the *Attribute Reference Manual* for more information.
6. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options](#), on page 74 for a list of choices). Click OK.

When you synthesize the design, the software compiles and maps the design using the options you set.

# Specifying Attributes and Directives

Attributes and directives are specifications that you assign to design objects to control the way your design is analyzed, optimized, and mapped.

Attributes control mapping optimizations and directives control compiler optimizations. Because of this difference, you must specify directives in the source code or the compiler directives file. This table describes the methods that are available to create attribute and directives specifications:

	<b>Attributes</b>	<b>Directives</b>
<b>VHDL</b>	Yes	Yes
<b>Verilog</b>	Yes	Yes
<b>SCOPE Editor</b>	Yes	No
<b>Constraints File</b>	Yes	No

It is better to specify attributes in the SCOPE editor or the constraints file, because you do not have to recompile the design first. For directives, you must compile the design for them to take effect.

If SCOPE/constraints file and the HDL source code are both specified for a design, the fdc constraints has priority.

For further details, refer to the following:

- [Specifying Attributes and Directives in VHDL](#), on page 90
- [Specifying Attributes and Directives in Verilog](#), on page 91
- [Specifying Attributes Using the SCOPE Editor](#), on page 92
- [Specifying Attributes in the Constraints File](#), on page 95
- [Handling Properties with Attributes or Directives](#), on page 96

## Specifying Attributes and Directives in VHDL

You can use other methods to add attributes to objects, as listed in [Specifying Attributes and Directives, on page 89](#). However, you can specify directives only in the source code. There are two ways of defining attributes and directives in VHDL:

- Using the predefined attributes package
- Declaring the attribute each time it is used

For details of VHDL attribute syntax, see [VHDL Attribute and Directive Syntax, on page 399](#) in the *Reference Manual*.

### Using the Predefined VHDL Attributes Package

The advantage to using the predefined package is that you avoid redefining the attributes and directives each time you include them in source code. The disadvantage is that your source code is less portable. The attributes package is located in *installDirectory/lib/vhd/synattr.vhd*.

1. To use the predefined attributes package included in the software library, add these lines to the syntax:

```
library synplify;
use synplify.attributes.all;
```

2. Add the attribute or directive you want after the design unit declaration.

```
declarations;
attribute attribute_name of objectName : objectType is value;
```

For details of the syntax conventions, see [VHDL Attribute and Directive Syntax, on page 399](#) in the *Reference Manual*.

3. Add the source file to the project.

## Declaring VHDL Attributes and Directives

If you do not use the attributes package, you must redefine the attributes each time you include them in source code.

1. Every time you use an attribute or directive, define it immediately after the design unit declarations using the following syntax:

```
design_unit_declaration;
attribute attributeName : dataType ;
attribute attributeName of objectName :
objectType is value ;
```

2. Add the source file to the project.

## Specifying Attributes and Directives in Verilog

You can use other methods to add attributes to objects, as described in [Specifying Attributes and Directives, on page 89](#). However, you can specify directives only in the source code.

Verilog does not have predefined synthesis attributes and directives, so you must add them as comments. The attribute or directive name is preceded by the keyword **synthesis**. Verilog files are case sensitive, so attributes and directives must be specified exactly as presented in their syntax descriptions. For syntax details, see [Verilog Attribute and Directive Syntax, on page 123](#) in the *Reference Manual*.

1. To add an attribute or directive in Verilog, use Verilog line or block comment (C-style) syntax directly following the design object. Block comments must precede the semicolon, if there is one.

### Verilog Block Comment Syntax

```
/* synthesis attributeName = value */
/* synthesis directoryName = value */
```

### Verilog Line Comment Syntax

```
// synthesis attributeName = value
// synthesis directoryName = value
```

For details of the syntax rules, see [Verilog Attribute and Directive Syntax, on page 123](#) in the *Reference Manual*. The following are examples:

```
module fifo(out, in) /* synthesis syn_hier = "hard" */;
```

```
module b_box(out, in); // synthesis syn_black_box
```

2. To attach multiple attributes or directives to the same object, separate the attributes with white spaces, but do not repeat the synthesis keyword. Do not use commas. For example:

```
case state /* synthesis full_case parallel_case */;
```

3. If multiple registers are defined using a single Verilog `reg` statement and an attribute is applied to them, then the synthesis software only applies the last declared register in the `reg` statement. For example:

```
reg [5:0] q, q_a, q_b, q_c, q_d /* synthesis syn_preserve=1 */;
```

The `syn_preserve` attribute is only applied to `q_d`. This is the expected behavior for the synthesis tools. To apply this attribute to all registers, you must use a separate Verilog `reg` statement for each register and apply the attribute.

## Specifying Attributes Using the SCOPE Editor

The SCOPE window provides an easy-to-use interface to add any attribute. You cannot use it for adding directives, because they must be added to the source files. (See [Specifying Attributes and Directives in VHDL, on page 90](#) or [Specifying Attributes and Directives in Verilog, on page 91](#)). The following procedure shows how to add an attribute directly in the SCOPE window.

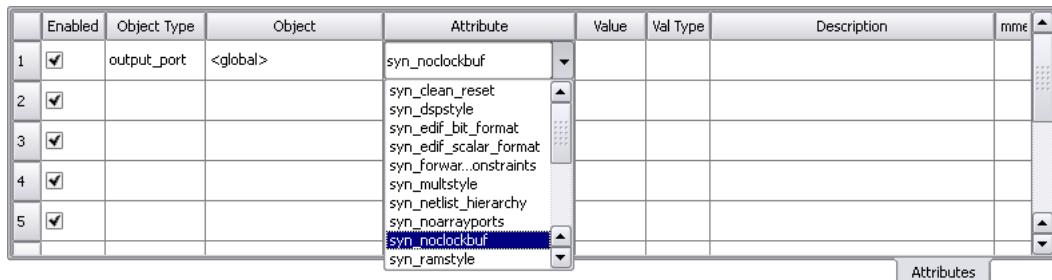
1. Start with a compiled design and open the SCOPE window. To add the attributes to an existing constraint file, open the SCOPE window by clicking on the existing file in the Project view. To add the attributes to a new file, click the SCOPE icon and click Initialize to open the SCOPE window.
2. Click the Attributes tab at the bottom of the SCOPE window.

You can either select the object first (step 3) or the attribute first (step 4).

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_allow_retiming		boolean	Controls retiming of registers
2							
3							
4							
5							
6							
7							
8							
9							

3. To specify the object, do one of the following in the Object column. If you already specified the attribute, the Object column lists only valid object choices for that attribute.
- Select the type of object in the Object Filter column, and then select an object from the list of choices in the Object column. This is the best way to ensure that you are specifying an object that is appropriate, with the correct syntax.
  - Drag the object to which you want to attach the attribute from the RTL or Technology views to the Object column in the SCOPE window. For some attributes, dragging and dropping may not select the right object. For example, if you want to set syn\_hier on a module or entity like an and gate, you must set it on the view for that module. The object would have this syntax: v:*moduleName* in Verilog, or v:*library.moduleName* in VHDL, where you can have multiple libraries.
  - Type the name of the object in the Object column. If you do not know the name, use the Find command or the Object Filter column. Make sure to type the appropriate prefix for the object where it is needed. For example, to set an attribute on a view, you must add the v: prefix to the module or entity name. For VHDL, you might have to specify the library as well as the module name.

- If you specified the object first, you can now specify the attribute. The list shows only the valid attributes for the type of object you selected. Specify the attribute by holding down the mouse button in the Attribute column and selecting an attribute from the list.



The screenshot shows a table titled "Attribute Specification" with columns: Enabled, Object Type, Object, Attribute, Value, Val Type, Description, and mme. Row 1: Enabled checked, Object Type "output\_port", Object "<global>", Attribute "syn\_noclockbuf". Row 2: Enabled checked, Object Type "", Object "", Attribute "syn\_clean\_reset". Row 3: Enabled checked, Object Type "", Object "", Attribute "syn\_dspstyle". Row 4: Enabled checked, Object Type "", Object "", Attribute "syn\_edif\_bit\_format". Row 5: Enabled checked, Object Type "", Object "", Attribute "syn\_edif\_scalar\_format". Row 6: Enabled checked, Object Type "", Object "", Attribute "syn\_forwar...onstraints". Row 7: Enabled checked, Object Type "", Object "", Attribute "syn\_multstyle". Row 8: Enabled checked, Object Type "", Object "", Attribute "syn\_netlist\_hierarchy". Row 9: Enabled checked, Object Type "", Object "", Attribute "syn\_noarrayports". Row 10: Enabled checked, Object Type "", Object "", Attribute "syn\_noclockbuf". Row 11: Enabled checked, Object Type "", Object "", Attribute "syn\_ramstyle". A scroll bar is visible on the right side of the table.

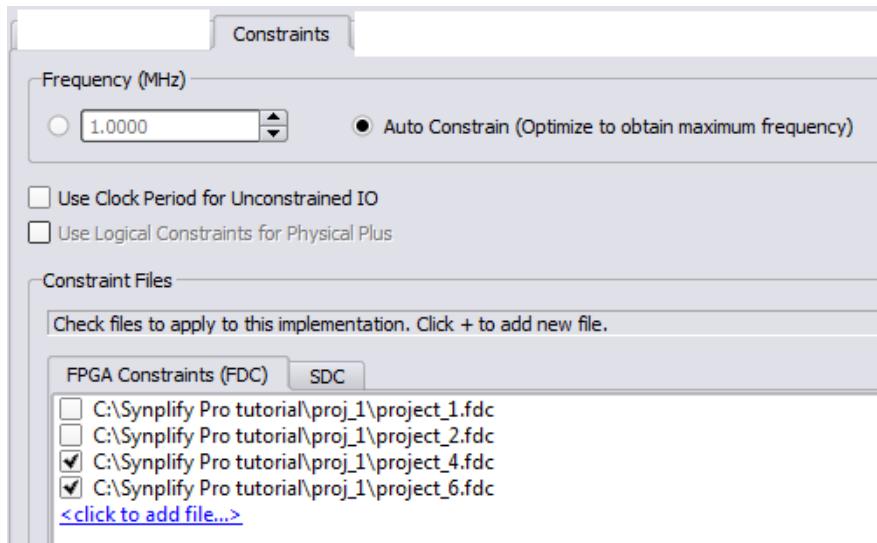
If you selected the object first, the choices available are determined by the selected object and the technology you are using. If you selected the attribute first, the available choices are determined by the technology.

When you select an attribute, the SCOPE window tells you the kind of value you must enter for that attribute and provides a brief description of the attribute. If you selected the attribute first, make sure to go back and specify the object.

- Fill out the value. Hold down the mouse button in the Value column, and select from the list. You can also type in a value.
- Save the file.

The software creates a Tcl constraint file composed of define\_attribute statements for the attributes you specified. See [How Attributes and Directives are Specified](#), on page 6 of the *Attribute Reference Manual* for the syntax description.

- Add it to the project, if it is not already in the project.
  - Choose Project -> Implementation Options.
  - Go to the Constraints panel and check that the file is selected. If you have more than one constraint file, select all those that apply to the implementation.



The software saves the SCOPE information in a Tcl constraint file, using `define_attribute` statements. When you synthesize the design, the software reads the constraint file and applies the attributes.

## Specifying Attributes in the Constraints File

When you use the SCOPE window ([Specifying Attributes Using the SCOPE Editor, on page 92](#)), the attributes are automatically written to a constraint file using the Tcl `define_attribute` syntax. This is the preferred method for defining constraints as the syntax is determined for you.

However, the following procedure explains how you can specify attributes directly in the constraint file.

1. Open a file in a text editor.
2. Enter the desired attributes. For example,

```
define_attribute {objectName} attributeName value
```

For commands and syntax, see [Summary of Attributes and Directives, on page 9](#) in the *Reference Manual*.

3. Save the constraints in a file using the FDC file extension.

## Handling Properties with Attributes or Directives

Any property added to an object (i.e. net or instance) that is preserved or kept during the flow will be annotated in the netlist. Only properties for the syn\_\* attributes/directives are processed by the tool; while all other properties are simply annotated in the netlist when the object is available in the flow.

### Examples

Suppose top\_property\_handling.v contains MyProp=Value, which is associated with out and is annotated in the netlist. If you apply the property on the net intermediate\_net, then the property is not annotated in the netlist since this net is not preserved/kept in the flow.

#### Using the syn\_keep Attribute

The syn\_keep attribute helps preserve the specified net. If top\_syn\_keep.v applies syn\_keep on the net intermediate\_net, then this same net is kept.

#### Using Properties with the syn\_keep Attribute

For this example, top.v applies the syn\_keep to the net intermediate\_net where the MyProp=Value is associated with intermediate\_net. In this case, the property is annotated in the netlist since syn\_keep instructs the tool to preserve/keep the net in the flow.

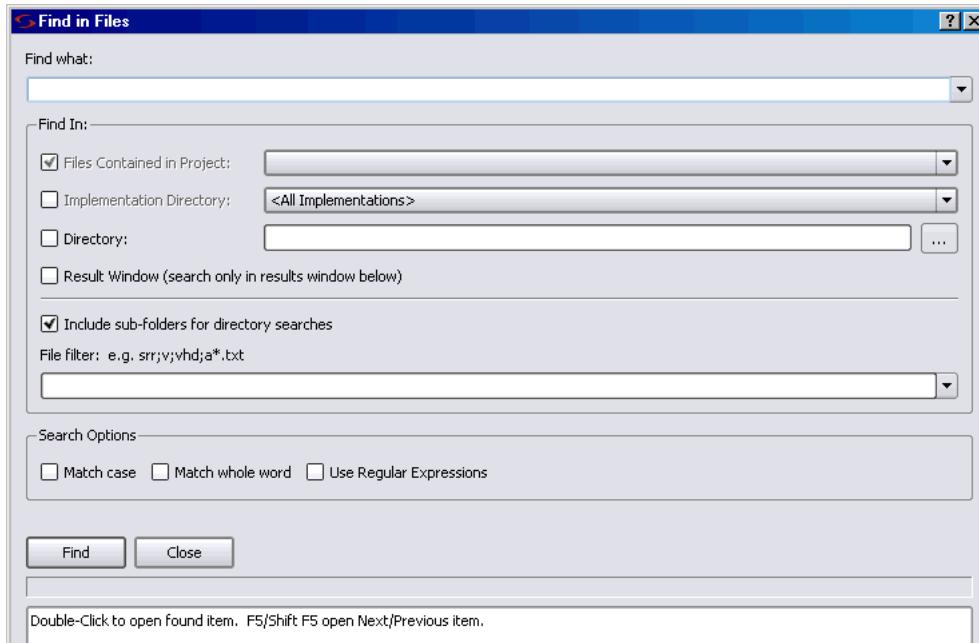
# Searching Files

A find-in-files feature is available to perform string searches within a specified set of files. Advantages to using this feature include:

- Ability to restrict the set of files to be searched to a project or implementation.
- Ability to cross probe the search results.

The find-in-files feature uses a dialog box to specify the search pattern, the criteria for selecting the files to be searched, and any search options such as match case or whole word. The files that meet the criteria are searched for the pattern, and a list of the files containing the search pattern are displayed at the bottom of the dialog box.

To use the find-in-files feature, open the Find in Files dialog box by selecting Edit->Find in Files and enter the search pattern in the Find what field at the top of the dialog box.



## Identifying the Files to Search

The Find In section at the top of the dialog box identifies the files to be searched:

- Project Files – Searches the files included in the selected project (use the drop-down menu to select the project). By default, the files in the active project are searched. The files can reside anywhere on the disk; any project ‘include’ files are also searched.
- Implementation Directory – Searches all files in the specified implementation directory (use the drop-down menu to select the implementation). By default, the files in the active implementation are searched. You can search all implementations by selecting <All Implementations> from the drop-down menu. If Include sub-folders for directory searches is also selected, all files in the implementation directory hierarchy are searched.
- Directory – Searches all files in the specified directory (use the browser button to select the directory). If Include sub-folders for directory searches is also selected, all files in the directory hierarchy are searched.

All of the above selection methods can be applied concurrently when searching for a specified pattern.

The Result Window selection is used after any of the above selection methods to search the resulting list of files for a subsequent sub-pattern.

## Filtering the Files to Search

A file filter allows the file set to be searched to be further restricted based on the matching of patterns entered into the File filter field.

- A pattern without a wildcard or a “.” (period) is interpreted as a filename extension. For example, fdc restricts the search to only constraint files.
- Multiple patterns can be specified using a semicolon delimiter. For example, v;vhd restricts the files searched to only Verilog and VHDL files.
- Wildcard characters can be used in the pattern to match file names. For example, a\*.vhd restricts the files searched to VHDL files that begin with an “a” character.

- Leaving the File filter field empty searches all files that meet the Find In criteria.
- The Match Case, Whole Word, and Regular Expressions search options can be used to further restrict searches.

## Initiating the Search

After entering the search criteria, click the Find button to initiate the search. All matches found are listed in the results area at the bottom of the dialog box; the status line just below the Find button reports the number of matches found in the indicated number of files and the total number of files searched.

While the find operation is running, the status line is continually updated with how many matches are found in how many files and how many files are being searched.

## Search Results

The search results are displayed in the results window at the bottom of the dialog box. For each match found, the entire line of the file is displayed in the following format:

*fullpath\_to\_file(lineNumber): matching\_line\_text*

For example, the entry

```
C:\Designs\leon\dcache.vhd(487): wdata := r.wb.data1;
```

indicates that the search pattern (data1) was found on line 487 of the dcache.vhd file.

To open the target file at the specified line, double-click the line in the results window.

# Archiving Files and Projects

Use the archive utility to archive, extract (unarchive), or copy design projects. Archived files are in a proprietary format and saved to a file name using the sar extension. The archive utility is available through the Project menu in the GUI or using the project command in the Tcl window.

This document provides a description of how to use the utility.

- [Archive a Project](#)
- [Un-Archive a Project](#)
- [Copy a Project](#)
- [Support for Hierarchical Include Paths](#)

## Archive a Project

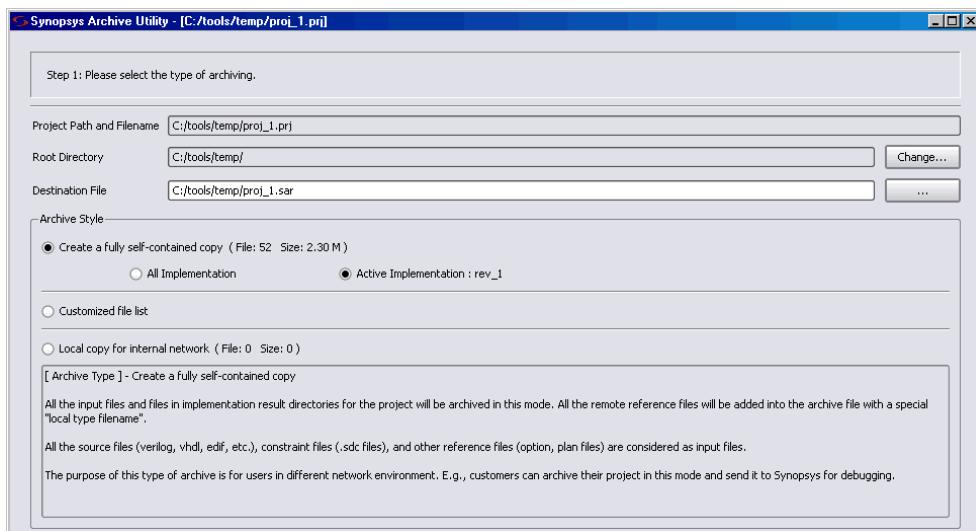
Use the archive utility to store the files for a design project into a single archive file in a proprietary format (sar). You can archive an entire project or selected files from a project. If you want to create a copy of a project without archiving the files, see [Copy a Project, on page 106](#).

Here are the steps to create an archive:

1. In the Project view, select Project->Archive Project to bring up the wizard.

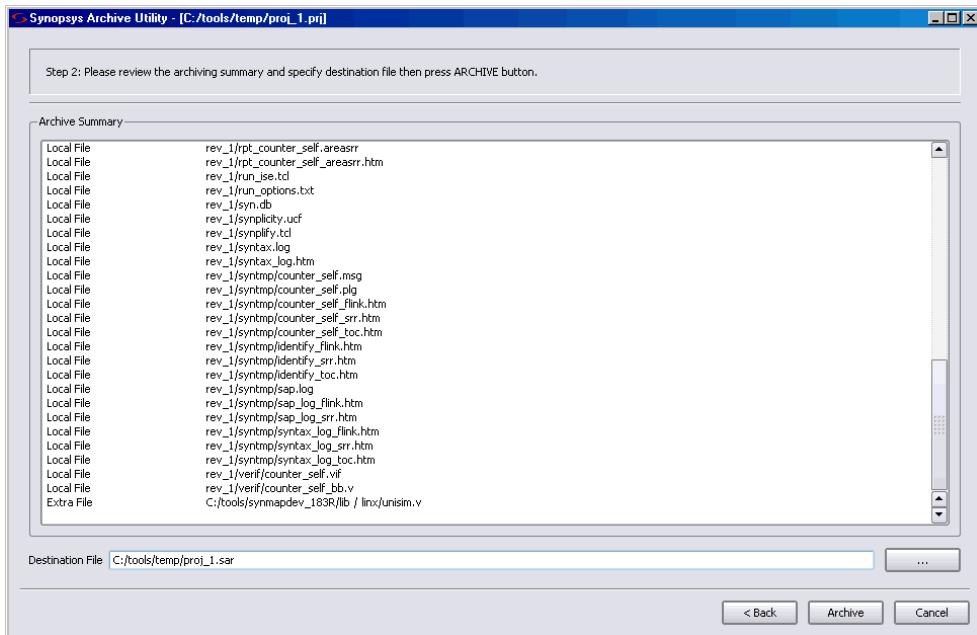
The Tcl command equivalent is project [-archive](#). For a complete description of the project Tcl command options for archiving, see [project, on page 66](#) of the *Reference Manual*.

The archive utility automatically runs a syntax check on the active project (Run->Syntax Check command) to ensure that a complete list of project files is generated. If you have Verilog 'include files in your project, the utility includes the complete list of Verilog files. It also checks the syntax automatically for each implementation in the project to ensure that the file list is complete for each implementation as well. The wizard displays the name of the project to archive, the top-level directory where the project file is located (root directory), and other information.

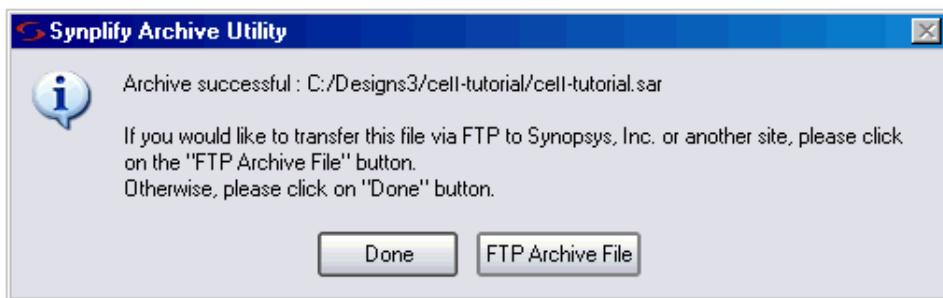


2. Do the following on the first page of the wizard:
  - Fill in Destination File with a location for the archive file.
  - Set Archive Style. You can archive all the project files with all the implementations or selectively archive files and implementations
  - To archive only the active implementation, enable Active Implementation.
  - To selectively archive files, enable Customized file list, click Next, and use the check boxes to include files in or exclude files from the archive. Use the Add Extra Files button on this page to include additional files in the project.
3. Click Next.

If you did not select Customized file list, the tool summary displays all the files in the archive and shows the full uncompressed file size as shown in step 5 (the actual size is smaller after the archiving operation as there is no duplication of files). When you select Customized file list, the following interim menu is displayed to allow you to exclude specific file from the archive.



4. Click next to advance to the next screen (step 3).
5. Verify that the current archive contains the files that you want, then click Archive which creates the project archive sar file. If the list of files is incorrect, click Back and include/exclude any desired files.



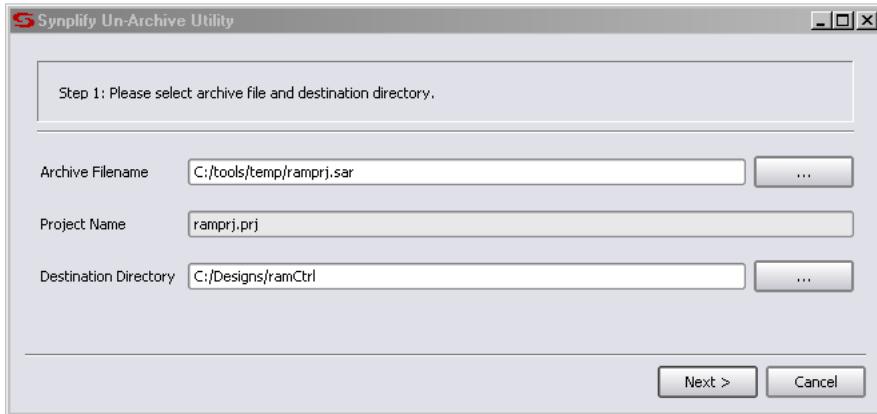
6. Click Archive if you are finished. The synthesis tool reports the archive success and the path location of the archive file.

## Un-Archive a Project

Uses this procedure to extract design project files from an archive file (sar).

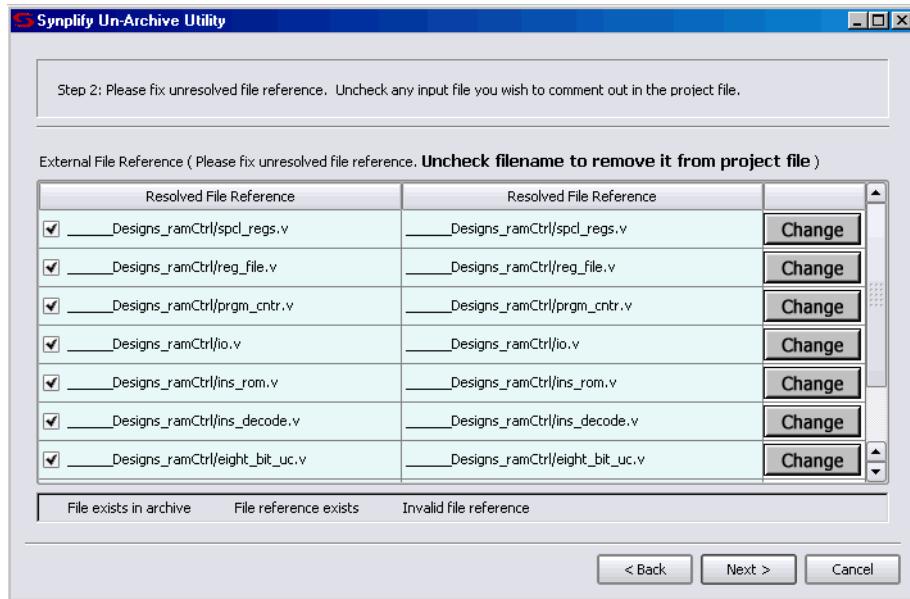
1. In the Project view, select Project->Un-Archive Project to display the wizard

The Tcl command equivalent is project [-unarchive](#). For a complete description of the project Tcl command options for archiving, see [project, on page 66](#) of the *Reference Manual*.

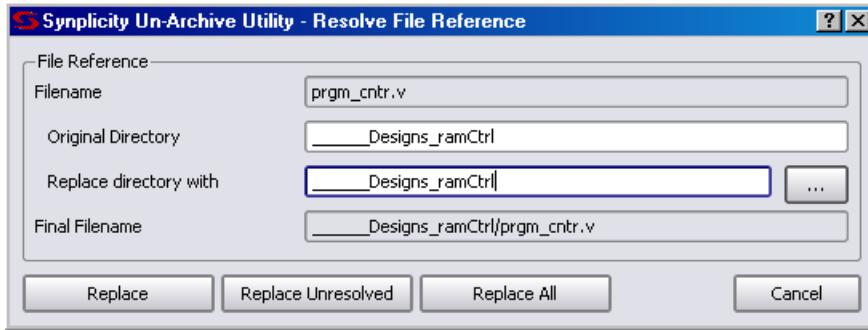


2. In the wizard, enter the following:

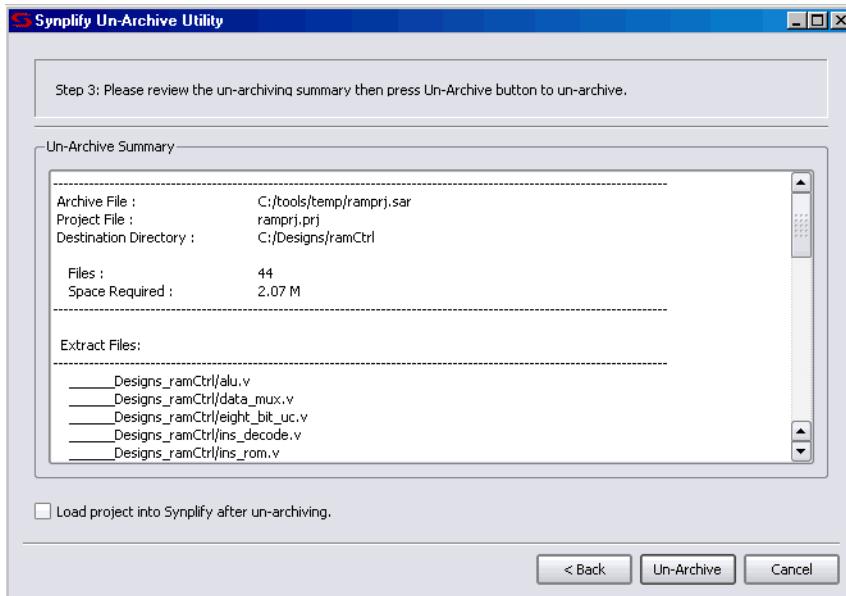
- Name of the sar file containing the project files.
- Name of project to extract (un-archive). This field is automatically extracted from the sar file and cannot be changed.
- Pathname of directory in which to write the project files destination.
- Click Next.



3. Make sure all the files that you want to extract are checked and references to these files are resolved.
  - If there are files in the list that you do not want to include when the project is un-archived, uncheck the box next to the file. The un-checked files will be commented out in the project file (`prj`) when project files are extracted.
  - If you need to resolve a file in the project before un-archiving, click the **Resolve** button and fill out the dialog box.
  - If you want to replace a file in the project, click the **Change** button and fill out the dialog box. Put the replacement files in the directory you specify in Replace directory. You can replace a single file, any unresolved files, or all the files. You can also undo the replace operation.



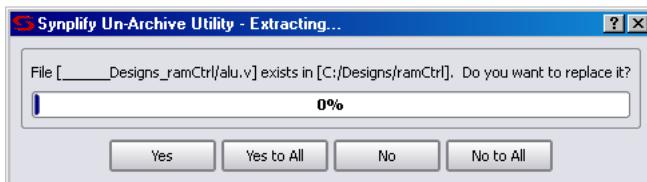
- Click Next and verify that the project files you want are displayed in the Un-Archive Summary.



- If you want to load this project in the UI after files have been extracted, enable the Load project into Synplify Pro after un-archiving option.
- When the Add extra input path to project file option is enabled, the archive utility finds all include files and copies them into a directory called extra\_input. This directory is added to the unarchived project file.

The Tcl command equivalent is `set_option -include_path "./extra_input"`.

7. If the archive files contain relative or absolute include paths, the `_SEARCHFILENAMEONLY_` directive can have the compiler remove the relative/absolute paths from the 'include' and search only for the file names. To use the `_SEARCHFILENAMEONLY_` directive, all include files must have unique names. For details, see [\\_SEARCHFILENAMEONLY\\_, on page 335](#).
  8. Click Un-Archive.
- A message dialog box is displayed while the files are being extracted.
9. If the destination directory already contains project files with the same name as the files you are extracting, you are prompted so that the existing files can be overwritten by the extracted files.



## Copy a Project

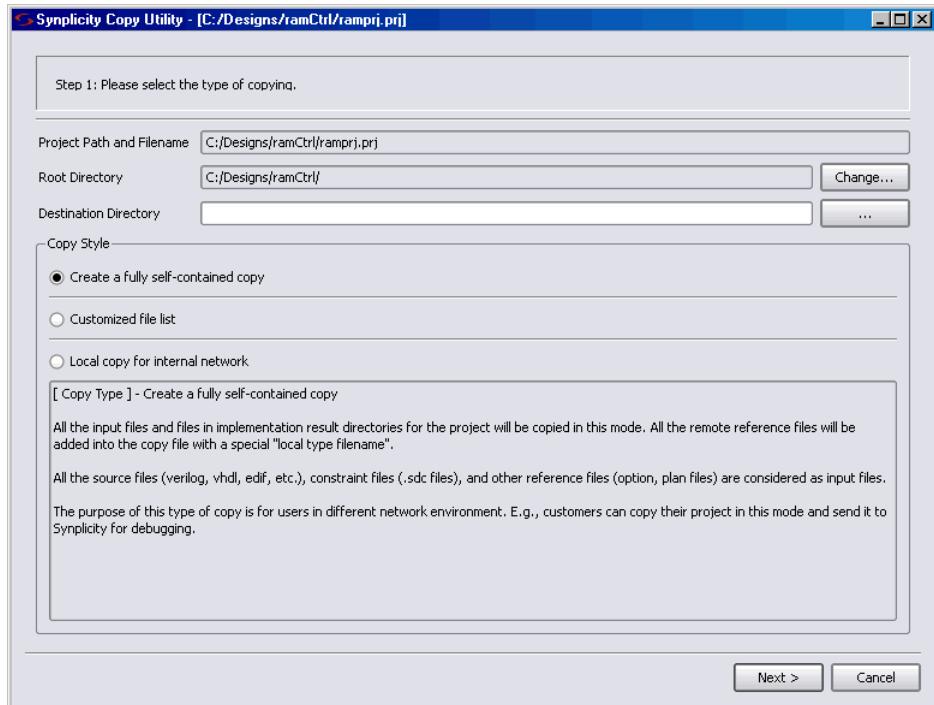
Use this utility to create an unarchived copy of a design project. You can copy an entire project or just selected files from the project. However, if you want to create an archive of the project, where the entire project is stored as a single file, see [Archive a Project, on page 100](#).

Here are the steps to create a copy of a design project:

1. From the Project view, select Project->Copy Project.

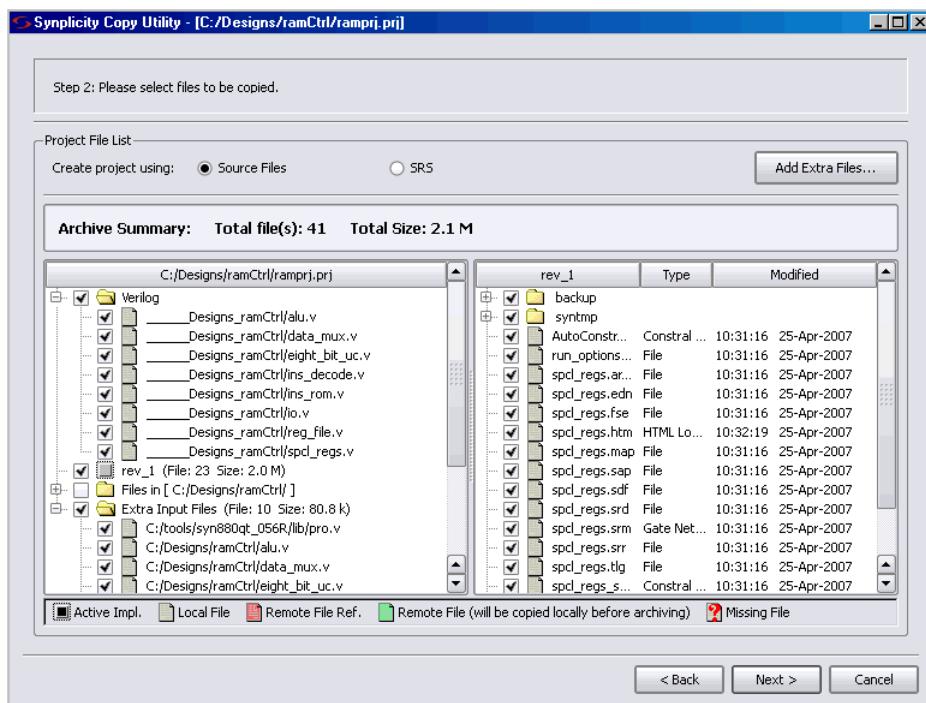
The Tcl command equivalent is project `-copy`. For a complete description of the project Tcl command options for archiving, see [project, on page 66](#) of the *Reference Manual*.

This command automatically runs a syntax check on the active project (Run->Syntax Check command) to ensure that a complete list of project files is generated. If you have Verilog include files in your project, they are included. The utility runs this check for each implementation in the project to ensure that the file list is complete for each implementation and then displays the wizard, which contains the name of the project and other information.

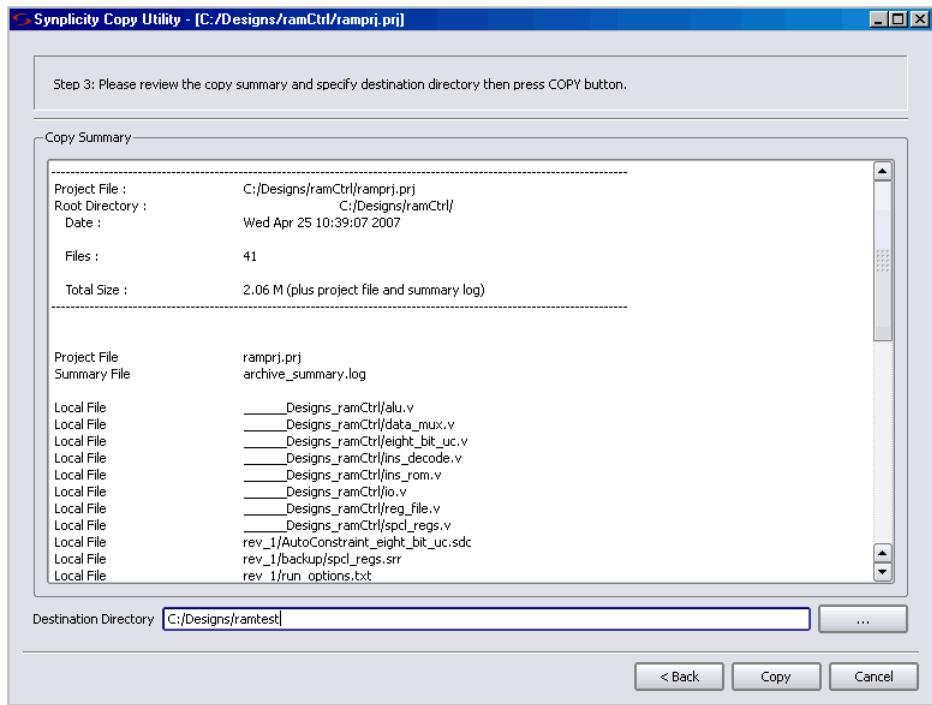


## 2. Do the following in the wizard:

- Specify the destination directory where you want to copy the files.
- Select the files to copy. You can choose to copy all the project files; one or more individual files, input files only, or customize the list to be copied.
- To specify a custom list of files, enable Customized file list. Use the check boxes to include or exclude files from the copy. Enable SRS if you want to copy all srs files (RTL schematics). You cannot enable the Source Files option if you select this. Use the Add Extra Files button to include additional files in the project.



- Click Next.



### 3. Do the following:

- Verify the copy information.
- Enter a destination directory. If the directory does not exist it will be created.
- Click Copy.

This creates the project copy.

## Support for Hierarchical Include Paths

The archive utility can support various forms of include path hierarchies to locate files for a project. For example:

- The include path can be relative to the location of the source file.

```
block_a/  
block_a.v -> `include "block_a.h"
```

- The include path can be a relative path outside of the project.

```
block_b.v -> `include "../../block_b.h"
```

The archive utility can determine the absolute path for the file from the relative path as shown below:

```
remote/sbg_pe/tests/feature_flow/include/block_b.h
```

After unarchiving the project, you can see the directory structure for the equivalent absolute path relative to the project.

```
"./remote/sbg_pe/tests/feature_flow/include/block_b.h"
```

- The file location can be specified by `include_path` in the project file.

```
block_c/  
block_c.v -> `include "block_c.h"
```

Where this file is located in the directory `/include1/`.

- The include path can be an absolute path outside of the project.

```
block_d/  
block_d.v -> `include "/slowfs/sbg/tests/include2/block_d.h"
```

When you archive the project, the absolute path becomes a relative path. After unarchiving the project, you can see the directory structure for the relative path to the project.

```
"./slowfs/sbg/tests/include2/block_d.h"
```

- The file location can be specified by `include_path` in the project file.

```
top_block/  
top_block.v -> `include "top_block.h"
```

Where the `top_block.v` file is located in the directory `/include2/`.

- Any additional search paths specified in the project file are copied and included as relative paths to the project.

After you archive and unarchive the project, the relative paths in the original project become absolute paths in the new unarchived project. In the project file, the `set_option -include_path` preserves the original search order for the files.

## Using the `_SEARCHFILENAMEONLY_` Compiler Directive

Whenever you have a SAR file that contains relative or absolute include paths for the files in the project, you can also use the `_SEARCHFILENAMEONLY_` directive to have the compiler remove the relative/absolute paths from the 'include and search only for the file names. Otherwise, you may have problems using the archive utility. For details, see [\\_SEARCHFILENAMEONLY\\_, on page 335](#).



## CHAPTER 5

# Specifying Constraints

---

This chapter describes how to specify constraints for your design. It covers the following:

- [Using the SCOPE Editor](#), on page 114
- [Specifying SCOPE Constraints](#), on page 121
- [Specifying Timing Exceptions](#), on page 132
- [Finding Objects with Tcl find and expand](#), on page 138
- [Using Collections](#), on page 147

The following chapters discuss related information:

- [Chapter 4, Constraint Commands](#) (*Command Reference Manual*), for an overview of constraints.
- [Chapter 5, SCOPE Constraints Editor](#) (*Reference Manual*), for a description of the SCOPE editor.

# Using the SCOPE Editor

The SCOPE (Synthesis Constraints OPtimization Environment®) presents a spreadsheet-like editor with a number of panels for entering and managing timing constraints and synthesis attributes. The SCOPE GUI is good for editing most constraints, but there are some constraints (like black box constraints) which can only be entered as directives in the source files. The SCOPE GUI also includes an advanced text editor that can help you edit constraints easily.

These constraints are saved to the FPGA Design Constraint (FDC) file. The FDC file contains *Synopsys SDC Standard* timing constraints (for example, `create_clock`, `set_input_delay`, and `set_false_path`), along with the non-timing constraints (design constraints) (for example, `define_attribute`, `define_scope_collection`). When working with these constraints, use the following processes:

- For new designs, use the SCOPE editor. See [Creating Constraints in the SCOPE Editor, on page 114](#) for more information.
- For new designs, use the `create_fdc_template` Tcl command. See [Creating Constraints With the FDC Template Command, on page 118](#) for details.

## Creating Constraints in the SCOPE Editor

The following procedure shows you how to use the SCOPE editor to create constraints for the FDC constraint file.

1. To create a new constraint file, follow these steps:

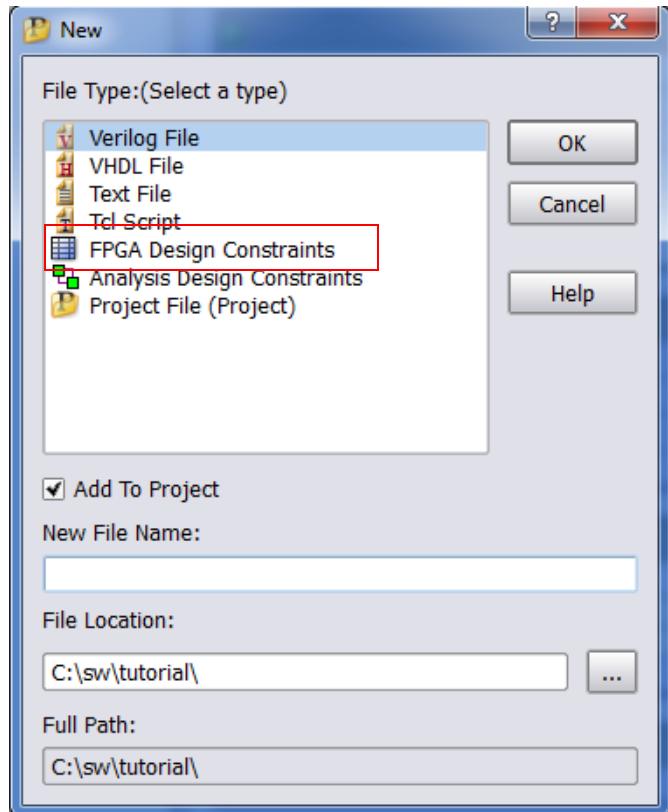
- Compile the design (F7).
- Open the SCOPE window by:

Clicking the SCOPE icon in the toolbar (  ).

This brings up the New Constraint File dialog box.

OR

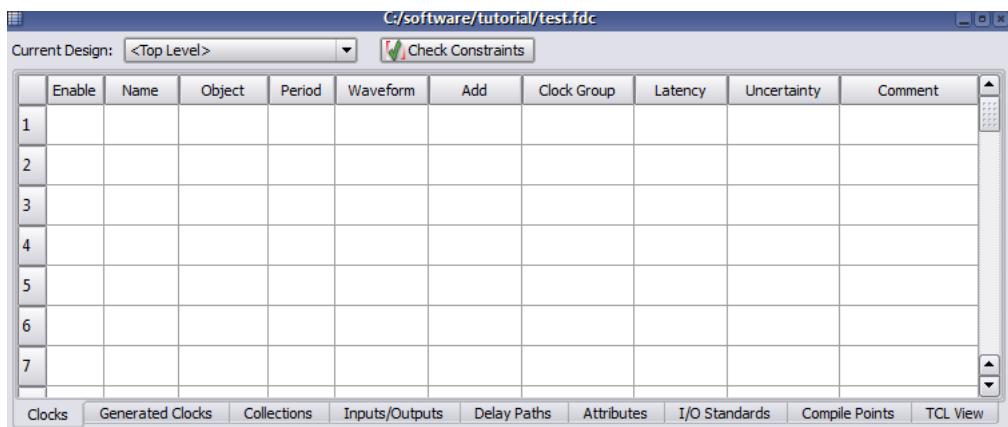
Pressing Ctrl-n or selecting File -> New. This brings up the New dialog box.



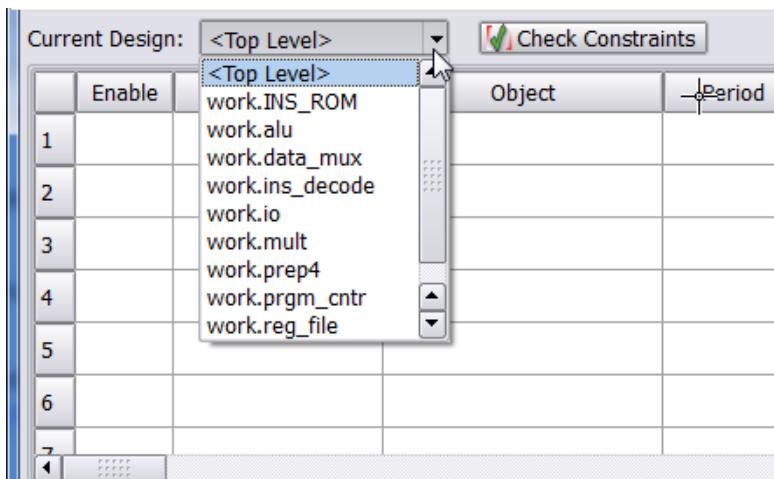
Both of these methods open the SCOPE editor GUI.

2. To open an existing file, do one of the following:
  - Double-click the file from the Project view.
  - Press Ctrl-o or select File->Open. In the dialog box, set the kind of file you want to open to Constraint Files (SCOPE) (fdc), and double-click to select the file from the list.

An empty SCOPE spreadsheet window opens. The tabs along the bottom of the SCOPE window list the different kinds of constraints you can add. For each kind of constraint, the columns contain specific data.



3. Select if you want to apply the constraint to the top-level or for modules from the Current Design option drop-down menu located at the top of the SCOPE editor.



4. You can enter or edit the following types of constraints:

- Timing constraints - on the Clocks, Generated Clocks, Inputs/Outputs, Registers, or Delay Paths tab.
- Design constraints - on the Collections, Attributes, I/O Standards, or Compile Points tab.

For details about these constraints, see [Specifying SCOPE Constraints, on page 121](#).

For information about ways to enter constraints within the SCOPE editor, see [Guidelines for Entering and Editing Constraints, on page 130](#).

5. The free form constraint editor is located in the TCL View tab, which is the last tab in SCOPE. The text editor has a help window on the right-hand side. For more information about this text editor, see [Using the TCL View of SCOPE GUI, on page 126](#).
6. Click on the Check Constraints button to run the constraint checker. The output provides information on how the constraints are interpreted by the tool.

All constraint information is saved in the same FPGA Design Constraint file (FDC) with clearly marked beginning and ending for each section. Do not manually modify these pre-defined SCOPE sections.

The following example shows the contents of an FDC file.

```
#####
# FDC constraints translated from Synplify Legacy Timing & Design Constraints
#####

set_rtl_ff_names {#}
##### BEGIN Header

# Synopsys, Inc. constraint file
# D:\bugs\timing_88\clk_prior\scratch\top.fdc
# Written on Wed Jun 20 10:50:15 2012
# by Synplify Premier with Design Planner, G-2012.09 FDC Constraint Editor

# Custom constraint commands may be added outside of the SCOPE tab sections bounded with BEGIN/END.
# These sections are generated from SCOPE spreadsheet tabs.

##### END Header

##### BEGIN Clocks - (Populated from tab in SCOPE, 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}
set_clock_groups -derive -name default_clkgroup_0 -asynchronous -group [get_clocks {clka}]
set_clock_groups -derive -name default_clkgroup_1 -asynchronous -group [get_clocks {clkb}]
##### END Clocks

##### BEGIN "Generated Clocks" - (Populated from tab in SCOPE, do not edit)
##### END "Generated Clocks"

##### BEGIN Collections - (Populated from tab in SCOPE, do not edit)
define_scope_collection all_inputs_fdc {find -port * -filter @direction==input}
define_scope_collection all_outputs_fdc {find -port * -filter @direction==output}
define_scope_collection all_clocks_fdc {find -hier -clock *}
define_scope_collection all_registers_fdc {find -hier -seq *}
define_scope_collection all_grp {define_collection [find -inst {i:FirstStbPhase}] [find -inst {i:Normal}]
define_scope_collection fdc_cmd_0 {find -seq {y*.q[*]}}
define_scope_collection fdc_cmd_1 {find {n:foo}}
define_scope_collection fdc_cmd_2 {expand -hier -seq -from $fdc_cmd_1}
##### END Collections

##### BEGIN Inputs/Outputs - (Populated from tab in SCOPE, do not edit)
set_input_delay -clock {c:clka} -clock_fall -add_delay 0.000 $all_inputs_fdc
set_output_delay -clock {c:clka} -add_delay 0.000 $all_outputs_fdc
set_input_delay -clock {c:clka} -add_delay 2.00 {p:a[7:0]}
set_input_delay -clock {c:clka} -add_delay 0 {p:rst}
##### END Inputs/Outputs

##### BEGIN "Delay Paths" - (Populated from tab in SCOPE, do not edit)
set_multicycle_path 3 -end -from $fdc_cmd_0
set_false_path -comment {false foo[0] free{iddhh}} -from {b:ena}
set_false_path -from $fdc_cmd_2 -to {i:abc.def.g_reg} -through {n:bar}
set_false_path -from {$boing} -to {c:dcm|clk0fx_derived_clock[2]} -through {n:fudge}
set_false_path -from {c:dcm|clk0fx_derived_clock} -to {i:abc.def.g_reg[0] i:abc} -through {n:fudge}
##### END "Delay Paths"
```

## Creating Constraints With the FDC Template Command

Use the Tcl command `create_fdc_template` to create an initial constraint file (fdc) for your specific design. This command lets you specify port clocks, I/O delays, and initial `set_clock_groups` for the clocks for which text headers are generated that can help guide you when creating this constraint.

The following procedure shows you how to create constraints in the FDC constraints file with the `create_fdc_template` command:

1. Create a project for your design.
2. Compile the design.
3. At the command line, for example, you can specify the following:

```
create_fdc_template -period 10 -out_delay 1.5
```

The command automatically updates your project to reflect the new constraint file(s). Do **Ctrl+s** to save the new settings.



4. If you open the SCOPE editor, you can check that the clock period and output delay values were added to the constraint file as shown in the following figure.

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	Uncertainty	Comment	
1	<input checked="" type="checkbox"/>	clock	[get_ports {p:clock}]	10	0 5.0	<input type="checkbox"/>	<default>				
2											
3											
4	<input checked="" type="checkbox"/>	input	p:porta[7:0]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0					
5	<input checked="" type="checkbox"/>	input	p:portb[7:0]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0					
6	<input checked="" type="checkbox"/>	input	p:portc[7:0]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0					
7	<input checked="" type="checkbox"/>	input	p:resetn	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0					
Clocks											
5	<input checked="" type="checkbox"/>	output	p:porta[7:0]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1.5					
6	<input checked="" type="checkbox"/>	output	p:portb[7:0]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1.5					
7	<input checked="" type="checkbox"/>	output	p:portc[7:0]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1.5					

Inputs/Outputs

5. Each port clock includes a `set_clock_groups` header with details displayed in the **TCL View**, which can help you determine whether clocks have been optimized away or if there are any derived clocks.

However, if there is only one clock port and no derived clocks, no explicit clock groups are created since they are not needed, as shown below.

```
#####
### Individual "set_clock_groups" commands for all "clock" 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 "clock":
###   clock DERIVED CLOCKS:
#####
```

For details about the command syntax, see [create\\_fdc\\_template, on page 38](#).

6. You can continue using the SCOPE editor to create other constraints.
7. Save the constraint file.

# Specifying SCOPE Constraints

Timing constraints define the performance goals for a design. The FPGA synthesis tool supports a subset of the Synopsys SDC Standard timing constraints (for example, `create_clock`, `set_input_delay`, and `set_false_path`). For additional support, see [Synopsys Standard Timing Constraints, on page 122](#).

Design constraints let you add attributes, define collections and specify constraints for them, and select specific I/O standard pad types for your design.

You can define both timing and design constraints in the SCOPE editor. For the different types of constraints, see the following topics:

- [Entering and Editing SCOPE Constraints](#)
- [Setting Clock and Path Constraints](#)
- [Defining Input and Output Constraints](#)
- [Specifying Standard I/O Pad Types](#)

To set constraints for timing exceptions like false paths and multicycle paths, see [Specifying Timing Exceptions, on page 132](#).

For information about collections, see [Using Collections, on page 147](#).

## Entering and Editing SCOPE Constraints

This section contains a description of the timing and design constraints you can enter in the SCOPE GUI that are saved to an FDC file. The SCOPE timing constraint panels include:

SCOPE Panel	See...	Tcl Commands
Clocks	<a href="#">Clocks</a>	<code>create_clock</code> <code>set_clock_groups</code> <code>set_clock_latency</code> <code>set_clock_uncertainty</code>
Generated Clocks	<a href="#">Generated Clocks</a>	<code>create_generated_clock</code>
Collections	<a href="#">Collections</a>	<code>define_scope_collection</code>

SCOPE Panel	See...	Tcl Commands
Inputs/Outputs	<a href="#">Inputs/Outputs</a>	<code>set_input_delay</code> <code>set_output_delay</code>
Registers	<a href="#">Registers</a>	<code>set_reg_input_delay</code> <code>set_reg_output_delay</code>
Delay Paths	<a href="#">Delay Paths</a>	<code>set_false_path</code> <code>set_max_delay</code> <code>set_multicycle_path</code>
Attributes	<a href="#">Attributes</a>	<code>define_attribute</code> <code>define_global_attribute</code>
Compile Points	<a href="#">Compile Points</a>	<code>define_compile_point</code> <code>define_current_design</code>
TCL View	<a href="#">TCL View</a>	--

## Synopsys Standard Timing Constraints

The FPGA synthesis tools support Synopsys standard timing constraints for a subset of the clock definition (Clocks and Generated Clocks), I/O delay (Inputs/Outputs), and timing exception constraints (Delay Paths).

## Setting Clock and Path Constraints

The following table summarizes how to set different clock and path constraints from the SCOPE window.

To define...	Pane	Do this to set the constraint...
Clocks	Clock	<p>Select the clock object (Clock).</p> <p>Specify a clock name (Clock Alias), if required.</p> <p>Type a period (Period).</p> <p>Change the rise and fall edge times for the clock waveforms of the clock in nanoseconds, if needed.</p> <p>Change the default clock group, if needed.</p> <p>Check the Enabled box.</p> <p>See <a href="#">Setting Clock and Path Constraints, on page 123</a> for information about clock attributes.</p>
Generated Clocks	Generated Clocks	<p>Select the generated clock object.</p> <p>Specify the master clock source (a clock source pin in the design).</p> <p>Specify whether to use invert for the generated clock signal.</p> <p>Specify whether to use: edges, divide_by, or multiply_by.</p> <p>Check the Enabled box.</p>
Input/output delays	Inputs/Outputs	See <a href="#">Defining Input and Output Constraints, on page 124</a> for information about setting I/O constraints.
Maximum path delay	Delay Paths	<p>Select the Delay Type path of Max Delay.</p> <p>Select the start/from point for either a port or register (From/Through). See <a href="#">Defining From/To/Through Points for Timing Exceptions, on page 132</a> for more information.</p> <p>Select the end/to point for either an output port or register. Specify a through point for a net or hierarchical port/pin (To/Through).</p> <p>Set the delay value (Max Delay).</p> <p>Check the Enabled box.</p>
Multicycle paths	Delay Paths	See <a href="#">Defining Multicycle Paths, on page 136</a> .

To define...	Pane	Do this to set the constraint...
False paths	Delay Paths	See <a href="#">Defining False Paths, on page 137</a> for details.
Global attributes	Attributes	Set Object Type to <global>. Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.
Attributes	Attributes	Do either of the following: <ul style="list-style-type: none"> <li>Select the type of object (Object Type). Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.</li> <li>Set the attribute (Attribute) and its value (Value). Select the object (Object). Check the Enabled box.</li> </ul>

## Defining Input and Output Constraints

In addition to setting I/O delays in the SCOPE window as described in [Setting Clock and Path Constraints, on page 123](#), you can also set the Use clock period for unconstrained IO option.

- Open the SCOPE window, click Inputs/Outputs, and select the port (Port). You can set the constraint for
  - All inputs and outputs (globally in the top-level netlist)
  - For a whole bus
  - For single bits

You can specify multiple constraints for the same port. The software applies all the constraints; the tightest constraint determines the worst slack. If there are multiple constraints from different levels, the most specific overrides the more global. For example, 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.

- Specify the constraint value in the SCOPE window:
  - Select the type of delay: input or output (**Type**).
  - Type a delay value (**Value**).
  - Check the **Enabled** box, and save the constraint file in the project.
- Make sure to specify explicit constraints for each I/O path you want to constrain.
- To determine how the I/O constraints are used during synthesis, do the following:
  - Select Project->Implementation Options, and click **Constraints**.
  - To use only the explicitly defined constraints disable **Use clock period for unconstrained IO**.
  - To synthesize with all the constraints, using the clock period for all I/O paths that do not have an explicit constraint enable **Use clock period for unconstrained IO**.
  - Synthesize the design. When you forward-annotate the constraints, the constraints used for synthesis are forward-annotated for place-and-route.
- Input or output ports with explicitly defined constraints, but without a reference clock (-ref option) are included in the System clock domain and are considered to belong to every defined or inferred clock group.
- If you do not meet timing goals after place-and-route and you need to adjust the input constraints; do the following:
  - Open the SCOPE window with the input constraint.
  - Use the `set_clock_route_delay` command to translate the `route` option for the constraint, so that you can specify the actual route delay in nanoseconds, as obtained from the place-and-route results. Adding this constraint is equivalent to putting a register delay on the input register. Use it as a fudge factor to force the synthesis engine to accommodate a routing delay that is larger than expected.
  - Resynthesize your design.

## Specifying Standard I/O Pad Types

You can specify a standard I/O pad type to use in the design.

1. Open the SCOPE window and go to the I/O Standard tab.
2. In the Port column, select the port. This determines the port type in the Type column.
3. Enter an appropriate I/O pad type in the I/O Standard column. The Description column shows a description of the I/O standard you selected.  
For details of supported I/O standards, see [Industry I/O Standards, on page 171](#).
4. Where applicable, set other parameters like drive strength, slew rate, and termination.

You cannot set these parameter values for industry I/O standards whose parameters are defined by the standard.

The software stores the pad type specification and the parameter values in the syn\_pad\_type attribute. When you synthesize the design, the I/O specifications are mapped to the appropriate I/O pads within the technology.

## Using the TCL View of SCOPE GUI

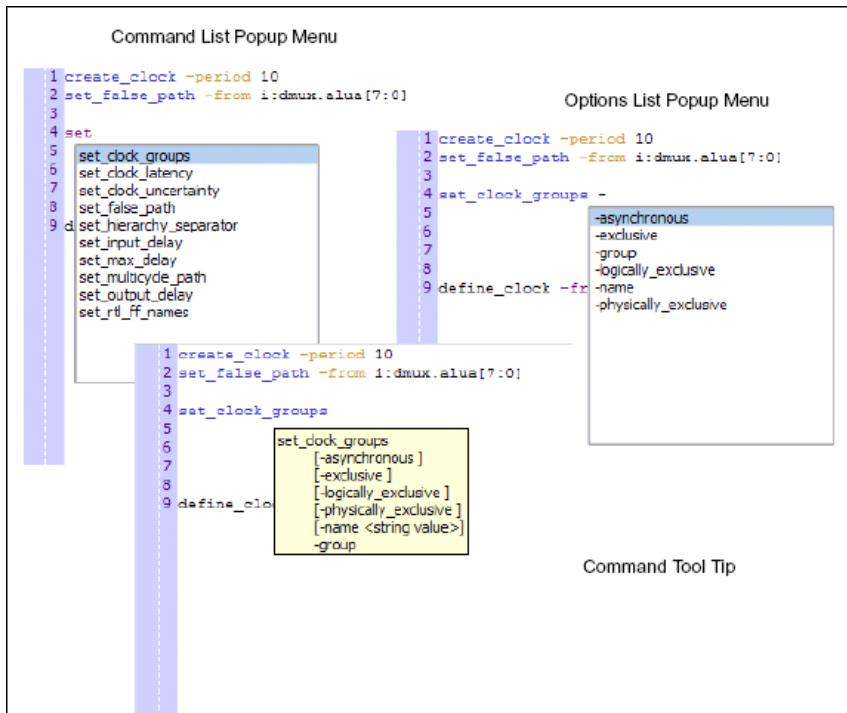
The TCL View of the SCOPE GUI is an advanced text file editor used for FPGA timing and design constraints. This text editor provides the following capabilities:

- 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

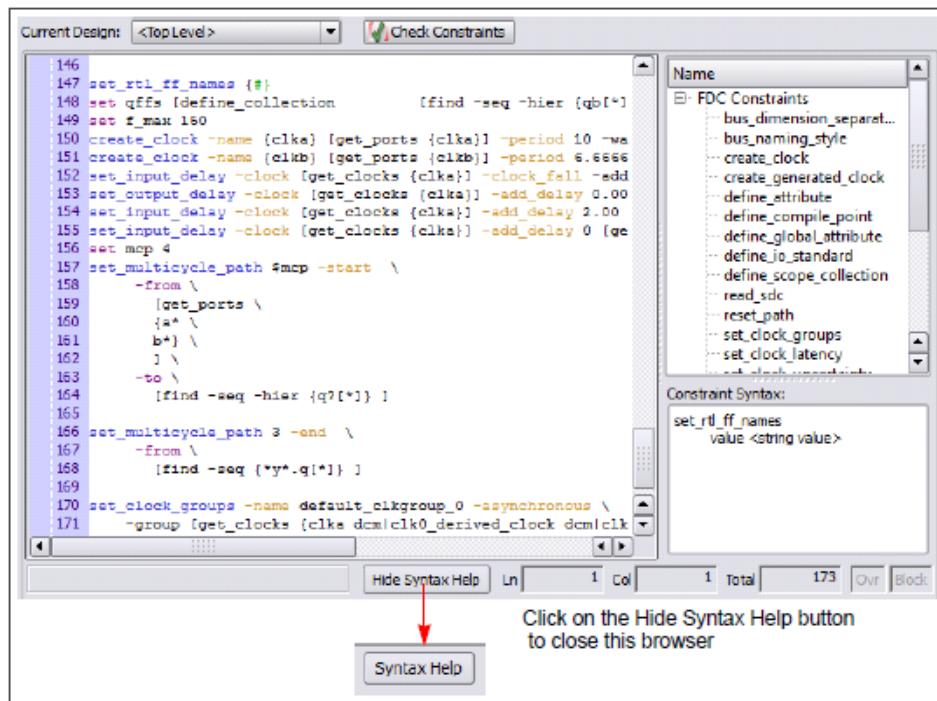
- Checks command syntax and uses color indicators that
  - Validates commands and command syntax
  - Distinguishes between FPGA design constraints and SCOPE legacy constraints
- Allows for standard editor commands, such as copy, paste, comment/un-comment a group of lines, and highlighting of keywords

To use the TCL View of the SCOPE GUI:

1. Click the TCL View of the SCOPE GUI.
2. You can specify FPGA design constraints as follows:
  - Type the command; after you type three characters a popup menu displays the design constraint command list. Select a command.
  - When you type a dash (-), the options popup menu list is displayed. Select an option.
  - When you hover over a command, a tool tip is displayed for the selected commands.



3. You can also specify a command by using the constraints browser that displays a constraints command list and associated syntax.
  - Double-click the specified constraint to add the command to the editor window.
  - Then, use the constraint syntax window to help you specify the options for this command.
  - Click the Hide Syntax Help button at the bottom of the editor window to close the syntax help browser.



- When you save this file, the constraint file is added to your project in the Constraint directory if the Add to Project option is checked on the New dialog box. Thereafter, you can double-click on the FDC constraint file to open it in the text editor.

## Guidelines for Entering and Editing Constraints

1. Enter or edit constraints as follows:

- For attribute cells in the spreadsheet, click in the cell and select from the pull-down list of available choices.
- For object cells in the spreadsheet, click in the cell and select from the pull-down list. When you select from the list, the objects automatically have the proper prefixes in the SCOPE window.

Alternatively, you can drag and drop an object from an HDL Analyst view into the cell, or type in a name. If you drag a bus, the software enters the whole bus (busA). To enter busA[3:0], select the appropriate bus bits before you drag and drop them. If you drag and drop or type a name, make sure that the object has the proper prefix identifiers:

Prefix Identifiers	Description for...
v: <i>design_name</i>	hierarchies or “views” (modules)
c: <i>clock_name</i>	clocks
i: <i>instance_name</i>	instances (blocks)
p: <i>port_name</i>	ports (off-chip)
t: <i>pin_name</i>	hierarchical ports, and pins of instantiated cells
b: <i>name</i>	bits of a bus (port)
n: <i>net_name</i>	internal nets

- For cells with values, type in the value or select from the pull-down list.
- Click the check box in the Enabled column to enable the constraint or attribute.
- Make sure you have entered all the essential information for that constraint. Scroll horizontally to check. For example, to set a clock constraint in the Clocks tab, you must fill out Enabled, Clock, Period, and Clock Group. The other columns are optional. For details about setting different kinds of constraints, go to the appropriate section listed in [Specifying SCOPE Constraints, on page 121](#).

2. For common editing operations, refer to this table:

To...	Do...
Cut, copy, paste, undo, or redo	Select the command from the popup (hold down the right mouse button to get the popup) or from the Edit menu.
Copy the same value down a column	Select Fill Down (Ctrl-d) from the Edit or popup menus.
Insert or delete rows	Select Insert Row or Delete Rows from the Edit or popup menus.
Find text	Select Find from the Edit or popup menus. Type the text you want to find, and click OK.

3. Edit your constraint file if needed. If your naming conventions do not match these defaults, add the appropriate command specifying your naming convention to the beginning of the file, as shown in these examples:

	Default	You use	Add this to your file
Hierarchy separator	A.B	Slash: A/B	set_hierarchy_separator {/}
Naming bit 5 of bus ABC	ABC[5]	Underscore	bus_naming_style {%s_%d}
Naming row 2 bit 3 of array ABC [2x16]	ABC [2] [3]	Underscore ABC[ 2_3 ]	bus_dimension_separator_style {}

# Specifying Timing Exceptions

You can specify the following timing exception constraints, either from the SCOPE interface or by manually entering the Tcl commands in a file:

- Multicycle Paths - Paths with multiple clock cycles.
- False Paths - Clock paths that you want the synthesis tool to ignore during timing analysis and assign low (or no) priority during optimization.
- Max Delay Paths - Point-to-point delay constraints for paths.

The following shows you how to specify timing exceptions in the SCOPE GUI. For the equivalent Tcl syntax, see [Chapter 8, Scripts](#) in the *Reference Manual*.

- [Defining From/To/Through Points for Timing Exceptions](#), on page 132
- [Defining Multicycle Paths](#), on page 136
- [Defining False Paths](#), on page 137

For information about resolving timing exception conflicts, see [Conflict Resolution for Timing Exceptions](#), on page 187 in the *Reference Manual*.

## Defining From/To/Through Points for Timing Exceptions

For multi-cycle path, false path, and maximum path delay constraints, you must define paths with a combination of From/To/Through points. Whenever the tool encounters a conflict in the way timing-exception constraints are written, see [Conflict Resolution for Timing Exceptions](#), on page 187 to determine how resolution occurs based on the priorities defined.

The following guidelines provide details for defining these constraints. You must specify at least one From, To, or Through point.

- In the From field, identify the starting point for the path. The starting point can be a clock, input or bidirectional port, or register. Only black box output pins are valid. To specify multiple starting points:
  - Such as the bits of a bus, enclose them in square brackets: A[15:0] or A[\*].

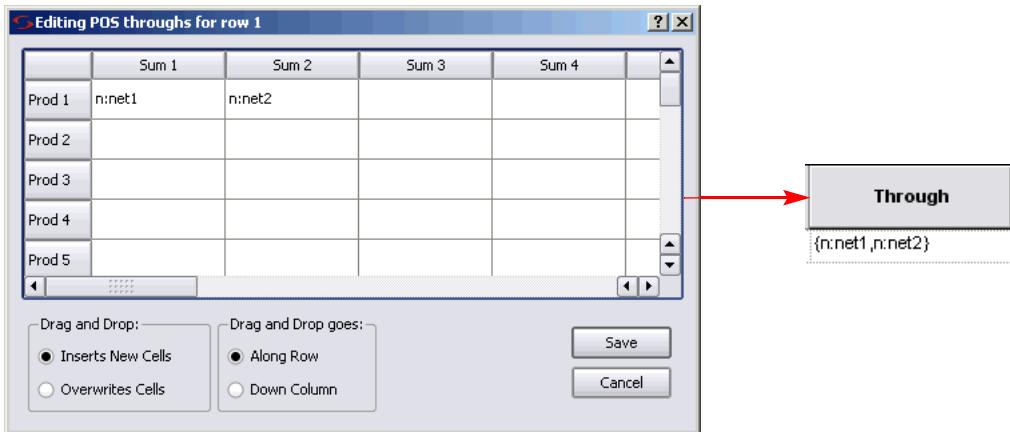
- Select the first start point from the HDL Analyst view, then drag and drop this instance into the From cell in SCOPE. For each subsequent instance, press the Shift key as you drag and drop the instance into the From cell in SCOPE. For example, valid Tcl command format include:

```
set_multicycle_path -from {i:aq i:bq} 2  
set_multicycle_path -from [i:aq i:bq] -through {n:xor_all} 2
```

- In the To field, identify the ending point for the path. The ending point can be a clock, output or bidirectional port, or register. Only black box input pins are valid. To specify multiple ending points, such as the bits of a bus, enclose them in square brackets: B[15:0].
- A single through point can be a combinational net, hierarchical port or instantiated cell pin. To specify a net:
  - Click in the Through field and click the arrow. This opens the Product of Sums (POS) interface.
  - Either type the net name with the n: prefix in the first cell or drag the net from an HDL Analyst view into the cell.
  - Click Save.

For example, if you specify n:net1, the constraint applies to any path passing through net1.

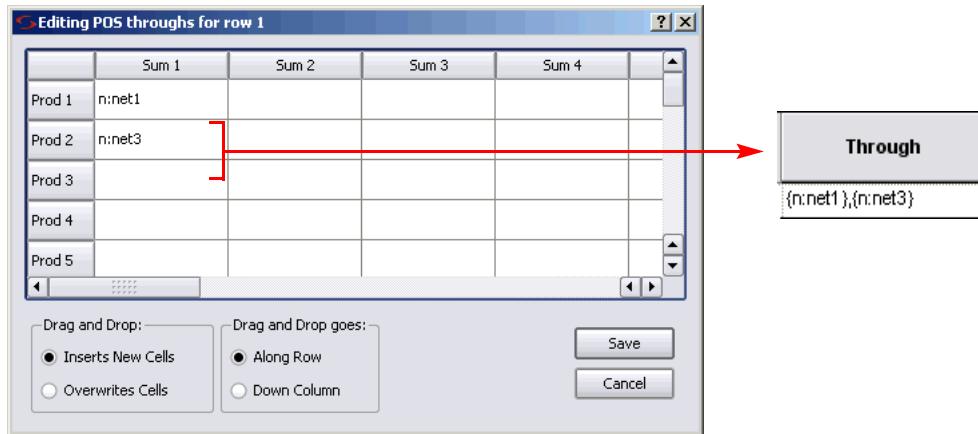
- To specify an OR when constraining a list of through points, you can type the net names in the Through field or you can use the POS UI. To do this:
  - Click in the Through field and click the arrow. This opens the Product of Sums interface.
  - Either type the first net name in a cell in a Prod row or drag the net from an 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.



- Alternatively, select Along Row in the SCOPE POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names along the row. The nets in each row form an OR list.
- Click Save.

The constraint works as an OR function and applies to any path passing through any of the specified nets. In the example shown in the previous figure, the constraint applies to any path that passes through net1 or net2.

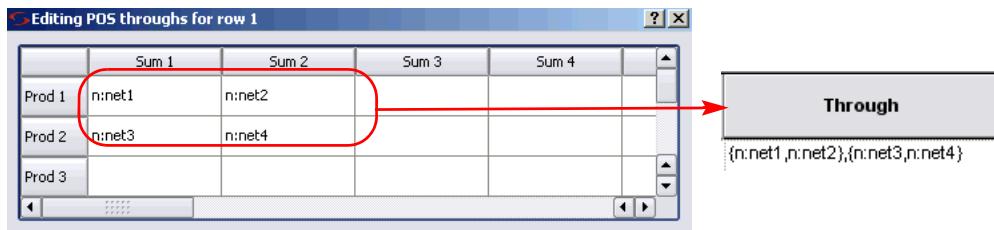
- To specify an AND when constraining a list of through points, type the names in the Through field or do the following:
  - Open the Product of Sums interface as described previously.
  - Either type the first net name in the first cell in a Sum column or drag the net from an HDL Analyst view into the cell. Repeat this step down the same Sum column.



- Alternatively, select Down Column in the SCOPE POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names down the column.

The constraint works as an AND function and applies to any path passing through all the specified nets. In the previous figure, the constraint applies to any path that passes through *net1 and net3*.

- To specify an AND/OR constraint for a list of through points, type the names in the Through field (see the following figure) or do the following:
  - Create multiple lists as described previously.
  - Click Save.



In this example, the synthesis tool applies the constraint to the paths through all points in the lists as follows:

```
net1 AND net3  
OR net1 AND net4  
OR net2 AND net3  
OR net2 AND net4
```

## Defining Multicycle Paths

To define a multicycle path constraint, use the Tcl `set_multicycle_path` command, or select the SCOPE Delay Paths tab and do the following;

1. From the Delay Type pull-down menu, select Multicycle.
2. Select a port or register in the From or To columns, or a net in the Through column. You must set at least one From, To, or Through point. You can use a combination of these points. See [Defining From/To/Through Points for Timing Exceptions, on page 132](#) for more information.
3. Select another port or register if needed (From/To/Through).
4. Type the number of clock cycles or nets (Cycles).
5. Specify the clock period to use for the constraint by going to the Start/End column and selecting either Start or End.

If you do not explicitly specify a clock period, the software uses the end clock period. The constraint is now calculated as follows:

$$\text{multicycle\_distance} = \text{clock\_distance} + (\text{cycles} - 1) * \text{reference\_clock\_period}$$

In the equation, `clock_distance` is the shortest distance between the triggering edges of the start and end clocks, `cycles` is the number of clock cycles specified, and `reference_clock_period` is either the specified start clock period or the default end clock period.

6. Check the Enabled box.

## Defining False Paths

You define false paths by setting constraints explicitly on the Delay Paths tab or implicitly on the Clock tab. See [Defining From/To/Through Points for Timing Exceptions, on page 132](#) for object naming and specifying through points.

- To define a false path between ports or registers, select the SCOPE Delay Paths tab, and do the following:
  - From the Delay Type pull-down menu, select False.
  - Use the pull-down to select the port or register from the appropriate column (From/To/Through).
  - Check the Enabled box.

The software treats this as an explicit false constraint and assigns it the highest priority. Any other constraints on this path are ignored.

- To define a false path between two clocks, select the SCOPE Clocks tab, and assign the clocks to different clock groups:

The software implicitly assumes a false path between clocks in different clock groups. This false path constraint can be overridden by a maximum path delay constraint, or with an explicit constraint.

- To set an implicit false path on a path to/from an I/O port, do the following:
  - Select Project->Implementation Options->Constraints.
  - Disable Use clock period for unconstrained IO.

# Finding Objects with Tcl find and expand

The `Tcl find` and `expand` commands are powerful search tools that you can use to quickly identify the objects you want. The following sections describe how to use these commands effectively:

- [Specifying Search Patterns for Tcl find](#), on page 138
- [Refining Tcl Find Results with -filter](#), on page 140
- [Using the Tcl Find Command to Define Collections](#), on page 143
- [Using the Tcl expand Command to Define Collections](#), on page 144
- [Checking Tcl find and expand Results](#), on page 145
- [Using Tcl find and expand in Batch Mode](#), on page 147

Once you have located objects with the `find` or `expand` commands, you can group them into collections, as described in [Using Collections, on page 147](#), and apply constraints to all the objects in the collection at the same time.

## Specifying Search Patterns for Tcl find

The usage tips in the following table apply for `Tcl find` search patterns, regardless of whether you specify the `find` command in the SCOPE window or as a `Tcl` command. For full details of the command syntax, refer to [Tcl Find Syntax, on page 119](#) of the *Reference Manual*.

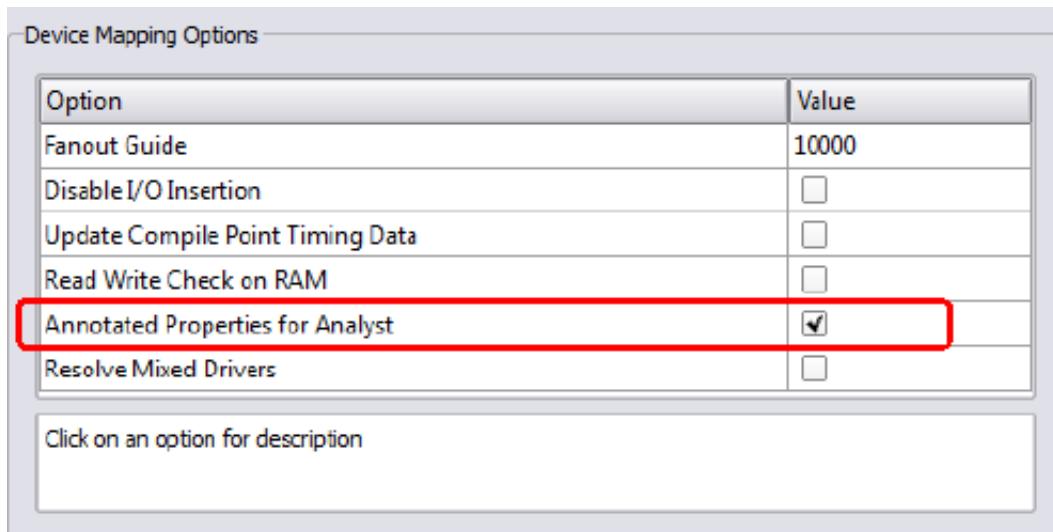
Case rules	<p>Use the case rules for the language from which the object was generated:</p> <ul style="list-style-type: none"><li>• VHDL: case-insensitive</li><li>• Verilog: case-sensitive. Make sure that the object name you type in the SCOPE window matches the Verilog name.</li></ul> <p>For mixed language designs, use the case rules for the parent module. The top level for this example is VHDL, so the following command finds any object in the current view that starts with either a or A:</p> <pre>find {a*} -nocase</pre>
Pattern matching	<p>You have two pattern-matching choices:</p> <ul style="list-style-type: none"><li>• Specify the <code>-regexp</code> argument, and then use regular expressions for pattern matching.</li><li>• Do not specify <code>-regexp</code>, and use only the * and ? wildcards for pattern matching.</li></ul> <p>For hierarchical instance names that use dots as separators, the dots must be escaped with a backward slash (\). For example: abc\l.d.</p>
Scope of the search	<p>The scope of the search varies, depending on where you enter the command. If you enter it in the SCOPE environment, the scope of the search is the entire database, but if it is entered in the Tcl window, the default scope of the search is the current HDL Analyst view. See <a href="#">Comparison of Methods for Defining Collections, on page 147</a> for a list of the differences.</p> <p>To set the scope to include the hierarchical levels below the current view in HDL Analyst, use the <code>-hier</code> argument. This example finds all objects below the current view that begin with a:</p> <pre>find {a*} -hier</pre>

Restricting search by type of object	Use the <i>-object_type</i> argument. The following command finds all nets that contain syn:
Restricting search by object property	Use the <i>-filter</i> option, as described in <a href="#">Refining Tcl Find Results with -filter, on page 140</a> .
Extending search through the hierarchy	Use the <i>-flat</i> option. With this option, the * wildcard matches hierarchy separators as well as regular characters. In the following example, the command finds all instances that include fft_stages in their name, whether it just matches an instance name (inst1_fft_stages_2) or matches a hierarchical name that includes the hierarchy separator (a1.fft_stages_xy): <code>find -seq -flat *fft_stages* -print</code>

## Refining Tcl Find Results with -filter

The *-filter* option of the `find` command lets you further refine the objects located by the `find` command, according to their properties. When used with other commands, it can be a powerful tool for generating statistics and for evaluation. To filter your `find` results, follow these steps:

1. Enable property annotation.
  - Select Project->Implementation Options. On the Device tab, enable Annotated Properties for Analyst. Alternatively, use the equivalent Tcl command:  
`set_option -run_prop_extract 1.`



- Compile or synthesize the design. After compilation, the tool annotates the design with properties that you can specify with the -filter option, like clock pins.
2. Specify the command using the find pattern as usual, and then specify the -filter option as the last argument:

```
find searchPattern -filter expression  
find searchPattern -filter !expression
```

With this command, the tool first finds objects that match the find *search-Pattern*, and then further filters the found objects the according to the property criteria specified in *-filter expression*. Use the ! character before *expression* if you want to select objects that do not match the properties specified in the filter *expression*.

*expression* can be a property name, specified as `@propertyName`, or a property name and value pair, specified as `@propertyName operator value`.

The following example finds registers in the current view that are clocked by myclk:

```
find -seq {*} -filter {@clock==myclk}
```

For further information about the command, see the following:

For...	See
Tips on using find search patterns	<a href="#">Specifying Search Patterns for Tcl find, on page 138</a>
find syntax details	<a href="#">find, on page 119</a> in the Reference Manual
find -filter syntax details	<a href="#">find -filter, on page 127</a> in the Reference Manual

## Examples of Useful Find -filter Commands

To find...	Use a command like this example...
Instances by slack value	set slack [find -hier -inst {*} -filter @slack <= {-1.000}]
Instances with negative slack	set negFF [find -hier -inst {*} -filter @slack <= {0.0}]
Instances within a slack range	set slackRange [find -hier -inst {*} -filter @slack <= {-1.000} && @slack >= {+1.000}]
Pins by fanout value	set pinResult [find -pin *.CE -hier -filter {@fanout > 15 && @slack < 0.0} -print]
Sequential elements within a clock domain	set clk1FF [find -hier -seq * -filter {@clock==clk1}]
Sequential components by primitive type	set fdrse [find -hier -seq {*} -filter @view=={FDRSE}]

## Using the Tcl Find Command to Define Collections

It is recommended that you use the SCOPE window rather than the Tcl window described here to specify the `find` command, for the reasons described in [Comparison of Methods for Defining Collections, on page 147](#).

The Tcl `find` command returns a collection of objects. If you want to create a collection of connectivity-based objects, use the Tcl `expand` command instead of `find` ([Specifying Search Patterns for Tcl find, on page 138](#)). This section lists some tips for using the Tcl `find` command.

1. Create a collection by typing the `set` command and assigning the results to a variable. The following example finds all instances with a primitive type DFF and assigns the collection to the variable `$result`:

```
set result [find -hier -inst {*} -filter @ view == DFF]
```

The result is a random number like `s:49078472`, which is the collection of objects found. The following table lists some usage tips for specifying the `find` command. For full details of the syntax, refer to [Tcl Find Syntax, on page 119](#) of the *Reference Manual*.

2. Check your `find` constraints. See [Checking Tcl find and expand Results, on page 145](#).

3. Once you have defined the collection, you can view the objects in the collection, using one of the following methods, which are described in more detail in [Viewing and Manipulating Collections with Tcl Commands, on page 154](#):
  - Print the collection using the `-print` option to the `find` command.
  - Print the collection without carriage returns or properties, using `c_list`.
  - Print the collection in columns, with optional properties, using `c_print`.
4. To manipulate the objects in the collection, use the commands described in [Viewing and Manipulating Collections with Tcl Commands, on page 154](#).
5. Combine the Tcl `find` command with other commands:

To...	Combine with...
Create or copy objects; create collections	<code>set</code> <code>define_collection</code>
Generate reports for evaluation	<code>c_list</code> <code>c_print</code>
Generate statistics	<code>c_info</code>

## Using the Tcl `expand` Command to Define Collections

The Tcl `expand` command returns a list of objects that are logically connected between the specified expansion points. This section contains tips on using the Tcl `expand` command to generate a collection of objects that are related by their connectivity. For the syntax details, refer to [expand, on page 134](#) in the *Command Reference Manual*.

1. Specify at least one from, to, or thru point as the starting point for the command. You can use any combination of these points.

The following example expands the cone of logic between `reg1` and `reg2`.

```
expand -from {i:reg1} -to {i:reg2}
```

If you only specify a thru point, the expansion stops at sequential elements. The following example finds all elements in the transitive fanout and transitive fanin of a clock-enable net:

```
expand -thru {n:cen}
```

2. To specify the hierarchical scope of the expansion, use the `-hier` argument.

If you do not specify this argument, the command only works on the current view. The following example expands the cone of logic to `reg1`, including instances below the current level:

```
expand -hier -to {i:reg1}
```

If you only specify a `thru` point, you can use the `-level` argument to specify the number of levels of expansion. The following example finds all elements in the transitive fanout and transitive fanin of a clock-enable net across one level of hierarchy:

```
expand -thru {n:cen} -level 1
```

3. To restrict the search by type of object, use the `-object_type` argument.

The following command finds all pins driven by the specified pin.

```
expand -pin -from {t:i_and3.z}
```

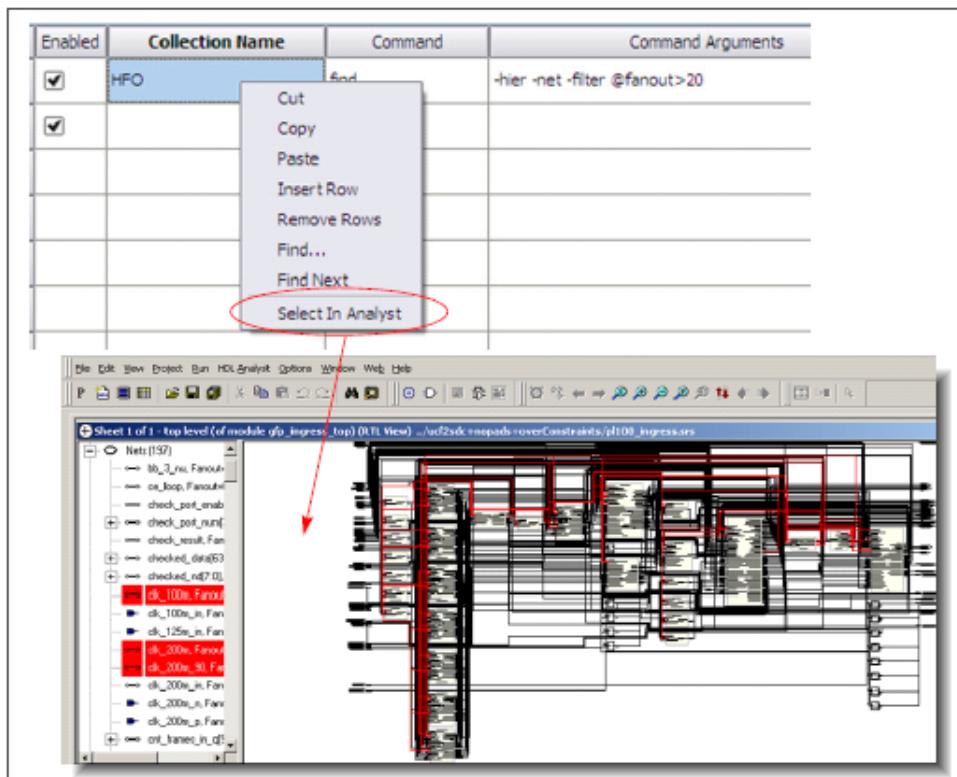
4. To print a list of the objects found, either use the `-print` argument to the `expand` command, or use the `c_print` or `c_list` commands (see [Creating Collections using Tcl Commands, on page 151](#)).

## Checking Tcl find and expand Results

You must check the validity of the find constraints you set. Use the methods described below.

1. Run the Constraints Checker, either from the UI or at the command line:
  - From the UI, select Run->Constraint Check.
  - At the command line specify the `-run constraint_check` option to the synthesis tool command. For example: `synplify_pro -batch design.prj -run constraint_check`.
  - If there are issues, the tool reports them in the `design_cck.rpt` report file. Check the Summary and Inapplicable Constraints sections in this file.

2. To list objects selected by the `find` or `expand` commands, use one of these methods:
  - List the results by specifying the `-print` option to the command.
  - List the results with the `c_list` command.
  - Print out the results one item per line, using the `c_print` command.
3. To visually validate the objects selected by the `find` or `expand` commands, do the following:
  - Run the command and save the results as a collection.
  - On the SCOPE Collections tab, select the collection.
  - Right-click and choose `Select in Analyst`. The objects in the collection are highlighted in the RTL view. The example below shows high fanout nets that drive more than 20 destinations.



## Using Tcl find and expand in Batch Mode

When you use the Tcl find command in batch mode, you must specify the open\_design command before the find or expand commands.

1. Create the Tcl file to be run in batch mode, making sure that the open\_design command precedes the find/expand commands you want.

You cannot include the Tcl find command in Timing Analyzer scripts. Instead, run Tcl Find to TXT command and use the results.

2. Run the script at the command line. For example, if the file created in step 1 was called analysis.tcl, specify it at the command line, as shown below:

```
synplify_pro -batch analysis.tcl
```

The tool generates two text files as specified, with the results of the two searches.

## Using Collections

A collection is a defined group of objects. The advantage offered by collections is that you can operate on all the objects in the collection at the same time. A collection can consist of a single object, multiple objects, or even other collections. You can either define collections in the SCOPE window or type the commands in the Tcl script window.

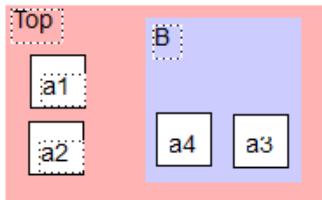
- [Comparison of Methods for Defining Collections, on page 147](#)
- [Creating and Using SCOPE Collections, on page 148](#)
- [Creating Collections using Tcl Commands, on page 151](#)
- [Viewing and Manipulating Collections with Tcl Commands, on page 154](#)

## Comparison of Methods for Defining Collections

You can enter the find and expand Tcl commands that are used to define collections in either the Tcl script window or in the SCOPE window. It is recommended that you use the SCOPE interface for the reasons outlined below:

	SCOPE Window	Tcl Window
Database used	Top level; includes all objects. See the example below.	Current Analyst view, which might be a lower-level view. If the current view is the Technology view after mapping, objects might be renamed, replicated, or removed.
Persistence	Collection saved in project file.	Collection only valid for the current session; you must redefine it the next time you open the project.
Constraints	Can apply to collection.	Cannot apply to collection.

In the design shown below, if you push down into B, and then type `find - hier a*` in the Tcl window, the command finds a3 and a4. However if you cut and paste the same command into the SCOPE Collections tab, your results would include a1, a2, a3, and a4, because the SCOPE interface uses the top-level database and searches the entire hierarchy.



## Creating and Using SCOPE Collections

A collection is a defined group of objects. Grouping objects let you operate on all the objects in the collection at the same time. A collection can consist of a single object, multiple objects, or even other collections. The following procedure shows you how to define collections in the SCOPE window. The SCOPE method is preferred over typing the commands in the Tcl window [Creating Collections using Tcl Commands, on page 151](#)) for the reasons described in [Comparison of Methods for Defining Collections, on page 147](#).

1. Define a collection by doing the following:
  - Open the SCOPE window and click the Collections tab.
  - In the Name column, type a name for the collection.

	Enabled	Collection Name	Command	Command Arguments	Comment
1	<input checked="" type="checkbox"/>	find_all	find	-hier -inst {*usbSlaveControl.u_endpMux.*}	
2	<input checked="" type="checkbox"/>	find_reg	find	-hier -seq {*usbSlaveControl.u_endpMux.*}	
3	<input checked="" type="checkbox"/>	find_comb	find	-hier -inst {*usbSlaveControl.u_endpMux.*} -filter @is_combination	
4					
5					
6					

Collections

- In the *Command* column, enter the command. See the *Command Reference* for complete syntax details. Additional information about specifying search patterns is described in [Specifying Search Patterns for Tcl find, on page 138](#) and [Specifying Search Patterns for Tcl find, on page 138](#).

You can also paste in a command. If you cut and paste a Tcl Find command from the Tcl window into the SCOPE Collections tab, remember that the SCOPE interface works on the top-level database, while the `find` command in the Tcl window works on the current level displayed in the Analyst view.

Objects in a collection do not have to be of the same type. The collections shown in the preceding figure do the following:

Collection	Finds...
find_all	All components in the module endpMux
find_reg	All registers in the module endpMux
find_comb	All combinational components under endpMux

The collections you define appear in the SCOPE pull-down object lists, so you can use them to define constraints.

- You can crossprobe the objects selected by the `find` and `expand` commands by right-clicking and choosing `Select in Analyst` column. The schematic views highlight the objects located by these commands. For other viewing operations, see [Viewing and Manipulating Collections with Tcl Commands, on page 154](#).

**Note:** Using collections with Tcl control constructs (such as if, for, foreach, and while) can produce unexpected synthesis results. Avoid defining constraints for collections with control constructs, especially since the constraint checker does not recognize these built-in Tcl commands.

- To create a collection that is made up of other collections, do this:

- Define the collections as described in the previous step. These collections must be defined before you can concatenate them or add them together in a new collection.
- To concatenate collections or add to collections, type a name for the new collection in the Name column. Type the appropriate operator command like c\_union or c\_diff in the Command column. See [Creating Collections using Tcl Commands, on page 151](#) for a list of available commands and the *Command Reference* for their syntax.

The software saves the collection information in the constraint file for the project.

- To apply constraints to a collection do the following:

- Define a collection as described in the previous steps.
- Go to the appropriate SCOPE tab and specify the collection name where you would normally specify the object name. Collections defined in the SCOPE interface are available from the pull-down object lists. The following figure shows the collections defined in step 1 available for setting a false path constraint.

	Enabled	Delay Type	From	To	Through	Start/End	Cycles	Max Delay(ns)	Comment
1	<input checked="" type="checkbox"/>	False	special_regs.status[7:0]						
2			i:decode.opcode_goto						
3			i:decode.opcode_call						
4			i:decode.opcode_retlw						
5			i:decode.skip						
6			i:decode.decodecs[13:0]						
			i:prgmctr..r_0 [10:0]						
			i:prgmctr..r_1 [10:0]						
			i:prgmctr..r_2 [10:0]						
			i:prgmctr..r_3 [10:0]						
			i:prgmctr..r_4 [10:0]						

- Specify the rest of the constraint as usual. The software applies the constraint to all the objects in the collection.

## Creating Collections using Tcl Commands

This section describes how to use the Tcl collection commands at the command line or in a script instead of entering them in the SCOPE window ([Creating and Using SCOPE Collections, on page 148](#)). There are differences in operation depending on where the collection commands are entered, and it is recommended that you use the SCOPE window, for the reasons described in [Comparison of Methods for Defining Collections, on page 147](#).

For details of the syntax for the commands described here, refer to [Collection Commands, on page 137](#) in the *Reference Manual*.

1. To create a collection using a Tcl command line command, name it with the `set` command and assign it to a variable.

A collection can consist of individual objects, Tcl lists (which can consist of a single element), or other collections. You can embed the Tcl `find` and `expand` commands in the `set` command to locate objects for the collection (see [Using the Tcl Find Command to Define Collections, on page 143](#) and [Specifying Search Patterns for Tcl find, on page 138](#)). The following example creates a collection called `my_collection` which consists of all the modules (views) found by the embedded `find` command:

```
set my_collection [find -view {*}]
```

2. To create collections derived from other collections, do the following:
  - Define a new variable for the collection.
  - Create the collection with one of the operator commands from this table:

To...	Use this command...
Add objects to a collection	<code>c_union</code> . See <a href="#">Examples: c_union Command, on page 152</a>
Concatenate collections	<code>c_union</code> . See <a href="#">Examples: c_union Command, on page 152</a> .
Isolate differences between collections	<code>c_diff</code> . See <a href="#">Examples: c_diff Command, on page 153</a> .
Find common objects between collections	<code>c_intersect</code> . See <a href="#">Examples: c_intersect Command, on page 153</a> .
Find objects that belong to just one collection	<code>c_symdiff</code> . See <a href="#">Examples: c_symdiff Command, on page 153</a> .

3. If your Tcl collection includes instances that use special characters, make sure to use extra curly braces or use a backslash to escape the special character.

Curly Braces { }	define_scope_collection GRP_EVENT_PIPE2 {find -seq {EventMux[2].event_inst?_sync[*]} -hier} define_scope_collection mytn {find -inst {i:count1.co[*]}}
------------------	--

Backslash Escape Character (\)	define_scope_collection mytn {find -inst i:count1.co\[*]}
-----------------------------------	---

---

Once you have created a collection, you can do various operations on the objects in the collection (see [Viewing and Manipulating Collections with Tcl Commands, on page 154](#)), but you cannot apply constraints to the collection.

## Examples: c\_union Command

This example adds the reg3 instance to collection1, which contains reg1 and reg2 and names the new collection sumCollection.

```
set sumCollection [c_union $collection1 {i:reg3}]  
c_list $sumCollection  
{"i:reg1" "i:reg2" "i:reg3"}
```

If you added reg2 and reg3 with the c\_union command, the command removes the redundant instances (reg2) so that the new collection would still consist of reg1, reg2, and reg3.

This example concatenates collection1 and collection2 and names the new collection combined\_collection:

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

## Examples: c\_diff Command

This example compares a list to a collection (collection1) and creates a new collection called subCollection from the list of differences:

```
set collection1 {i:reg1 i:reg2}
set subCollection [c_diff $collection1 {i:reg1}]
c_print $subCollection
    "i:reg2"
```

You can also use the command to compare two collections:

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

## Examples: c\_intersect Command

This example compares a list to a collection (collection1) and creates a new collection called interCollection from the objects that are common:

```
set collection1 {i:reg1 i:reg2}
set interCollection [c_intersect $collection1 {i:reg1 i:reg3}]
c_print $interCollection
    "i:reg1"
```

You can also use the command to compare two collections:

```
set common_collection [c_intersect $collection1 $collection2]
```

## Examples: c\_symmdiff Command

This example compares a list to a collection (collection1) and creates a new collection called diffCollection from the objects that are different. In this case, reg1 is excluded from the new collection because it is in the list and collection1.

```
set collection1 {i:reg1 i:reg2}
set diffCollection [c_symmdiff $collection1 {i:reg1 i:reg3}]
c_list $diffCollection
    {"i:reg2" "i:reg3"}
```

You can also use the command to compare two collections:

```
set symmdiff_collection [c_symmdiff $collection1 $collection2]
```

## Examples: Names with Special Characters

Your instance names might include special characters, as for example when your HDL code uses a generate statement. If your instance names have special characters, do the following:

Make sure that you include extra curly braces {}, as shown below:

```
define_scope_collection GRP_EVENT_PIPE2 {find -seq  
    {EventMux\[2\].event_inst?_sync[*]} -hier}  
define_scope_collection mytn {find -inst {i:count1.co[*]}}
```

Alternatively, use a backslash to escape the special character:

```
define_scope_collection mytn {find -inst i:count1.co\[*\\\]}
```

## Viewing and Manipulating Collections with Tcl Commands

The following section describes various operations you can do on the collections you defined. For full details of the syntax, see [Collections, on page 154](#) in the *Reference Manual*.

1. To view the objects in a collection, use one of the methods described in subsequent steps:

- Select the collection in an HDL Analyst view (step 2).
- Print the collection without carriage returns or properties (step 3).
- Print the collection in columns (step 4).
- Print the collection in columns with properties (step 5).

2. To select the collection in an HDL Analyst view, type `select <collection>`.

For example, `select $result` highlights all the objects in the `$result` collection.

3. To print a simple list of the objects in the collection, uses the `c_list` command, which prints a list like the following:

```
{i:EP0RxFifo.u_fifo.dataOut[0]} {i:EP0RxFifo.u_fifo.dataOut[1]}  
{i:EP0RxFifo.u_fifo.dataOut[2]} ...
```

The `c_list` command prints the collection without carriage returns or properties. Use this command when you want to perform subsequent Tcl commands on the list. See [Example: c\\_list Command, on page 156](#).

4. To print a list of the collection objects in column format, use the `c_print` command. For example, `c_print $result` prints the objects like this:

```
{i:EP0Rx_fifo.u_fifo.dataOut[0]}
{i:EP0Rx_fifo.u_fifo.dataOut[1]}
{i:EP0Rx_fifo.u_fifo.dataOut[2]}
{i:EP0Rx_fifo.u_fifo.dataOut[3]}
{i:EP0Rx_fifo.u_fifo.dataOut[4]}
{i:EP0Rx_fifo.u_fifo.dataOut[5]}
```

5. To print a list of the collection objects and their properties in column format, use the `c_print` command as follows:

- Annotate the design with a full list of properties by selecting Project->Implementation Options, going to the Device tab, and enabling Annotated Properties for Analyst. Synthesize the design. If you do not enable the annotation option, properties like clock pins will not be annotated as properties.
- Check the properties available by right-clicking on the object in the HDL Analyst view and selecting Properties from the popup menu. You see a window with a list of the properties that can be reported.
- In the Tcl window, type the `c_print` command with the `-prop` option. For example, typing `c_print -prop slack -prop view -prop clock $result` lists the objects in the `$result` collection, and their slack, view and clock properties.

Object Name	slack	view	clock
{i:EP0Rx_fifo.u_fifo.dataOut[0]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[1]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[2]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[3]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[4]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[5]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[6]}	0.3223	"FDE"	clk
{i:EP0Rx_fifo.u_fifo.dataOut[7]}	0.3223	"FDE"	clk
{i:EP0Tx_fifo.u_fifo.dataOut[0]}	0.1114	"FDE"	clk
{i:EP0Tx_fifo.u_fifo.dataOut[1]}	0.1114	"FDE"	clk

- To print out the results to a file, use the `c_print` command with the `-file` option. For example, `c_print -prop slack -prop view -prop clock $result -file results.txt` writes out the objects and properties listed above to a file called `results.txt`. When you open this file, you see the information in a spreadsheet format.

6. You can do a number of operations on a collection, as listed in the following table. For details of the syntax, see [Collections, on page 154](#) in the *Reference Manual*.

To...	Do this...
Copy a collection	<p>Create a new variable for the copy and copy the original collection to it with the <code>set</code> command. When you make changes to the original, it does not affect the copy, and vice versa.</p> <p style="text-align: center;"><code>set my_collection_copy \$my_collection</code></p>
List the objects in a collection	<p>Use the <code>c_print</code> command to view the objects in a collection, and optionally their properties, in column format:</p> <pre>"v:top" "v:block_a" "v:block_b"</pre> <p>Alternatively, you can use the <code>-print</code> option to an operation command to list the objects.</p>
Generate a Tcl list of the objects in a collection	<p>Use the <code>c_list</code> command to view a collection or to convert a collection into a Tcl list. You can manipulate a Tcl list with standard Tcl commands. In addition, the Tcl collection commands work on Tcl lists.</p> <p>This is an example of <code>c_list</code> results:</p> <pre>{"v:top" "v:block_a" "v:block_b"}</pre> <p>Alternatively, you can use the <code>-print</code> option to an operation command to list the objects.</p>

## Example: `c_list` Command

The following provides a practical example of how to use the `c_list` command. This example first finds all the CE pins with a negative slack that is less than 0.5 ns and groups them in a collection:

```
set get_components_list [c_list [find -hier -pin {*.CE} -filter
@slack < {0.5}]]
```

The `c_list` command returns a list:

```
{t:EP0RxFifo.u_fifo.dataOut[0].CE}
{t:EP0RxFifo.u_fifo.dataOut[1].CE}
{t:EP0RxFifo.u_fifo.dataOut[2].CE} ..
```

You can use the list to find the terminal (pin) owner:

```
proc terminal_to_owner_instance {terminal_name terminal_type} {  
    regsub -all $terminal_type\$ $terminal_name {} suffix  
    regsub -all {^t:} $suffix {i:} prefix  
    return $prefix  
}  
  
foreach get_component $get_components_list {  
    append owner [terminal_to_owner_instance $get_component {.CE}]  
"  
"  
}  
  
puts "terminal owner is $owner"
```

This returns the following, which shows that the terminal (pin) has been converted to the owning instance:

```
terminal owner is i:EP0Rx_fifo.u_fifo.dataOut[0]  
i:EP0Rx_fifo.u_fifo.dataOut[1] i:EP0Rx_fifo.u_fifo.dataOut[2]
```



## CHAPTER 6

# Synthesizing and Analyzing the Results

---

This chapter describes how to run synthesis, and how to analyze the log file generated after synthesis. See the following:

- [Synthesizing Your Design, on page 160](#)
- [Checking Log File Results, on page 166](#)
- [Handling Messages, on page 182](#)

# Synthesizing Your Design

Once you have set your constraints, options, and attributes, running synthesis is a simple one-click operation. See the following:

- [Running Logic Synthesis](#), on page 160
- [Using Up-to-date Checking for Job Management](#), on page 161

## Running Logic Synthesis

When you run logic synthesis, the tool compiles the design and then maps it to the technology target you selected.

1. If you want to compile your design without mapping it, select Run-> Compile Only or press F7.

A compiled design has the RTL mapping, and you can view the RTL view. You might want to just compile the design when you are not ready to synthesize the design, but when you need to use a tool that requires a compiled design, like the SCOPE interface.

2. To synthesize the logic, set all the options and attributes you want, and then click Run.

## Using Up-to-date Checking for Job Management

Synthesis is becoming more complex and consists of running many jobs. Often, part or all of the job flow is already up-to-date and rerunning the job may not be necessary. For large designs that may take hours to run, up-to-date checking can reduce the time for rerunning jobs.

Up-to-date checking includes the following:

- The GUI launches mapper modules (pre-mapping and technology mapping) and saves the intermediate netlists and log files in the synwork and synlog folders, respectively.
- After each individual module run completes, the GUI optionally copies the contents of these intermediate log files from the synlog folder and adds them to the Project log file (`rev_1/projectName.srr`). To set this option, see [Limitations and Risks, on page 163](#).
- If you re-synthesize the design and there are no changes to the inputs (HDL, constraints, and Project options):
  - The GUI does not rerun pre-mapping and technology mapping and no new netlist files are created.
  - In the HTML log file, the GUI adds a link that points to the existing pre-mapping and mapping log files from the previous run. Double-click on this link (@L: indicates the link) to open the new text file window.

If you open the text log file, the link is a relative path to the implementation folder for the pre-mapping and mapping log files from the previous run.

Also, the GUI adds a note that indicates mapping will not be re-run and to use the Run->Resynthesize All option in the Project view to force synthesis to be run again.

```
#####
# Synopsys Pre-mapping Report
#N: : | premapping output is up to date. No run necessary.
# To force a re-synthesis, select [Resynthesize All] in menu [Run].
# session mep610dev, Build
# 11 Rights Reserved
# Log file from previous run:
@L:top_premapping.srx
#####
@W:"C:\syn_hier\fixed\gated_clocks\myreq.v":13:14:18:22|Net U_req
@W:"C:\syn_hier\fixed\gated_clocks\top.v":17:14:17:24|Net U_gt_re
#
Finished Pre Mapping Phase. (Time elapsed Oh:00m:00s; Memory used
@N: BN225 |Writing default property annotation file C:\syn_hier\f
Pre Mapping successful!
Process took Oh:00m:01s realtime, Oh:00m:01s cputime
# Mon Jan 24 18:39:06 2011
#####

```

As the job is running, you can click in the job status field of the Project view to bring up the Job Status display. When you rerun synthesis, the job status identifies which modules (pre-mapping or mapping) are up-to-date.

Job Status for Synthesis Run

Job Name	State	Run Time	Job Command
(test)aged_clocks (synthesis)	Done	00:00:09	
└ Premap Flow (pre_map_flow)	Done	00:00:05	
└ Compile Flow (compile_flow)	Done	00:00:02	
└ HDL Compile (compiler)	Done	00:00:02	c_ver.exe - C:\builds\syn201103_114R\bin\c_ver.exe
└ Pre-mapping (premapping)	Done	00:00:04	m_xilinx.exe - C:\builds\syn201103_114R\bin\m_xilinx.exe
└ Mapping Flow (mapping_flow)	Done	00:00:02	
└ Mapping (virtex6 Mapper)	Done	00:00:02	m_xilinx.exe - C:\builds\syn201103_114R\bin\m_xilinx.exe

Cancel Job    Close

Job Status for Re-synthesis Run

Job Name	State	Run Time	Job Command
(test)aged_clocks (synthesis)	Done	00:00:06	
└ Premap Flow (pre_map_flow)	Done	00:00:04	
└ Compile Flow (compile_flow)	Done	00:00:02	
└ HDL Compile (compiler)	Done	00:00:02	c_ver.exe - C:\builds\syn201103_114R\bin\c_ver.exe
└ Pre-mapping (premapping)	Done (up-to-date)	00:00:02	m_xilinx.exe - C:\builds\syn201103_114R\bin\m_xilinx.exe
└ Mapping Flow (mapping_flow)	Done	00:00:02	
└ Mapping (virtex6 Mapper)	Done (up-to-date)	00:00:02	m_xilinx.exe - C:\builds\syn201103_114R\bin\m_xilinx.exe

Cancel Job    Close

See also:

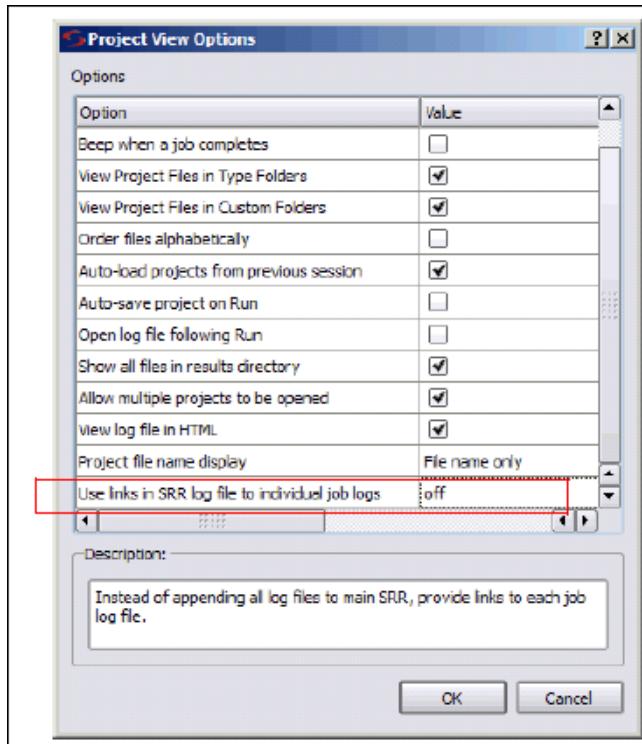
- [Copy Individual Job Logs to the SRR Log File](#)
- [Limitations and Risks](#)

## Copy Individual Job Logs to the SRR Log File

By default, up-to-date checking uses links in the log file (`srr`) to individual job logs. To change this option so that individual job logs are always appended to the main log file (`srr`), do the following:

1. Select Options->Project View Options from the Project menu.
2. On the Project View Options dialog box, scroll down to the Use links in SRR log file to individual job logs option.

3. Use the pull-down menu, and select off.



## Limitations and Risks

Up-to-date checking limitations and risks include the following:

- Compiler up-to-date checks are done internally by the compiler and with no changes to the compiler reporting structure.
- GUI up-to-date checks use timestamp information of its input files to decide when mapping is rerun. Be aware that:
  - The GUI uses netlist files (`srs` and `srd`) from the `synwork` folder for timestamp checks. If you delete an `srs` file from the implementation folder, this does not trigger compiler or mapper reruns. You must delete netlist files from the `synwork` folder instead.

- The copy command behaves differently on Windows and Linux. On Windows, the timestamp does not change if you copy a file from one directory to another. But on Linux (and MKS shell), the timestamp information gets changed.
- When running a design, the up-to-date checking feature automatically determines if the design needs to be re-synthesized. However, when you modify constraints in a Tcl file sourced within the constraints file, the software is not aware of these changes and does not force the design to be re-synthesized.

# Checking Log File Results

You can check the log file for information about the synthesis run. In addition, the user interface has a Tcl Script window, that echoes each command as it is run. The following describe different ways to check the results of your run:

- [Viewing and Working with the Log File](#), on page 166
- [Accessing Specific Reports Quickly](#), on page 170
- [Accessing Results Remotely](#), on page 173
- [Analyzing Results Using the Log File Reports](#), on page 176
- [Using the Watch Window](#), on page 177
- [Checking Resource Usage](#), on page 178
- [Querying Metrics for a Design](#), on page 180

## Viewing and Working with the Log File

The log file contains the most comprehensive results and information about a synthesis run. The default log file is in HTML format, but there is a text version available too.

For users who only want to check a few critical performance criteria, it is easier to use the Watch Window (see [Using the Watch Window](#), on page 177) instead of the log file. For details, read through the log file.

1. To open the log file, use one of these listed methods, according to the format you want:

HTML    

- Select View->Log File.
- Click the View Log button in the Project window.
- Double-click the *designName.htm* file in the Implementation Results view.

Text    Double-click the *designName.srr* file in the Implementation Results view.  
To set the text file version to open by default instead of the HTML version, select Options->Project View Options, and toggle off the View log file in HTML option.

---

The log file lists the compiled files, details of the synthesis run, and includes color-coded errors, warnings and notes, and a number of reports. For information about the reports, see [Analyzing Results Using the Log File Reports, on page 176](#).

```

10  # Start of Compile
11  #Wed Apr 25 08:59:31 2007
12
13  Synplicity Verilog Compiler, version 3.7, Build 196R, built Apr 16 2007
14  Copyright (C) 1994-2007, Synplicity Inc. All Rights Reserved
15
16  @I:::"C:/tools/syn880qt_056R/lib/proasic\proasic.v"
17  @I:::"C:\Designs\ramCtrl\alu.v"
18  @N: CG346 :"C:\Designs\ramCtrl\alu.v":78:27:78:35|Read full case directive
19
20
21
22
23
24
25
26
27
28
29

```

Main   Hierarchical Area

Compiler Report  
Mapper Report  
Timing Report  
Performance Summary  
Clock Relationships  
Interface Information  
Detailed Report for Clock\_eight  
    Starting Points with Wires  
    Ending Points with Wires  
    Worst Path Information  
  
Log File (HTML)  
Log File Links:  
rev\_4  
Hierarchical Area Report (C)  
rev\_4/par\_1

#Thu May 10 08:46:04 2007  
\$ Start of Compile  
#Thu May 10 08:46:04 2007  
Synplicity VHDL Compiler, version 3.7, Build 196R, built Apr 16 2007  
Copyright (C) 1994-2007, Synplicity Inc. All Rights Reserved  
  
@N:CD720 : std.vhd(123) | Setting time resolution to ns  
@I::: "C:\Designs\8-bit-vhdl\const\_pkg.vhd"  
@I::: "C:\Designs\8-bit-vhdl\ins\_rom.vhd"  
@I::: "C:\Designs\8-bit-vhdl\io.vhd"  
@I::: "C:\Designs\8-bit-vhdl\reg\_file.vhd"  
@I::: "C:\Designs\8-bit-vhdl\alu.vhd"  
@I::: "C:\Designs\8-bit-vhdl\data\_mux.vhd"  
@I::: "C:\Designs\8-bit-vhdl\ins\_decode.vhd"  
@I::: "C:\Designs\8-bit-vhdl\pc.vhd"  
@I::: "C:\Designs\8-bit-vhdl\spcl\_regs.vhd"  
@I::: "C:\Designs\8-bit-vhdl\eight\_bit\_uc.vhd"  
VHDL syntax check successful!  
  
Compiler output is up to date. No re-compile necessary

```

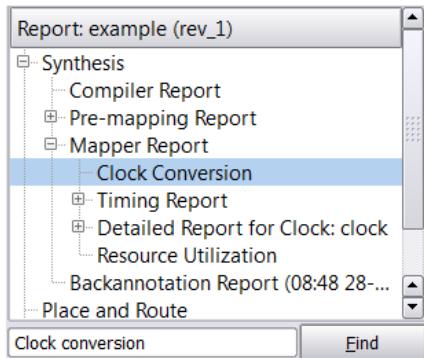
@N:CD630 : eight_bit_uc.vhd(7) | Synthesizing work.eight_bit_uc.structural
@N:CD233 : const_pkg.vhd(7) | Using sequential encoding for type alu_sel_type_a
@N:CD233 : const_pkg.vhd(6) | Using sequential encoding for type alu_sel_type
@N:CD233 : const_pkg.vhd(10) | Using sequential encoding for type aluop_type
@N:CD630 : ins_decode.vhd(7) | Synthesizing work.ins_decode.rtl
@N:CD233 : const_pkg.vhd(10) | Using sequential encoding for type aluop_type
@N:CD233 : const_pkg.vhd(7) | Using sequential encoding for type alu_sel_type_a

```

## 2. Navigate the log file to view specific pieces of information.

For quicker access to specific log information, use alternative access methods, described in [Accessing Specific Reports Quickly, on page 170](#) instead of the ones described here.

- Use the panel on the left of the HTML log file to navigate to the section you want. You can use the Find button and the search field at the bottom of this panel to search the headings.



- To search the body of the log file, use Control-f or the Edit->Find command. See [Viewing and Working with the Log File, on page 166](#) for details.
- To add bookmarks or for general information about working in an editing window, see [Editing HDL Source Files with the Built-in Text Editor, on page 37](#).

The areas of the log file that are most important are the warning messages and the timing report. The log file includes a timing report that lists the most critical paths. The synthesis product also lets you generate a report for a path between any two designated points, see [Generating Custom Timing Reports with STA, on page 344](#). The following table lists places in the log file you can use when searching for information.

To find...	Search for...
Notes	@N or look for blue text
Warnings and errors	@W and @E, or look for purple and red text respectively
Performance summary	Performance Summary
The beginning of the timing report	START TIMING REPORT

To find...	Search for...
Detailed information about slack times, constraints, arrival times, etc.	Interface Information
Resource usage	Resource Usage Report. See <a href="#">Checking Resource Usage, on page 178</a> .
Gated clock conversions	Gated clock report

3. Resolve any errors and check all warnings.

You must fix errors, because you cannot synthesize a design with errors. Check the warnings and make sure you understand them. See [Checking Results in the Message Viewer, on page 182](#) for information. Notes are informational and usually can be ignored. For details about crossprobing and fixing errors, see [Working with Downgradable Errors and Critical Warnings, on page 194](#), [Editing HDL Source Files with the Built-in Text Editor, on page 37](#), and [Crossprobing from the Text Editor Window, on page 306](#).

If you see Automatic dissolve at startup messages, you can usually ignore them. They indicate that the mapper has optimized away hierarchy because there were only a few instances at the lower level.

4. If you are trying to find and resolve warnings, you can bookmark them as shown in this procedure:
- Select Edit->Find or press Ctrl-f.
  - Type @W as the criteria on the Find form and click Mark All. The software inserts bookmarks at every line with a warning. You can now page through the file from bookmark to bookmark using the commands in the Edit menu or the icons in the Edit toolbar. For more information on using bookmarks, see [Editing HDL Source Files with the Built-in Text Editor, on page 37](#).
5. To crossprobe from the log file to the source code, click on the file name in the HTML log file or double-click on the warning text (not the ID code) in the ASCII text log file.

## Accessing Specific Reports Quickly

The log file contains all the results from the synthesis run, but you might want to hone in on specific information. Instead of browsing the log file to find the information you need, you can use the techniques described below:

1. To quickly view specific pieces of log information, go to the Project Status window and click the appropriate links to display the corresponding reports or specific parts of the log file.

Timing reports	Click Detailed Report or Timing Report View in the Timing Summary panel.
Log at different stages	Click Detailed Report in the Run Status panel.
Area reports	Click Detailed Report or Hierarchical Area Report in the Area Summary panel.
Optimizations	Click Detailed Report in the Optimizations Summary panel.

The Detailed Report links display parts of the log file, and the other links go to special view windows for different kinds of reports. See [The Project Results View, on page 25](#) for more information about different reports that can be accessed from the Project Results view.

The screenshot shows the Run Status panel in the Project Status window. It displays two jobs: "Compile Input (compiler)" and "Pre-mapping (premap)". The "Compile Input" job is complete with 25 errors, 0 warnings, and 0 notes. The "Pre-mapping" job is also complete with 4 errors, 1 warning, and 0 notes. The "Run Status" table has columns for Job Name, Status, Errors, Warnings, Notes, CPU Time, Real Time, Memory, and Date/Time.

Below the table, a tree view shows the "Report: tutorial (rev 3)" structure, with "Pre-mapping Report" selected. The log file content is displayed in the main pane:

```

Synopsys Generic Technology Pre-mapping, Version maprc, Build 4872, Built
Copyright (C) 1994-2016 Synopsys, Inc. All rights reserved. This Synopsys
Product Version K-2016.09C-SP1-beta

Mapper Startup Complete (Real Time elapsed 0h:00m:00s; CPU Time elapsed 0h:00m:00s)

Reading constraint file: C:\sw\tutorial\tutorial_3.fdc
Linked file: eight_bit uc_seck.rpt
Printing clock summary report in "C:\sw\tutorial\rev 3\eight bit uc_seck.rpt"
@N:MF219 : | Running in 32-bit mode.
@N:MF656 : | Clock conversion enabled

Design Input Complete (Real Time elapsed 0h:00m:00s; CPU Time elapsed 0h:00m:00s)

Mapper Initialization Complete (Real Time elapsed 0h:00m:00s; CPU Time elapsed 0h:00m:00s)

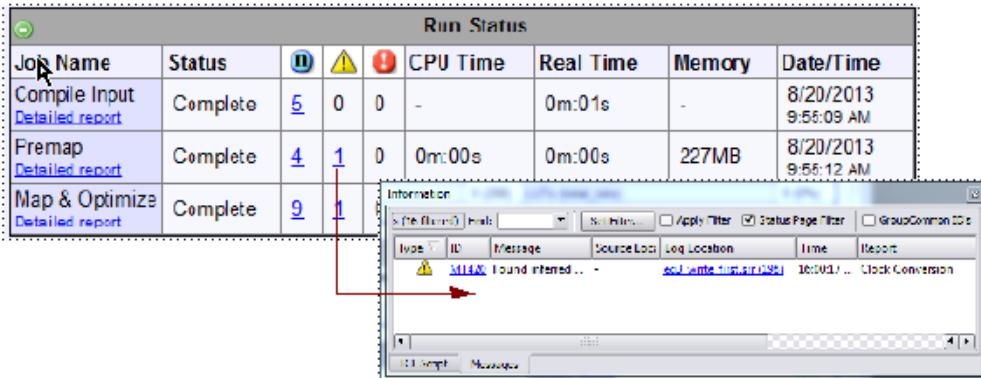
Adding property diff_term, value 1, to instance io_buff.un2l_ports

Start loading timing files (Real Time elapsed 0h:00m:00s; CPU Time elapsed 0h:00m:00s)

Finished loading timing files (Real Time elapsed 0h:00m:00s; CPU Time elapsed 0h:00m:00s)

```

2. To view timing information, use one of these methods:
    - Set important timing parameters to monitor in the Watch window, like slack and frequency. See [Using the Watch Window, on page 177](#) for details.
    - Click View Log in the Project view and navigate to the appropriate section in the log file.
  
  3. To view messages, use any of the following methods
    - From the Run Status panel in the Project Status window, click the link that lists the number of errors, warnings, or notes at different design stages. The Message window opens. Click the message ID to get more information about the error and how to fix it.
- This is the quickest method to narrow down the list of messages and access the one you want.



The numbers of notes, errors, and warnings reported in the Run Status panel might not match the numbers displayed in the Messages window if the design contains compile points. The numbers reported are for the top level.

- Click the Messages tab at the bottom of the Project view to open a window with a list of all the notes, errors and warnings. See [Checking Results in the Message Viewer, on page 182](#) for more information about using this window.

1 warning, 37 notes		Find:	Set Filter...	<input type="checkbox"/> Apply Filter	<input checked="" type="checkbox"/> GroupCommon	
Type	ID	Message	Source Location	Log Location	Time	Report
● [9]	CD233	Using sequential encoding for type aluop_type	const_pkg.vhd	spd_regs.srr	07:37:47 Wed May 09	Vhdl Compiler
● [9]	CD630	Synthesizing work.reg_file.first	-	spd_regs.srr	07:37:47 Wed May 09	Vhdl Compiler
● [8]	FX271	Instance "DECODE.ALUOP[3]" with 30 loads has been...	-	spd_regs.srr	07:37:47 Wed May 09	SPARTAN3 Mapper
● [4]	FX271	Instance "SPECIAL_REGS.INST[9]" with 19 loads has ...	spd_regs.vhd (44)	spd_regs.srr (194)	07:37:47 Wed May 09	SPARTAN3 Mapper
● [4]	FX271	Instance "SPECIAL_REGS.INST[11]" with 30 loads ha...	spd_regs.vhd (44)	spd_regs.srr (167)	07:37:47 Wed May 09	SPARTAN3 Mapper
● [4]	FX271	Instance "DECODE.ALUOP[1]" with 15 loads has been...	ins_decode.vhd (296)	spd_regs.srr (193)	07:37:47 Wed May 09	SPARTAN3 Mapper
● [4]	FX271	Instance "DECODE.ALUOP[3]" with 30 loads has been...	ins_decode.vhd (296)	spd_regs.srr (168)	07:37:47 Wed May 09	SPARTAN3 Mapper
● [4]	FX271	Instance "DECODE.ALUOP[0]" with 36 loads has been...	ins_decode.vhd (296)	spd_regs.srr (166)	07:37:47 Wed May 09	SPARTAN3 Mapper
● [4]	CD720	Setting time resolution to ns	std.vhd (123)	spd_regs.srr (16)	07:37:47 Wed May 09	Vhdl Compiler
● [4]	FX107	No read/write conflict check. Simulation mismatch pos...	req_file.vhd (23)	spd_regs.srr (112)	07:37:47 Wed May 09	Mapper Report
● [4]	CL134	Found RAM mem, depth=32, width=8	req_file.vhd (23)	spd_regs.srr (80)	07:37:47 Wed May 09	Vhdl Compiler
● [4]	CL201	Trying to extract state machine for register STACKLEV...	pc.vhd (34)	spd_regs.srr (82)	07:37:47 Wed May 09	Vhdl Compiler
● [4]	FX214	Generating ROM ROM.Data_1[11:0]	ins_rom.vhd (22)	spd_regs.srr (145)	07:37:47 Wed May 09	Mapper Report
● [4]	MT206	Autoconstraint Mode is ON	-	spd_regs.srr (108)	07:37:47 Wed May 09	Mapper Report
● [4]	MT197	Clock constraints cover only FF-to-FF paths associate...	-	spd_regs.srr (246)	07:37:47 Wed May 09	Timing Report

TCL Script    Messages

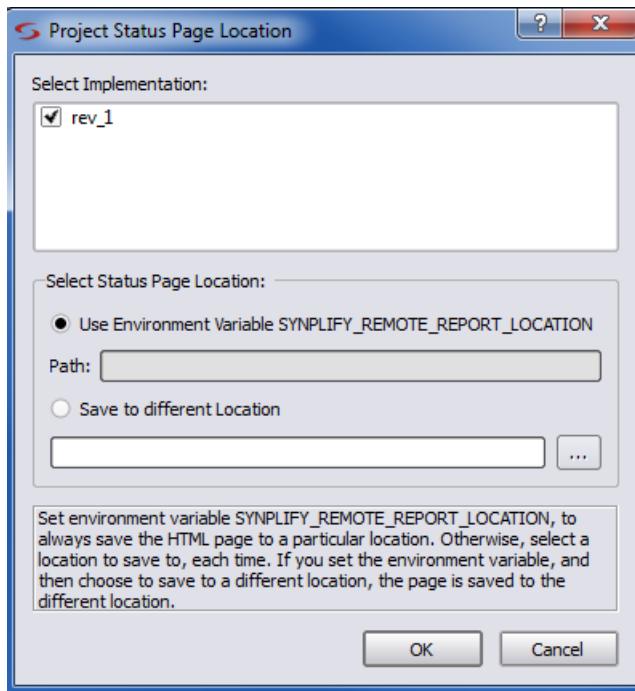
- Open the log file, locate the message, and click the message ID. The log file includes all the results from the run, so it could be harder to locate the message you want.

## Accessing Results Remotely

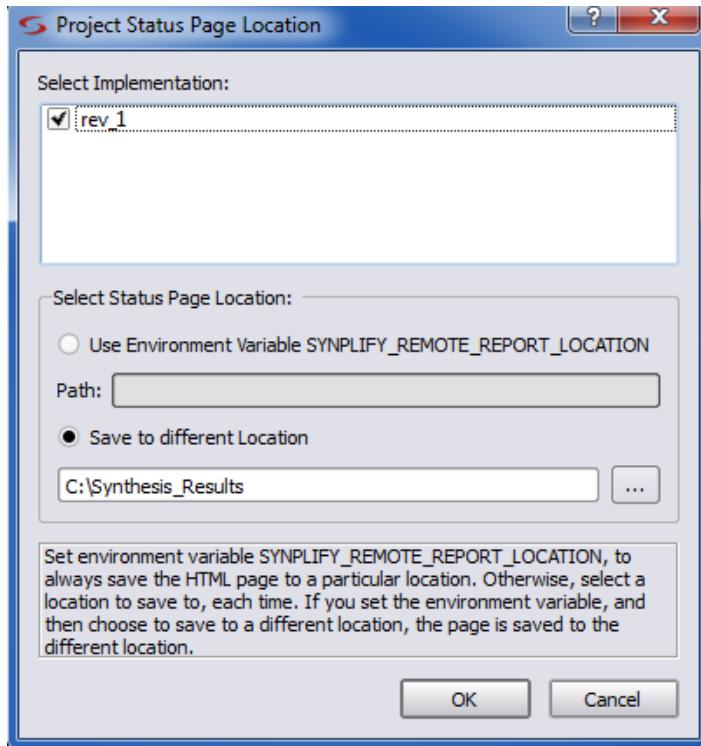
You can access the log file results remotely from various mobile devices. For example, you can use this feature to run synthesis for jobs with long run times and then check the results of the synthesis run later from anywhere. The Project Status report files can be accessed from any browser without bringing up the synthesis tool.

To access the log file remotely, do the following:

1. Select Options->Project Status Page Location from the Project menu and select the implementation for which you want the reports.



2. Set the location for storing the project status page, using either of these methods:
  - Enable Save to different location and specify a path for the location of the status page. This allows you to save the status reports in different locations.



- Use an environment variable by enabling Use Environment Variable `SYNPLIFY_REMOTE_REPORT_LOCATION`.

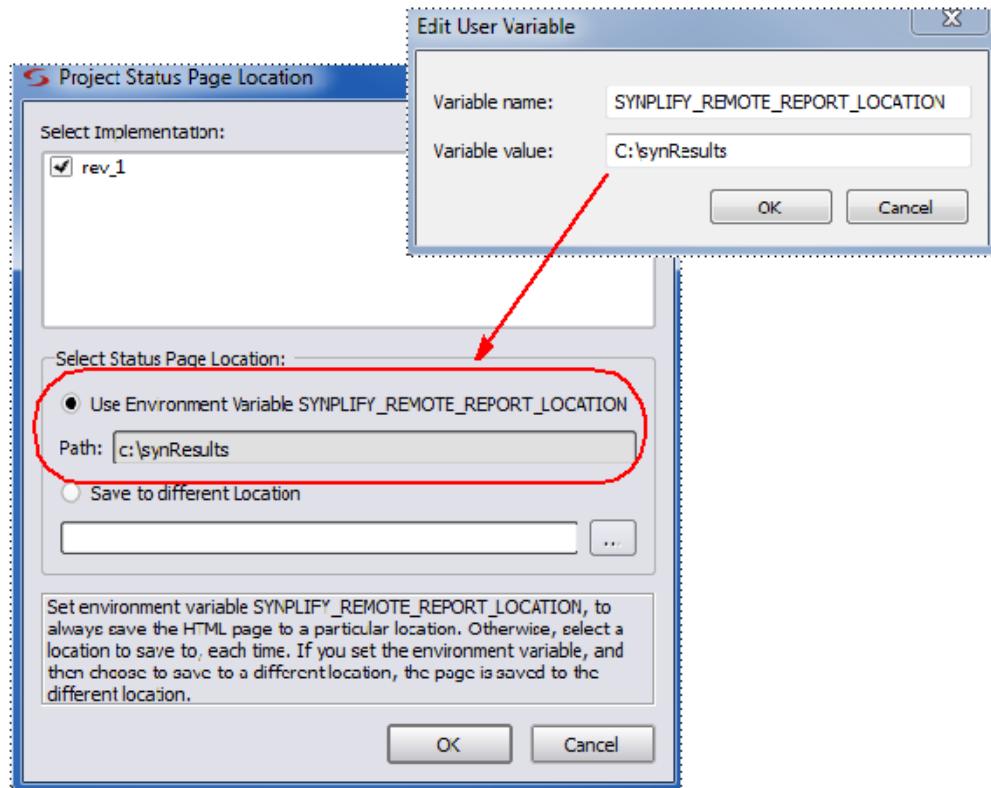
Windows Enable Use Environment Variable `SYNPLIFY_REMOTE_REPORT_LOCATION`.

Specify the variable name `SYNPLIFY_REMOTE_REPORT_LOCATION` and the location you want from the Control Panel on the Edit User Variable dialog box.

Linux Specify `setenv SYNPLIFY_REMOTE_REPORT_LOCATION pathLocation` in the `.cshrc` file.

Enable Use Environment Variable `SYNPLIFY_REMOTE_REPORT_LOCATION`.

If you use this option, you must restart the tool the first time, since the environment variable is not applied dynamically. This option always saves the status report to the location indicated by the variable.



3. Click OK.
4. Run synthesis.

The status reports are saved to the location you specified for your project. For example:

C:\synResults\tutorial\rev\_1

5. Access the location you set up from any browser on a mobile device (for example, a smart phone or tablet).
  - Access the location you set in the previous steps.
  - Open the *projectName/implementationName/index.html* file with any browser.

Your company may need to set up a location on its internal internet, where the status reports can be saved and later accessed with a URL address.

## Analyzing Results Using the Log File Reports

The log file contains technology-appropriate reports like timing reports, resource usage reports, and net buffering reports, in addition to any notes, errors, and warning messages.

1. To analyze timing results, do the following:
  - View the Timing Report (Performance Summary section of the log file) and check the slack times. See [Handling Negative Slack, on page 342](#) for details.
  - Check the detailed information for the critical paths, including the setup requirements at the end of the detailed critical path description. You can crossprobe and view the information graphically and determine how to improve the timing.
  - In the HTML log file, click the link to open up the HDL Analyst view for the path with the worst slack.

To generate timing information for a path between any two designated points, see [Generating Custom Timing Reports with STA, on page 344](#).

2. To check buffers, do the following:
  - Check the report by going to the Net Buffering Report section of the log file.
  - Check the number of buffers or registers added or replicated and determine whether this fits into your design optimization strategy.
3. To check logic resources, check the Resource Usage Report section at the end of the log file, as described in [Checking Resource Usage, on page 178](#).

## Using the Watch Window

The Watch window provides a more convenient viewing mechanism than the log file for quickly checking key performance criteria or comparing results from different runs. Its limitation is that it only displays certain criteria. If you need details, use the log file, as described in [Viewing and Working with the Log File, on page 166](#).

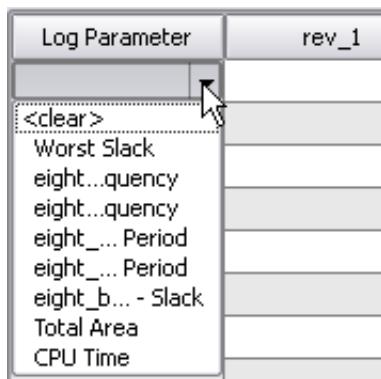
1. Open the Watch window, if needed, by checking View->Watch Window.

If you open an existing project, the Watch window shows the parameters set the last time you opened the window.

2. If you need a larger window, either resize the window or move the Watch Window as described below.
  - Hold down Ctrl or Shift, click the window, and move it to a position you want. This makes the Watch window an independent window, separate from the Project view.
  - To move the window to another position within the Project view, right-click in the window border and select **Float in Main Window**. Then move the window to the position you want, as described above.

See [Watch Window, on page 35](#) in the *Reference Manual* for information about the popup menu commands.

3. Select the log parameter you want to monitor by clicking on a line and selecting a parameter from the resulting popup menu.



The software automatically fills in the appropriate value from the last synthesis run. You can check the clock requested and estimated frequencies, the clock requested and estimated periods, the slack, and some resource usage criteria.

4. To compare the results of two or more synthesis runs, do the following:
  - If needed, resize or move the window as described above.
  - Click the right mouse button in the window and select Configure Watch from the popup.
  - Click Watch Selected Implementations and either check the implementations you want to compare or click Watch All Implementations. Click OK. The Watch window now shows a column for each implementation you selected.
  - In the Watch window, set the parameters you want to compare.

The software shows the values for the selected implementations side by side. For more information about multiple implementations, see [Tips for Optimization, on page 380](#).

Log Parameter	rev_1	rev_2	rev_3
Worst Slack	989.029	995.811	997.109
system clk_inferred_clock - Requested Frequency	1.0 MHz	1.0 MHz	1.0 MHz
system clk_inferred_clock - Estimated Frequency	91.1 MHz	238.7 MHz	345.9 MHz

## Checking Resource Usage

Each FPGA architecture has a certain number of dedicated FPGA resources. Use the Resource Usage section of the log file to check whether you are exceeding the available resources.

1. Go to the Resource Usage report at the end of the log file (srr).
2. Check the number and types of components used to determine if you have used too much of your resources.

The following is an example:

```
Resource Usage Report for top

Mapping to part: gw2a55pbga484-6
Cell usage:

ALU          84 uses
DFF          2 uses
DFFC         80 uses
DFFCE        15 uses
DFFE         62 uses
DFFP         10 uses
DFFR         36 uses
DFFRE        17 uses
GSR          1 use
INV          11 uses
LUT2         123 uses
LUT3         212 uses
LUT4         336 uses

I/O ports: 54
I/O primitives: 53
IBUF          31 uses
IOBUF         8 uses
OBUF          14 uses

I/O Register bits:           0
Register bits not including I/Os: 222 (0%)

Mapping Summary:
Total LUTs: 671 (1%)
```

If your design is over-utilized, you can manage usage with resource-specific attributes like `syn_ramstyle`, `syn_dspstyle`, and so on. For hierarchical designs you can set limits with attributes like `syn_allowed_resources` or the `Allocate Timing and Resource Budgets` command.

## Querying Metrics for a Design

Keeping track of metrics is important for measuring and tuning the QoR of a design. Metrics include data from various steps in the design flow; the data is saved and can be retrieved anytime. The design metrics you can query include the number of LUTs, the runtime for each process step, the worst slack, the slack for specific clocks, and the number of unconverted gated clocks.

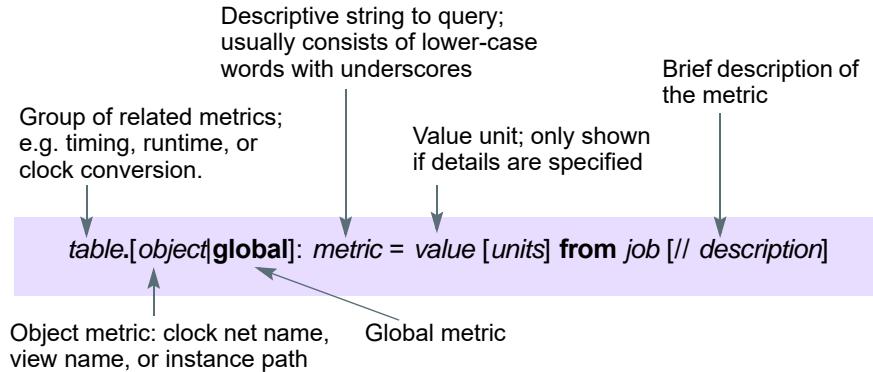
1. Start from the Tcl window and make sure you are located in the current implementation directory.
2. To find the names of the metrics available for the design, use one of the following command:

<b>Use the Command ...</b>	<b>To ...</b>
<code>dump_metrics</code>	Show metrics and values available for the current implementation of a design.
<code>query_available_metrics</code>	Show metrics that can be queried for the design. You should use this command primarily for scripting, since it returns a Tcl list.

3. Use the metric names with one of the following commands, according to the level of detail you want to see.

<b>Use the Command ...</b>	<b>To ...</b>
<code>query_metric</code>	Query specific QoR metrics. For example: <pre>% query_metric runtime.realpath -jobname compiler</pre> 3.154000
<code>query_metric_details</code>	Query information about a QoR metric. For example: <pre>% query_metric_details clock_conversion.clean_clock_pins -jobname fpga_mapper</pre> 271 {} {Number of clock pins driven by non-gated/non-generated clock trees}

Metrics can be global for the entire design, or specific to an object, such as a clock, module, or net. The command returns values in the default output format shown below:



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

The following is an example of reported metrics:

```
clock_conversion.global: instancesConverted = 0 from fpga_mapper
//Number of sequential instances converted

runtime.global: realtime = 4.160000 seconds from compiler
runtime.global: cputime = 3.073220 seconds from compiler
```

# Handling Messages

This section describes how to work with the error messages, notes, and warnings that result after a run. See the following for details:

- [Checking Results in the Message Viewer, on page 182](#)
- [Filtering Messages in the Message Viewer, on page 184](#)
- [Filtering Messages from the Command Line, on page 186](#)
- [Automating Message Filtering with a Tcl Script, on page 187](#)
- [Log File Message Controls, on page 189](#)
- [Working with Downgradable Errors and Critical Warnings, on page 194](#)

## Checking Results in the Message Viewer

The Tcl Script window includes a Message Viewer. By default, the Tcl window is in the lower left corner of the main window. This procedure shows you how to check results in the message viewer.

1. If you need a larger window, either resize the window or move the Tcl window. Click in the window border and move it to a position you want. You can float it outside the main window or move it to another position within the main window.
2. Click the Messages tab to open the message viewer.

The window lists the errors, warnings, and notes in a spreadsheet format. See [Message Viewer, on page 39](#) in the *Reference Manual* for a full description of the window.

The screenshot shows the Synplify Pro interface with the 'Messages' tab selected in the bottom navigation bar. The main window displays a table of messages with the following columns: Type, ID, Message, Source Location, Log Location, Time, and Report. There are 3 warnings and 22 notes listed. The 'Report' column contains links to various reports like 'Clock Conversion', 'Pre-mapping Repo', 'Compiler Report', and 'Clock Conversion'. The 'Source Location' and 'Log Location' columns show file paths such as 'alu.v (107)', 'alu.v (93)', and 'eight\_bit.sv'. The 'Time' column shows timestamps like '13:00:39...', '13:00:36...', '13:00:25...', and '13:00:39...'. The 'Find' and 'Set Filter...' buttons are visible at the top of the table.

Type	ID	Message	Source Location	Log Location	Time	Report
W	MT320	Found inferred clock aluclk3 with period 1000.0 ...	-	eight_bit.sv	13:00:39...	Clock Conversion
W	MT329	Found inferred clock aluclk3 which controls 10 ...	alu.v (107)	eight_bit.sv...	13:00:36...	Pre-mapping Repo
W	CL269	Static error detection not built	alu.v (93)	eight_bit.sv	13:00:25...	Compiler Report
W	BW103	The default time unit for the Synopsys Contral ...	-	eight_bit.sv...	13:00:39...	Clock Conversion
W	BW107	Synchronous Counter did not converge until 1000	-	eight_bit.sv	12:00:20...	Clock Conversion

3. To reduce the clutter in the window and make messages easier to find and understand, use the following techniques:
  - Use the color cues. For example, when you have multiple synthesis runs, messages that have not changed from the previous run are in black; new messages are in red.
  - Enable the Group Common IDs option in the upper right. This option groups all messages with the same ID and puts a plus symbol next to the ID. You can click the plus sign to expand grouped messages and see individual messages.

There are two types of message groups:

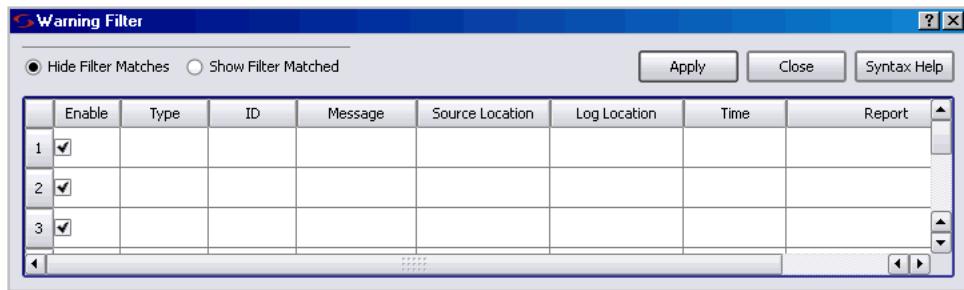
- The same warning or note ID appears in multiple source files indicated by a dash in the source files column.
  - Multiple warnings or notes in the same line of source code indicated by a bracketed number.
  - Sort the messages. To sort by a column header, click that column heading. For example, click Type to sort the messages by type. For example, you can use this to organize the messages and work through the warnings before you look at the notes.
  - To find a particular message, type text in the Find field. The tool finds the next occurrence. You can also click the F3 key to search forward, and the Shift-F3 key combination to search backwards.
4. To filter the messages, use the procedure described in [Filtering Messages in the Message Viewer, on page 184](#). Crossprobe errors from the message window:
    - If you need more information about how to handle a particular message, click the message ID in the ID column. This opens the documentation for that message.
    - To open the corresponding source code file, click the link in the Source Location column. Correct any errors and rerun synthesis.
    - To view the message in the context of the log file, click the link in the Log Location column.

## Filtering Messages in the Message Viewer

The Message viewer lists all the notes, warnings, and errors. The following procedure shows you how to filter out the unwanted messages from the display, instead of just sorting it as described in [Checking Results in the Message Viewer, on page 182](#). For the command line equivalent of this procedure, see [Filtering Messages from the Command Line, on page 186](#).

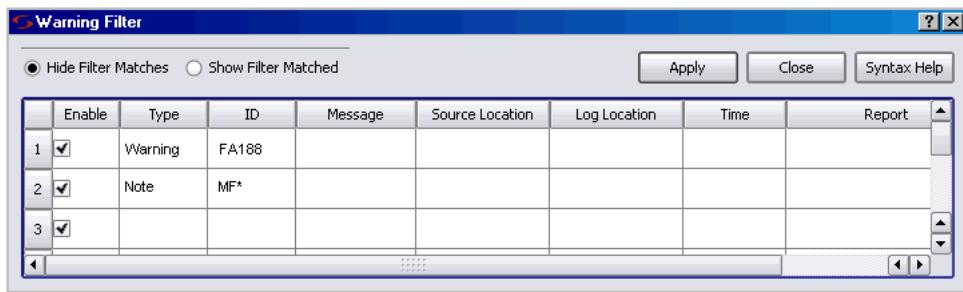
1. Open the message viewer by clicking the Messages tab in the Tcl window as previously described.
2. Click Filter in the message window.

The Warning Filter spreadsheet opens, where you can set up filtering expressions. Each line is one filter expression.



3. Set your display preferences.
  - To hide your filtered choices from the list of messages, click Hide Filter Matches in the Warning Filter window.
  - To display your filtered choices, click Show Filter Matches.
4. Set the filtering criteria.
  - Set the columns to reflect the criteria you want to filter. You can either select from the pull-down menus or type your criteria. If you have multiple synthesis runs, the pull-down menu might contain selections that are not relevant to your design.

The first line in the following example sets the criteria to show all warnings (Type column) with message ID FA188 (ID). The second set of criteria displays all notes that begin with MF.



- Use multiple fields and operators to refine filtering. You can use wildcards in the field, as in line 2 of the example. Wildcards are case-sensitive and space-sensitive. You can also use ! as a negative operator. For example, if you set the ID in line 2 to !MF\*, the message list would show all notes except those that begin with MF.
- Click **Apply** when you have finished setting the criteria. This automatically enables the **Apply Filter** button in the messages window, and the list of messages is updated to match the criteria.

The synthesis tool interprets the criteria on each line in the Warning Filter window as a set of AND operations (Warning and FA188), and the lines as a set of OR operations (Warning and FA188 or Note and MF\*).

- To close the Warning Filter window, click **Close**.

5. To save your message filters and reuse them, do the following:

- Save the project. The synthesis tool generates a Tcl file called *projectName.pfl* (Project Filter Log) in the same location as the main project file. The following is an example of the information in this file:

```
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"
```

- When you want to reuse the filters, source the *projectName.pfl* file.

You can also include this file in a synhooks Tcl script to automate your process.

## Filtering Messages from the Command Line

The following procedure shows you how to use Tcl commands to filter out unwanted messages. If you want to use the GUI, see [Filtering Messages in the Message Viewer, on page 184](#).

1. Type your filter expressions in the Tcl window using the `log_filter` command. For details of the syntax, see [log\\_filter, on page 60](#) in the *Command Reference Manual*.

For example, to hide all the notes and print only errors and warnings, type the following:

```
log_filter -enable
log_filter -hide_matches
log_filter -field type==Note
```

2. To save and reuse the filter commands, do the following:

- Type the `log_filter` commands in a Tcl file.
- Source the file when you want to reuse the filters you set up.

3. To print the results of the `log_filter` commands to a file, add the `log_report` command at the end of a list of `log_filter` commands.

```
log_report -print filteredMsg.txt
```

This command prints the results of the preceding `log_filter` commands to the specified text file, and puts the file in the same directory as the main project file. The file contains the filtered messages, for example:

```
@N MF138 Rom slaveControlSel_1 mapped in logic. Mapper Report
    wishbonebi.v (156) usbHostSlave.srr (819) 05:22:06 Mon Oct 18
@N(2) M0106 Found ROM, 'slaveControlSel_1', 15 words by 1 bits
    Mapper Report wishbonebi.v (156) usbHostSlave.srr (820)
    05:22:06 Mon Oct 18
@N M0106 Found ROM, 'slaveControlSel_1', 15 words by 1 bits Mapper
    Report wishbonebi.v (156) usbHostSlave.srr (820) 05:22:06 Mon
    Oct 18
@N MF138 Rom hostControlSel_1 mapped in logic. Mapper Report
    wishbonebi.v (156) usbHostSlave.srr (821) 05:22:06 Mon Oct 18
@N M0106 Found ROM, 'hostControlSel_1', 15 words by 1 bits Mapper
    Report wishbonebi.v (156) usbHostSlave.srr (822) 05:22:06 Mon
    Oct 18
@N Synthesizing module writeUSBWireData Compiler Report
    writeusbwiredata.v (59) usbHostSlave.srr (704) 05:22:06 Mon Oct 18
```

## Automating Message Filtering with a Tcl Script

The following example shows you how to use a `synhooks` Tcl script to automatically load a message filter file when a project opens and to send email with the messages after a run.

1. Create a message filter file like the following. (See [Filtering Messages in the Message Viewer, on page 184](#) or [Filtering Messages from the Command Line, on page 186](#) for details about creating this file.)

```
log_filter -clear
log_filter -hide_matches
log_filter -field report=="GW2A55 MAPPER"
log_filter -field type==NOTE
log_filter -field message=="Input *"
log_filter -field message=="Pruning *"
puts "DONE!"
```

2. Copy the `synhooks.tcl` file and set the environment variable as described in [Automating Flows with synhooks.tcl, on page 494](#).

3. Edit the synhooks.tcl file so that it reads like the following example. For syntax details, see *synhooks File Syntax*, on page 260 in the *Reference Manual*.

- The following loads the message filter file when the project is opened. Specify the name of the message filter file you created in step 1. Note that you must source the file.

```
proc syn_on_open_project {project_path} {
    set filter filterFilename
    puts "FILTER $filter IS BEING APPLIED"
    source d:/tcl/filters/$filterFilename
}
```

- Add the following to print messages to a file after synthesis is done:

```
proc syn_on_end_run {runName run_dir implName} {
    set warningFileName "messageFilename"

    if {$runName == "synthesis"} {
        puts "Mapper Done!"
        log_report -print $warningFileName
    set f [open [lindex $warningFileName] r]
    set msg ""
    while {[gets $f warningLine]>=0} {
        puts $warningLine
        append msg $warningLine\n
    }
    close $f
}
```

- Continue by specifying that the messages be sent in email. You can obtain the `smtp` email packages off the web.

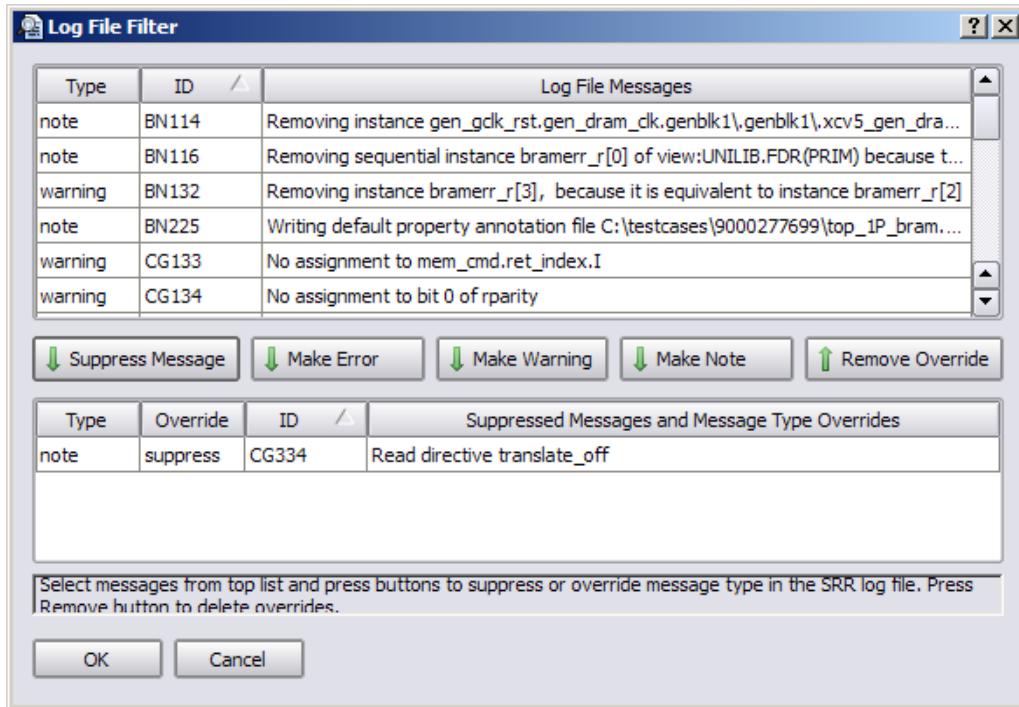
```
source "d:/tcl/smtp_setup.tcl"
proc send_simple_message {recipient email_server subject body} {
    set token [mime::initialize -canonical text/plain -string
               $body]
    mime::setheader $token Subject $subject
    smtp::sendmessage $token -recipients $recipient -servers
                           $email_server
    mime::finalize $token
}
puts "Sending email..."

send_simple_message {address1,address2}
    yourEmailServer subjectText> emailText
}
}
```

When the script runs, an email with all the warnings from the synthesis run is automatically sent to the specified email addresses.

## Log File Message Controls

The log file message control feature 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. This dialog box is displayed by selecting Set Filter from the Messages window and clicking Log File Filter from the GUI.



## Log File Filter Dialog Box

The Log File Filter dialog box is the primary control for changing a message priority or suppressing a message. When you initially open the dialog box, all of the messages from the log (srr) file for the active implementation are displayed in the upper section and the lower section is empty. To use the dialog box:

1. Select (highlight) the message to be promoted, demoted, or suppressed from the messages displayed in the upper section.
2. Select the Suppress Message, Make Error, Make Warning, or Make Note button to move the selected message from the upper section to the lower section. The selected message is repopulated in the lower section with the Override column reflecting the disposition of the message according to the button selected.

## Allowed Severity Changes

Allowed severity levels and preference settings for warning, note, and advisory messages are:

- Promote – warning to error, note to warning, note to error
- Demote – warning to note
- Suppress – suppress warning, suppress note, suppress advisory

---

**Note:** Normal error messages (messages generated by default) cannot be suppressed or changed to a lesser severity level.

---

When using the dialog box:

- Use the control and shift keys to select multiple messages.
- If an `srr` file is not present (for example, if you are starting a new project) the table will be empty. Run the design at least once to generate an `srr` file.
- Clicking the OK button saves the message status changes to the `projectName.pfl` file in the project directory.

## Message Reporting

The compiler and mapper must be rerun before the impact of the message status changes can be seen in the updated log file.

When a `projectName.pfl` input file is present at the start of the run, the message-status changes in the file are forwarded to the mapper and compiler which generate an updated log file. Depending on the changes specified:

- If an ID is promoted to an error, the mapper/compiler stops execution at the first occurrence of the message and prints the message in the `@E:msgID :messageText` format
- If an ID is promoted to a warning, the mapper/compiler prints the message in the `@W:msgID :messageText` format.

- If an ID is demoted to a note, the mapper/compiler prints the message in the @N:*msgID* :*messageText* format.
- If an ID is suppressed, the mapper/compiler excludes the message from the *srr* file.

---

**Note:** The online, error-message help documentation is unchanged by any message modification performed by the filtering mechanism. If a message is initially categorized as a warning in the synthesis tool, it continues to be reported as a warning in error-message help irrespective its promotion/demotion status.

---

## Updating the *projectName.pfl* file

The *projectName.pfl* file in the top-level project directory stores the user message filter settings from the Log File Filter dialog box for that project. This file can be edited with a text editor. The file entry syntax is:

```
message_override -suppress ID [ID ...] | -error ID [ID ...] | -warning ID [ID ...]
| -note ID [ID ...]
```

For example, to override the default message definition for note FX702 as a warning, enter:

```
message_override -warning FX702
```

You can also limit the number of occurrences for specified message IDs with the following syntax:

```
message_override [-limit value] [-count value]
```

For example, limit messages with IDs FX214 and FX271 to 100 each in each log file as follows:

```
message_override -limit {FX214 FX271} -count 100
```

Then, select the message filter file (.pfl) to be read for the project with the Read Message File option.

---

**Note:** After editing the pfl file, close and reopen the project to update the overrides.

---

## **messagefilter.txt File**

A messagefilter.txt file in the *implementation/syntmp* directory lists any changes made to message priority or suppression through the Log File Filter dialog box. This file, which is only generated when changes are made to the default status of a message, can be accessed outside of the GUI without consuming a license.

## Working with Downgradable Errors and Critical Warnings

You can temporarily change the classification for certain kinds of messages:

Downgradable errors (@DE)	Can be downgraded to warnings. This is a small set of <i>non-fatal</i> errors where you can temporarily postpone addressing the error and continue with the design flow and verification of other aspects of the design.
Critical warnings (@CW)	Can be upgraded to errors. This small set of warnings represent critical problems. Elevating them to errors ensures that they are recognized and dealt with, because the error status forces the tool to stop.

You can downgrade or upgrade these messages from the GUI or through a Tcl command.

### Downgrading or Upgrading Messages from the GUI

1. Open the log file.
2. Right-click within the log file and select Log File Message Filter from the pop-up menu to display the Log File Filter dialog box.
3. Highlight the DE or CW message from the list of messages displayed at the top of the dialog box. You can only downgrade messages with a DE prefix, or upgrade messages with a CE prefix.
4. Depending on the message type selected:
  - Click the Make Warning button to downgrade the error to a warning. This action moves the message from the top section of the message filter to the bottom section. The Override column in the bottom section displays the updated status for the message (Warning).
  - Click the Make Error button to upgrade the critical warning to an error. This action moves the message from the top section of the message filter to the bottom section. The Override column in the bottom section displays the updated status for the message (Error).
5. To see the changes reflected, click the Run button. You can verify that the message is now treated as:
  - A warning (the mapping operation continues past the initial point of the error condition)

- An error (the operation stops at the initial point of the error).
6. To revert the DE or CW message:
    - Click on one of the downgraded DE warning IDs in the report and select Log File Message Filter from the pop-up menu to display the Log File Filter dialog box. This reverts the warning message to an error.
    - Click on one of the upgraded CW warning IDs in the report and select Log File Message Filter from the pop-up menu to display the Log File Filter dialog box. This reverts the error message to a warning.
  7. From the dialog box, highlight the DE or CW message from the list of messages displayed at the top of the dialog box. Click either the:
    - Make Error button to return the message to its original error status.
    - Make Warning button to return the message to its original critical warning status.

Modifying the status of a message does not affect the message string. A message originally categorized as an error continues to be reported as an error regardless of its user-assigned status.

## Downgrading or Upgrading Messages with a Tcl Command

Type the `message_override` command in the Tcl script window to change the classification of DE and CW messages.

1. To downgrade DE errors or upgrade CW warnings, use the appropriate `message_override` command:

```
message_override -warning DEmessageID
message_override -error CWmessageID
```

Enter the message ID without the @ prefix, as shown above.

2. To see the changes reflected, rerun the state from the database where you changed the message classification.

After you have fixed the cause of the upgraded warning (critical warning) or completed the rest of the flow (downgradable error), you must change the message classification back to its original status.

3. To revert the messages back to their original status, use the appropriate `message_override` command:

**message\_override -error DEmessageID**  
**message\_override -warning CWmessageID**

4. Rerun again to confirm that the message status has reverted to the original.

## CHAPTER 7

# Analyzing with HDL Analyst

---

This chapter describes how to analyze logic in the HDL Analyst and FSM Viewer.

See the following for detailed procedures:

- [Working in the Schematic](#), on page 198
- [Exploring Design Hierarchy](#), on page 221
- [Finding Objects](#), on page 227
- [Crossprobing](#), on page 238
- [Analyzing With the HDL Analyst Tool](#), on page 245
- [Working in the Standard Schematic](#), on page 265
- [Exploring Design Hierarchy \(Standard\)](#), on page 280
- [Finding Objects \(Standard\)](#), on page 288
- [Crossprobing \(Standard\)](#), on page 303
- [Analyzing With the Standard HDL Analyst Tool](#), on page 311
- [Using the FSM Viewer \(Standard\)](#), on page 329

# Working in the Schematic

The HDL Analyst tool includes single-page schematics, which can help you graphically analyze and navigate your entire design easier. This section describes basic procedures you use in the schematics. These procedures include the following topics:

- Clone Schematic – See [Cloning Schematics](#), on page 201
- Instance Groups – See [Grouping Objects in the Schematic](#), on page 214
- Partial Dissolve – See [Grouping Objects in the Schematic](#), on page 214
- Net Based Filtering – See [Filtering Schematics](#), on page 249
- Unfilter – See [Filtering Schematics](#), on page 249
- Multi-threaded Find – See [Browsing to Find Objects in HDL Analyst Views](#), on page 227
- New Mouse Strokes with Cancel Display– See [Mouse Stroke Conventions](#), on page 201
- New Push View Tab – See [Cloning Schematics](#), on page 201
- Peek – See [Viewing Design Hierarchy and Context](#), on page 245
- Improved Bus Display and Handling – See [Dissolving and Partial Dissolving of Buses and Pins](#), on page 255

For information on specific tasks like analyzing critical paths, see the following sections:

- [Traversing Design Hierarchy with the Hierarchy Browser](#), on page 221
- [Exploring Object Hierarchy with Push/Pop Commands](#), on page 223
- [Crossprobing](#), on page 238
- [Analyzing With the HDL Analyst Tool](#), on page 245

## Opening the Views

The procedure for opening a view is the same at different design stages; the main difference is the content that is available at the different design database states.

1. Start at the database state you want.

RTL view	Start with a compiled design.
Technology view	Start with a mapped (synthesized) design.

2. To enable the new HDL Analyst tool, use one of the following methods:

- From the UI, select HDL Analyst->Use New HDL Analyst option
- From the UI: Select Options->Use New HDL Analyst option.

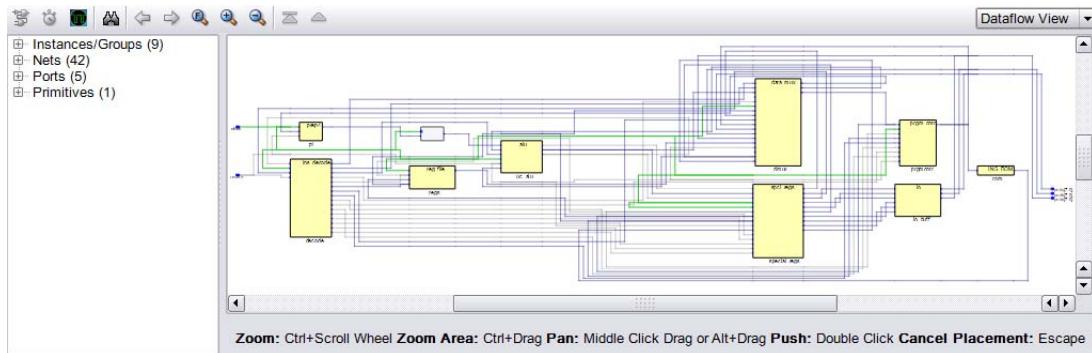
By default, this option is enabled.

3. Open the schematic using one of the following commands:

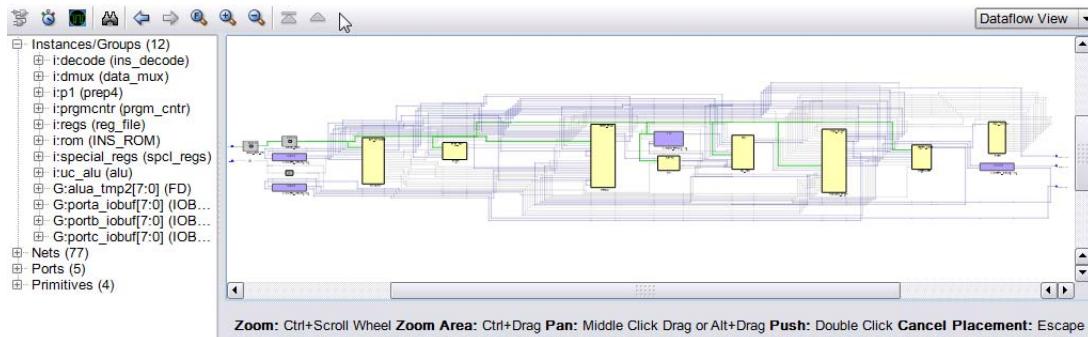
Hierarchical RTL or Technology view	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst-&gt;RTL-&gt;Hierarchical View.</li> <li>• Click the RTL View icon ()</li> <li>• Double-click the <code>srs</code> file in the Implementation Results view.</li> </ul> <p>To open a flattened RTL view, select HDL Analyst-&gt;RTL-&gt;Flattened View.</p>
Hierarchical Technology view	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst-&gt;Technology-&gt;Hierarchical View.</li> <li>• Click the Technology View icon ()</li> <li>• Double-click the <code>srm</code> file in the Implementation Results view.</li> </ul>
Flattened RTL or Technology view	Select HDL Analyst->RTL->Flattened View or HDL Analyst->Technology->Flattened View
Design Plan view	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst-&gt;RTL-&gt;Design Plan View.</li> <li>• Double-click the partitioned netlist (<code>srp</code>) file from the Implementation Results view.</li> </ul>

All schematic views have the schematic on the right and a pane on the left that contains a hierarchical list of the objects in the design. This pane is called the Hierarchy Browser.

### Compiled View



### Mapped View

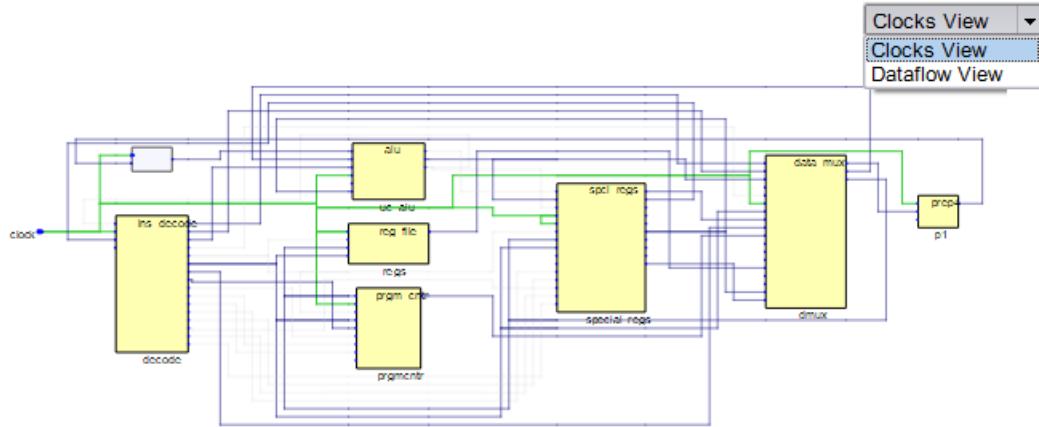


## Dataflow View

Both the compiled and mapped views have a Dataflow View. Use this view to display objects from a left to right datapath flow as shown above. You can display a Clock View as well.

## Clock View

To display all sequential elements connected to clock nets and debug the clocks in the design use the view selector. Select Clocks View from the drop-down menu in the upper right corner of the schematic view. Clock nets are displayed with the color green.



## Mouse Stroke Conventions

Use the mouse strokes to control navigation and the display, which are listed at the bottom of the schematic window. They include:

- Zoom – Ctrl-scroll wheel
- Zoom Area – Ctrl-drag
- Pan – Middle-click drag or Alt-drag
- Push – Double-click
- Pop – Double-click on an empty space
- Cancel Display – Press Esc  
(For large designs, you can cancel displaying the netlist but still use the netlist from the hierarchy browser, Tcl window, and Find dialog box.)

## Cloning Schematics

Most operations performed in any of the HDL Analyst views (Clock View or Dataflow View) are displayed in the current view. To create a new view of the netlist, use the clone commands.

1. To clone the current view displayed, right-click and select Clone Schematic from the drop-down menu. This view opens in a new window. You can open multiple clone views.

Tcl equivalent: `analyst clone_view`

To close a clone view: `analyst close_design designID`.

For example: `analyst close_design d:3`

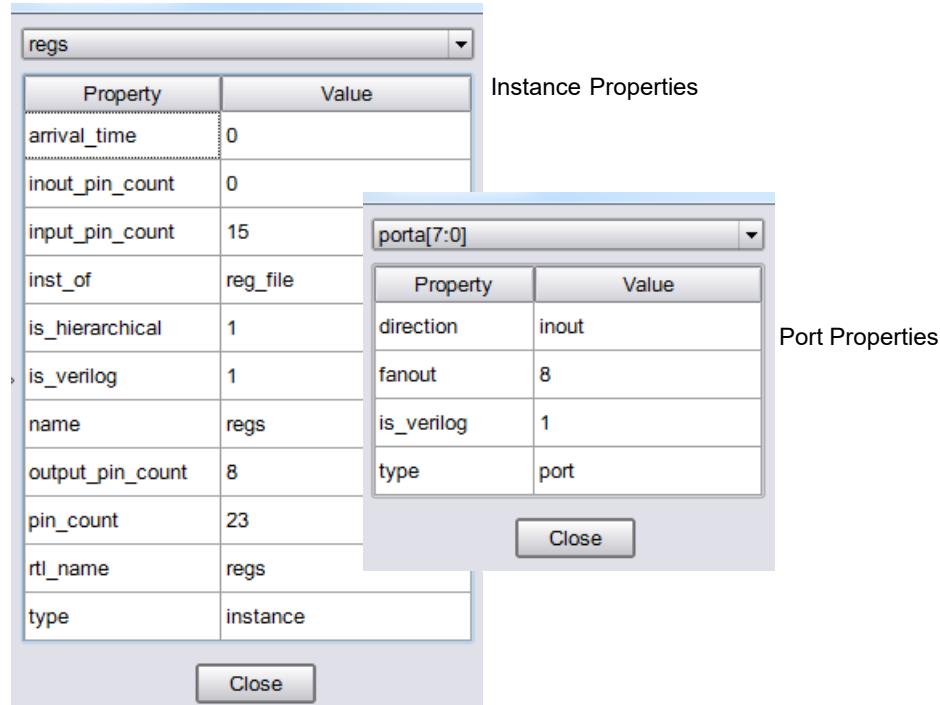
2. To push into an object and create a new view, select an object then right-click and Push in New Tab from the drop-down menu. For more information, see [Exploring Object Hierarchy with Push/Pop Commands, on page 223](#).
3. To filter objects and create a new view, select the objects then right-click and select Filter in a New Tab from the drop-down menu. For more information, see [Filtering Schematics, on page 249](#)

## Viewing Object Properties

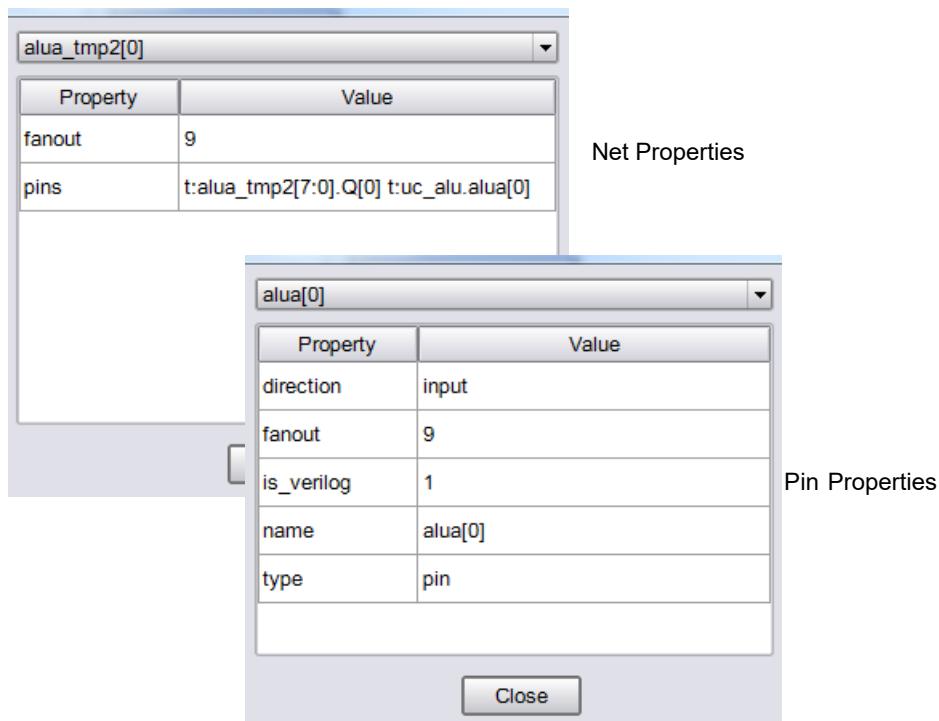
There are a few ways in which you can view the properties of objects.

1. To temporarily display the properties of a particular object, hold the cursor over the object.
2. Select the object, right-click, and select Properties. The properties and their values are displayed in a table.

For example, you can view the properties for instances and ports as shown below.



Similarly, you can view the properties for pins and nets.



3. You can copy any number of fields from the Properties dialog box and paste the properties to the Tcl window or a text file from within the tool.

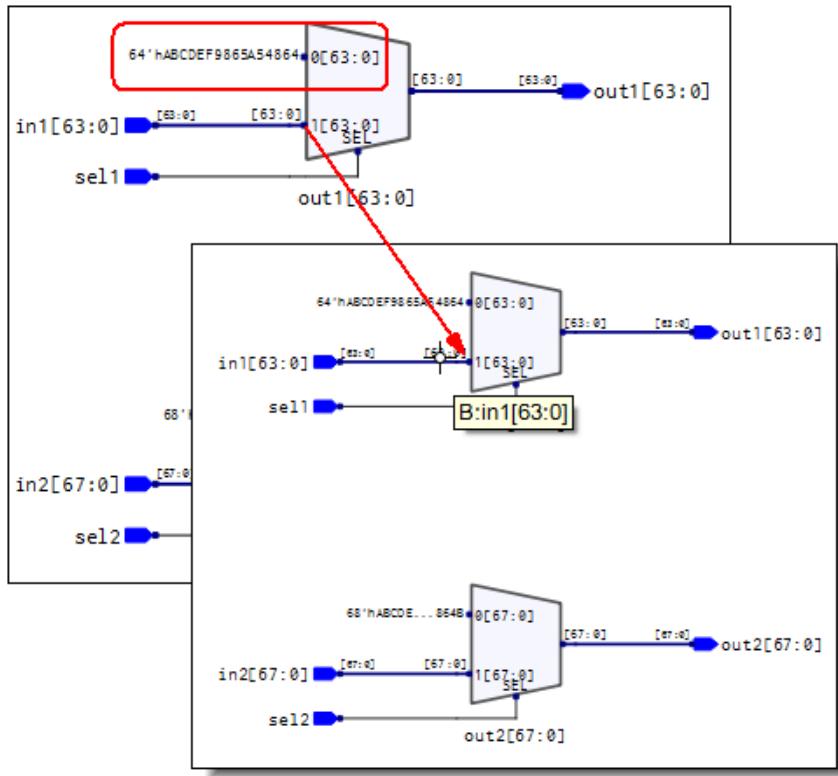
For example, use this field with the collection commands to identify groups of objects in the schematic.

The screenshot shows a properties dialog box titled "decode". It contains a table with two columns: "Property" and "Value". The properties listed are: arrival\_time (Value: 0), inout\_pin\_count (Value: 5), input\_pin\_count (Value: 15), inst\_of (Value: ins\_decode), is\_hierarchical (Value: 1), is\_verilog (Value: 1), and lib.cell.view (Value: work.ins\_decode.verilog). A "Close" button is at the bottom right of the dialog.

Property	Value
arrival_time	0
inout_pin_count	5
input_pin_count	15
inst_of	ins_decode
is_hierarchical	1
is_verilog	1
lib.cell.view	work.ins_decode.verilog

## Viewing Objects with Constant Values

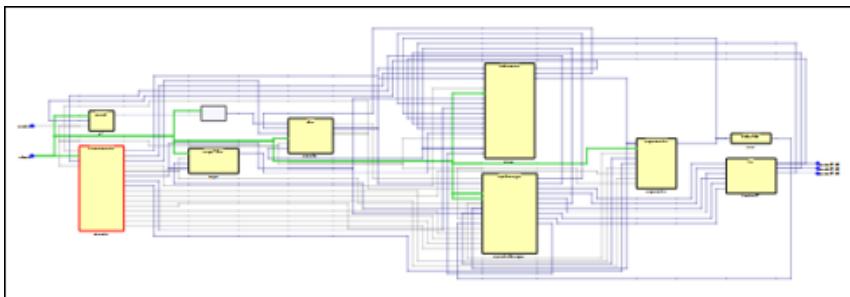
You can display constants and view their constant values. Individual functions of the constant are displayed. Use tool tips to see the full value for the constant.



## Viewing Objects in a Source File

The HDL Analyst view provides various ways to view objects in a source file. For example:

1. Select an object in the schematic view.



2. Then, right-click and select one of the following:
  - View Instance in Source – Opens the RTL source file and finds the instantiated instance selected.

```

1 module eight_bit_uc ( clock, resetn, porta, portb, portc );
2
3
4 input clock, resetn;
5
6 inout [7:0] porta, portb, portc;
7
8 reg [7:0] alua_tmp2;
9 wire resetn, aluz;
10 wire [11:0] inst;
11 wire f_we, w_we, status_z_we, status_c_we, tris_we, skip;
12 wire [7:0] k;
13 wire [4:0] fsel;
14 wire [8:0] longk;
15 wire [3:0] aluop;
16 wire [1:0] alua_sel, alub_sel;
17 wire bdpol;
18 wire opcode_call, opcode_goto, opcode_retlw;
19 wire [2:0] b_mux;
20 wire [7:0] fin;
21 wire [10:0] pc;
22 wire [7:0] regfile_out;
23 wire [7:0] fsrc, rtcc;
24 wire [7:0] status, porta, portb, portc, w;
25 wire [7:0] alua, alub;
26 wire alu_out;
27 wire [7:0] aluout;
28 wire [11:0] romdata;
29 wire [7:0] port_int_a;
30 wire [7:0] port_int_b;
31 wire [7:0] port_int_c;
32 wire [7:0] trisa;
33 wire [7:0] trisb;
34 wire [7:0] trisc;
35 wire clk1, clk2, clk3, clk4;
36 wire [7:0] alua_tmp;
37 assign clk1 = clock;
38 assign clk2 = clock;
39 assign clk3 = clock;
40 assign clk4 = clock;
41 assign rtcc = 0;
42 assign b_mux = 0;
43 // instantiating prep4 block
44 prep4 pl (alua_tmp, alua, clk3, resetn);
45
46 // instantiating decode block
47 ins_decode Decode (clk2, resetn, aluz, inst, f_we, w_we,
48                     status_z_we, status_c_we, tris_we,
49                     skip, k, fsel, longk, aluop,
50                     alua_sel, alub_sel, bdpol, opcode_goto,
51                     opcode_call, opcode_retlw );

```

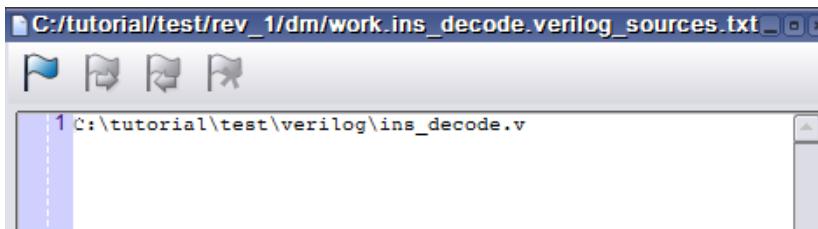
- View Module in Source – Opens the RTL source file and finds the instantiated module selected.

```

1 module ins_decode ( clk2,resetn ,aluz, inst, f_we, w_we,
2                      status_z_we, status_c_we, tris_we,
3                      skip, k, fsel, longk, aluop,
4                      alua_sel, alub_sel, bdpol, opcode_goto,
5                      opcode_call, opcode_retlw);
6 input clk2, resetn;
7 input aluz;
8 input [11:0] inst;
9
10 inout f_we;
11
12
13 output w_we, status_z_we, status_c_we, tris_we, skip;
14 output [7:0] k;
15
16 inout [4:0] fsel;
17
18 output [8:0] longk;
19
20 output [3:0] aluop;
21
22 output [1:0] alua_sel, alub_sel;
23
24 output bdpol;
25
26 output opcode_call, opcode_retlw, opcode_goto;
27
28 reg [13:0] decodes, decodes_in;
29
30 wire reset = -resetn;
31 // this is never used either take it out
32 // or hook it up if necessary
33 //reg [7:0] bit_decoder;
34

```

- View dependent source file list – A source file (*moduleName\_source.txt*) containing the specified module is created in the dm directory of the Implementation Results directory.



- View source file list – A source file (*designName\_source.txt*) containing the specified modules for the design is created in the dm directory of the Implementation Results directory.

```

1 C:\tutorial\test\verilog\mult.v
2 C:\tutorial\test\verilog\alu.v
3 C:\tutorial\test\verilog\spcl_regs.v
4 C:\tutorial\test\vhdl\ins_rom.vhd
5 C:\tutorial\test\verilog\reg_file.v
6 C:\tutorial\test\verilog\pc.v
7 C:\tutorial\test\verilog\state_mc.v
8 C:\tutorial\test\verilog\io.v
9 C:\tutorial\test\verilog\data_mux.v
10 C:\tutorial\test\verilog\ins_decode.v
11 C:\tutorial\test\verilog\eight_bit_uc.v

```

## Selecting Objects in the Schematic

For mouse selection, standard object selection rules apply:

To select ...	Do this ...
Single objects	<p>Click the object in the schematic, or click the object name in the Hierarchy Browser.</p> <p>Tcl equivalent: <b>select {i:instanceName}</b></p> <p>For a net: <b>select {n:netName}</b></p> <p>For a pin: <b>select {t:pinName}</b></p> <p>For a port: <b>select {p:portName}</b></p>
Multiple objects	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>Draw a rectangle around the objects.</li> <li>Select an object, press Ctrl, and click other objects you want to select.</li> <li>Select multiple objects in the Hierarchy Browser. See <a href="#">Browsing With the Hierarchy Browser, on page 227</a>.</li> <li>Use Find to select the objects you want. See <a href="#">Finding Objects, on page 227</a>.</li> </ul> <p>Tcl equivalent: <b>select {{i:instance1} {i:instance2}}</b></p> <ul style="list-style-type: none"> <li>Select all instances in the current view or press Ctrl+A:</li> </ul> <p>Tcl equivalent: <b>select -instances</b></p> <ul style="list-style-type: none"> <li>Select all primitives in the current view or press Ctrl+Alt+A:</li> </ul> <p>Tcl equivalent: <b>select -primitives</b></p>

To select ...	Do this ...
Objects by type (instances, ports, nets)	Use Find to select the objects (see <a href="#">Browsing With the Find Command, on page 233</a> ), or use the Hierarchy Browser, which lists objects by type.
No objects (deselect all currently selected objects)	Click the left mouse button in a blank area of the schematic. Deselected objects are no longer highlighted. Tcl equivalent: <b>select -clear</b>

The HDL Analyst view highlights selected objects in red. If you have other windows that are cloned, the selected object is highlighted in the other windows as well (crossprobing).

## Selecting a Sequence of Objects in the Schematic

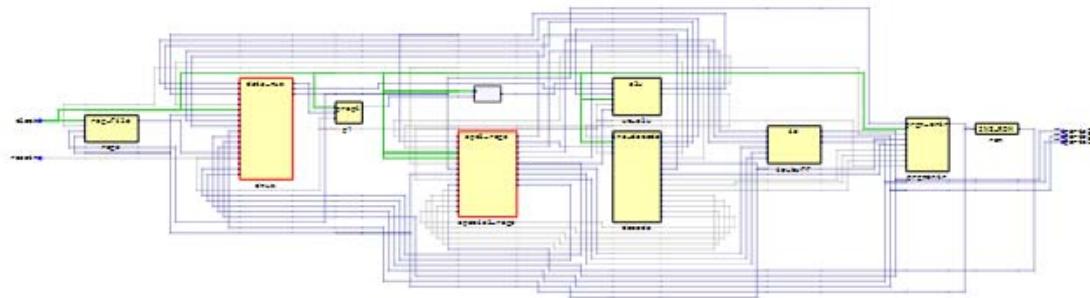
You can select a series of objects in the schematic, then traverse back and forth through these selections in the order they were chosen. Use the backwards and forwards icons ( ) to move between each selection. This is handy to help you undo or redo any changes with your selections. Once you select an operation, the other selections become unavailable, unless the view changes.

When an object is selected in the schematic, you can use the following command to print the name of the object (instance, port, pin, or net):

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

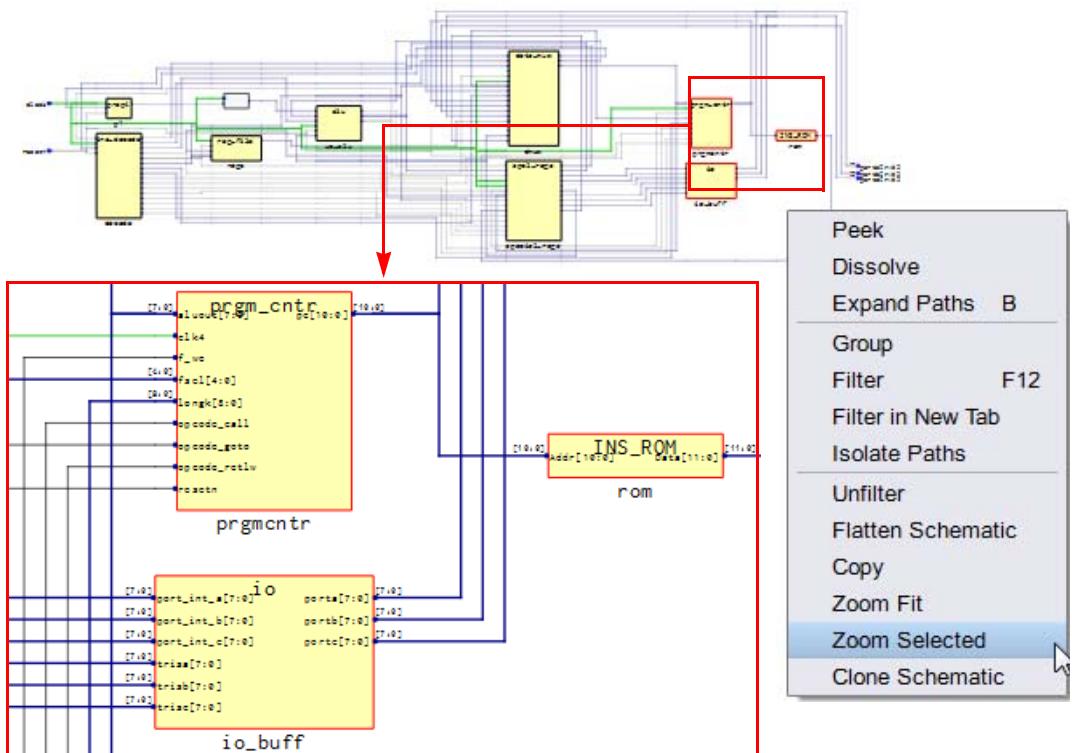
In the following example, two instances are selected. Specify the following command to display their names in the Tcl window:

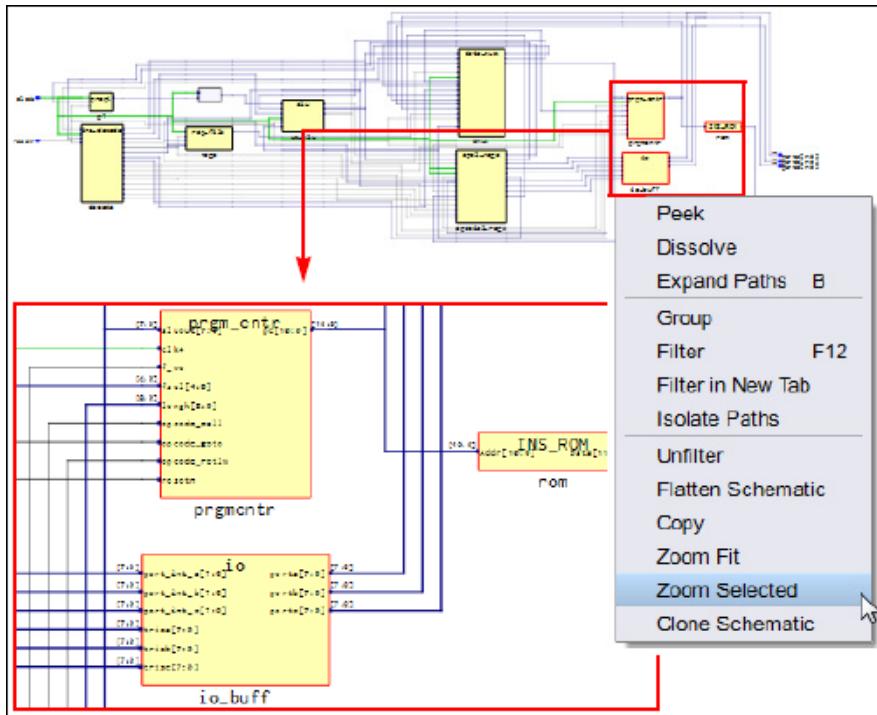
```
% analyst get_selected -inst
{i:dmux} {i:special_regs}
```



## Zooming in on Selected Objects

Once you have selected objects in the schematic view, you can automatically zoom in on these objects. To do this, highlight the required objects, then right-click and select **Zoom Selected** from the drop-down menu.





## Grouping Objects in the Schematic

You can group objects in the schematic. Sometimes the HDL Analyst tool automatically determines groups shown with the color purple below. When you push into the group block, the content of its objects is displayed.

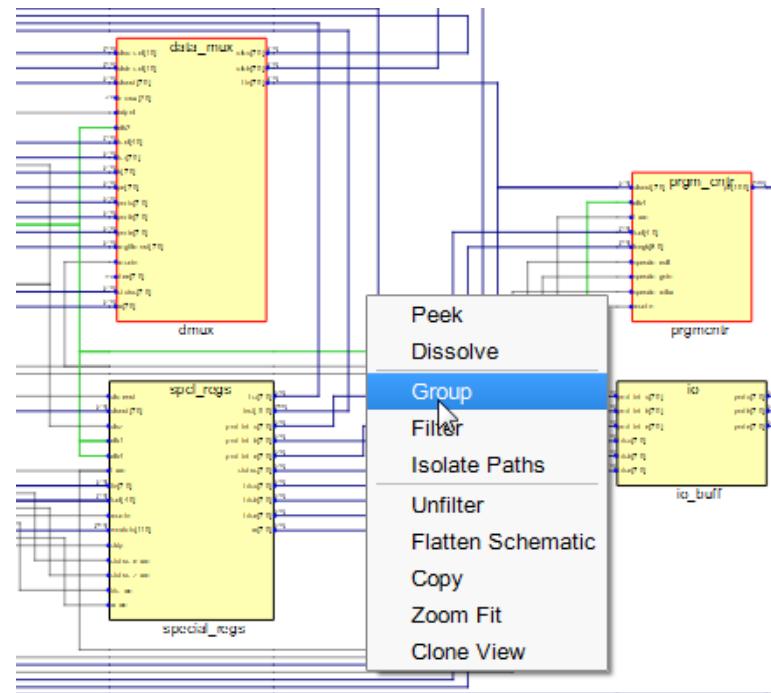
The HDL Analyst tool automatically groups instances with similar names at all levels of hierarchy, when you enable the Allow Automatic Grouping option on the HDL Analyst Options dialog box. For example, suppose there are three registers with the names `out_reg[1]`, `out_reg[2]`, and `out_reg[3]`. A group will be created with the registers having the name `out_reg[3:1]`.

To create your own groups:

1. Select the specified objects in the view.

2. Right-click and select the Group option from the pop-up menu.

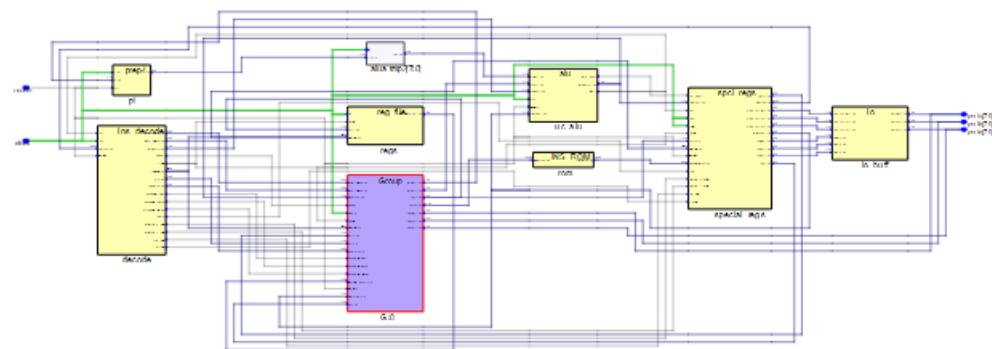
This creates a dummy hierarchy that groups the objects together, which is only displayed in the HDL Analyst GUI. It does not generate any hierarchical changes to the design.



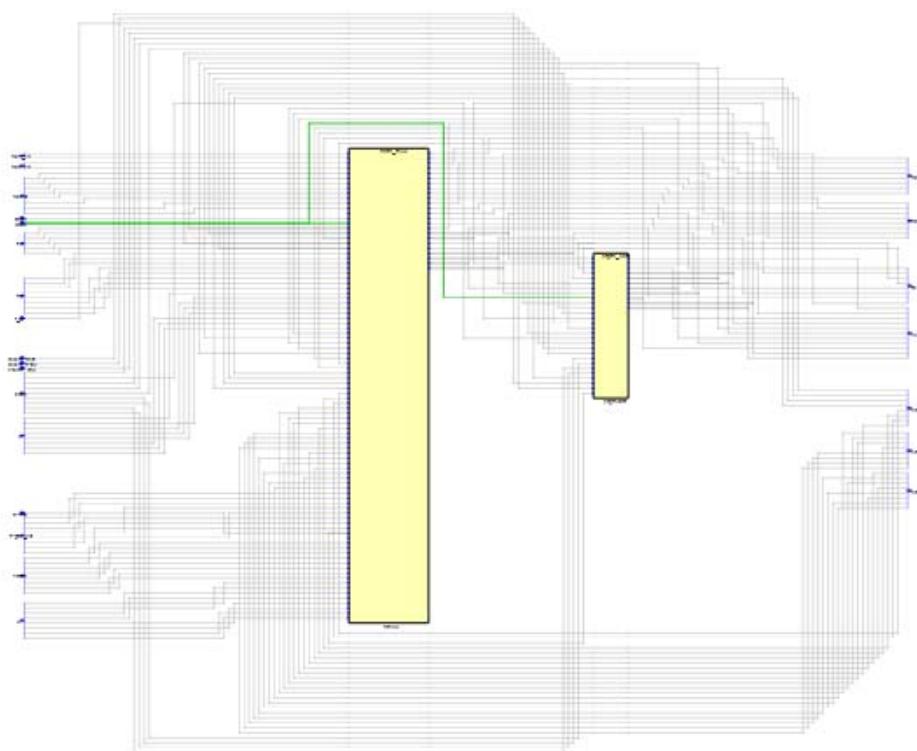
3. Specify a group name in the dialog box that pops up.



This forms a group of objects in group1 that is displayed with the purple block.

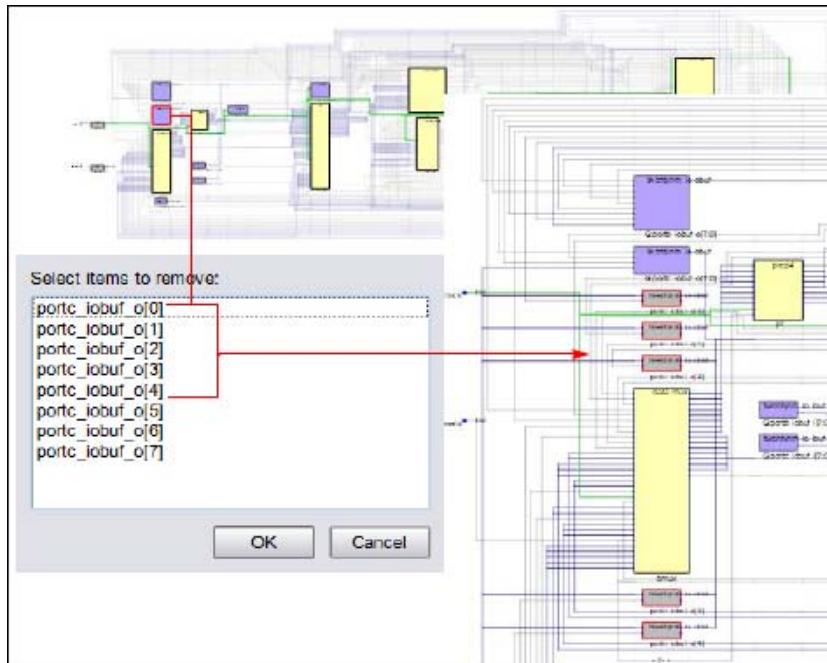


4. Push into the purple block to see the grouped objects.



5. To ungroup selections, right-click and select Dissolve from the drop-down menu.

6. To remove individual instances of a group, right-click and select Partial Dissolve from drop-down menu. A dialog box is displayed where you can select the items to remove.



7. User-created groups are not saved when you close and re-open the same netlist.

## Moving Between Views in a Schematic Window

When you filter or expand your design, you move through a number of different design views in the same schematic window. For example, you might start with a view of the entire design, then filter an object and finally expand a connection in the filtered view, for a total of three views. You can also move back after flattening a view.

1. To move back to the previous view, click the Back icon ().

The software displays the last view.

2. To move forward again, click the Forward icon (  ).

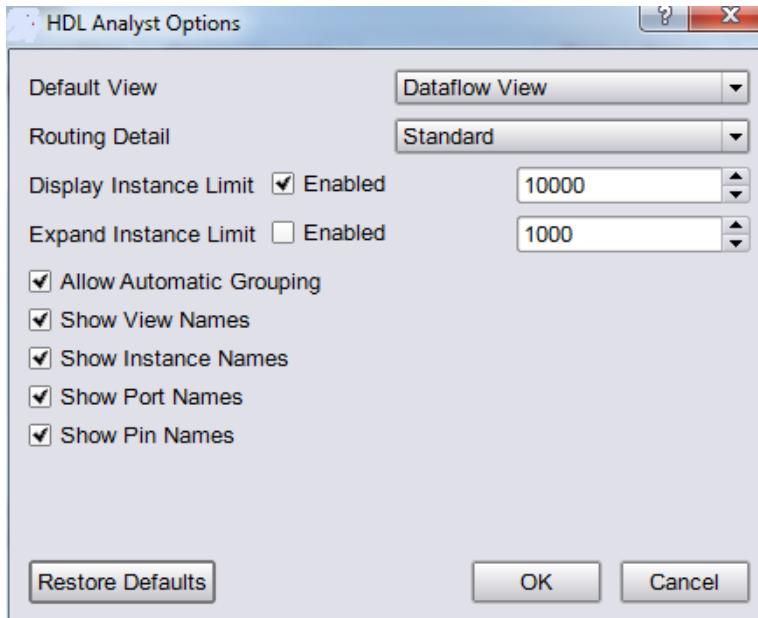
The software displays the next view in the display history.

## Setting Schematic Preferences

You can set various preferences for the schematic from the user interface.

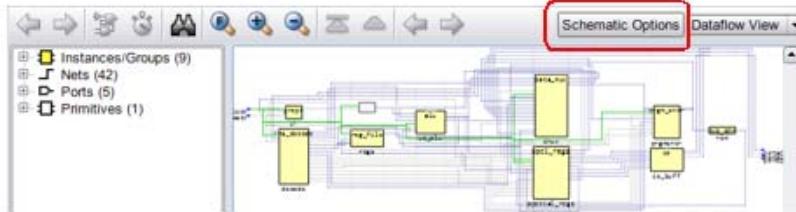
1. Select either:

- HDL Analyst->Schematic Options
- Options->Schematics Options



For a description of all the options on this form, see [HDL Analyst Options Command](#), on page 395 in the *Command Reference Manual*.

- Also, for your convenience you can simply select the Schematic Options button from the top of the HDL Analyst view.



2. The table details the following operations:

To ...	Do this ...
Specify how you want the schematic to display.	Select Clock View or Dataflow View (default).
Specify how the tool determines the detailed routing for the design.	Select Standard (default) or Quick (direct connection).
Show names for the view, instances, ports, and pins.	Enable any of the following: <ul style="list-style-type: none"> <li>• Show View Names</li> <li>• Show Instance Names</li> <li>• Show Port Names</li> <li>• Show Pin Names</li> </ul>
Show the out of date popup message for the design.	When enabled, shows the design out of date popup message if the design file has changed while the HDL Analyst view was opened.
Specify limits when displaying or expanding instances.	Set the limit and select Enabled.
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.
Determine if you want automatic grouping for the design.	When Allow Automatic Grouping is enabled, the tool automatically groups instances with similar names at every level in the design.

3. Enable the Show design out of date popup message option to ensure that the correct version of the HDL Analyst view is being displayed. You might be

looking at inconsistent results, if the design netlist file (srs) has changed. You can choose to close the current HDL Analyst view and reload the updated version.

When you select this option, the following warning message is displayed at the bottom of this dialog box.



# Exploring Design Hierarchy

Schematics generally have a certain amount of design hierarchy. You can move between hierarchical levels using the Hierarchy Browser mode. For additional information, see [Analyzing With the HDL Analyst Tool, on page 245](#). See [Traversing Design Hierarchy with the Hierarchy Browser, on page 221](#).

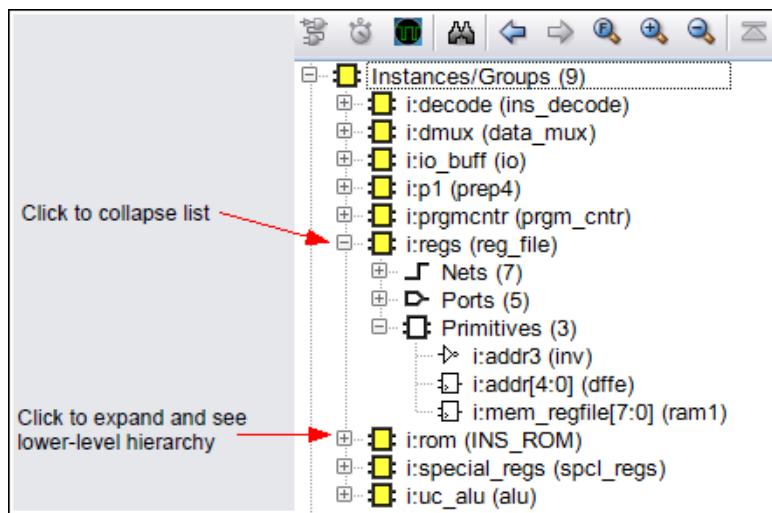
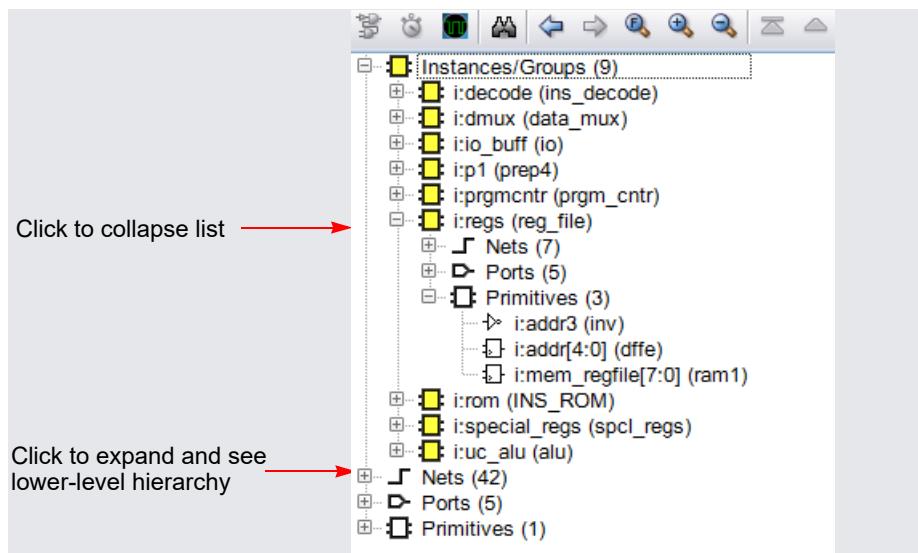
## Traversing Design Hierarchy with the Hierarchy Browser

The Hierarchy Browser is the list of objects on the left side of the schematic view. It is best used to get an overview, or when you need to browse and find an object. If you want to move between design levels of a particular object, using the Push command is more direct. Refer to [Exploring Object Hierarchy with Push/Pop Commands, on page 223](#) for details.

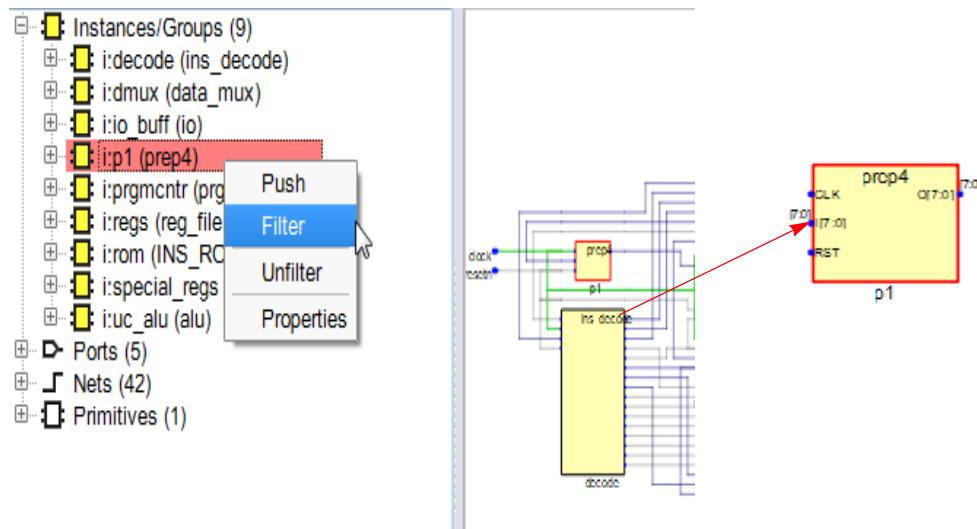
The hierarchy browser allows you to traverse and select the following:

- Instances or Groups
- Ports
- Internal nets

The browser lists the objects by type. Use the expand (⊕) and collapse (⊖) signs to ascend or descend the hierarchy.



1. You can perform some similar operations as done in the schematic view, such as filtering an object from the Hierarchy Browser. For example, highlight an instance, then right-click and select Filter as shown below.



2. You can also crossprobe to instances and modules in the source file from the Hierarchy Browser. Right-click and select either:
  - View Instance in Source
  - View Module in Source

For details, see [Crossprobing, on page 238](#).

3. To extract logic for a partially dissolved net:
  - Select a partially selected net from the hierarchy browser.
  - Right-click and select Extract Net from the drop-down menu in the HDL Analyst view.

## Exploring Object Hierarchy with Push/Pop Commands

To view the internal hierarchy of a specific instance, use the Push/Pop commands from the drop-down menu or mouse strokes. When combined with other commands like filtering and expansion commands, Push/Pop can be a very powerful tool for isolating and analyzing logic. See [Filtering](#)

[Schematics, on page 249](#) and [Expanding Pin and Net Logic, on page 251](#) for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

- [Pushing into Objects, on page 224](#)
- [Popping up a Hierarchical Level, on page 225](#)

## Pushing into Objects

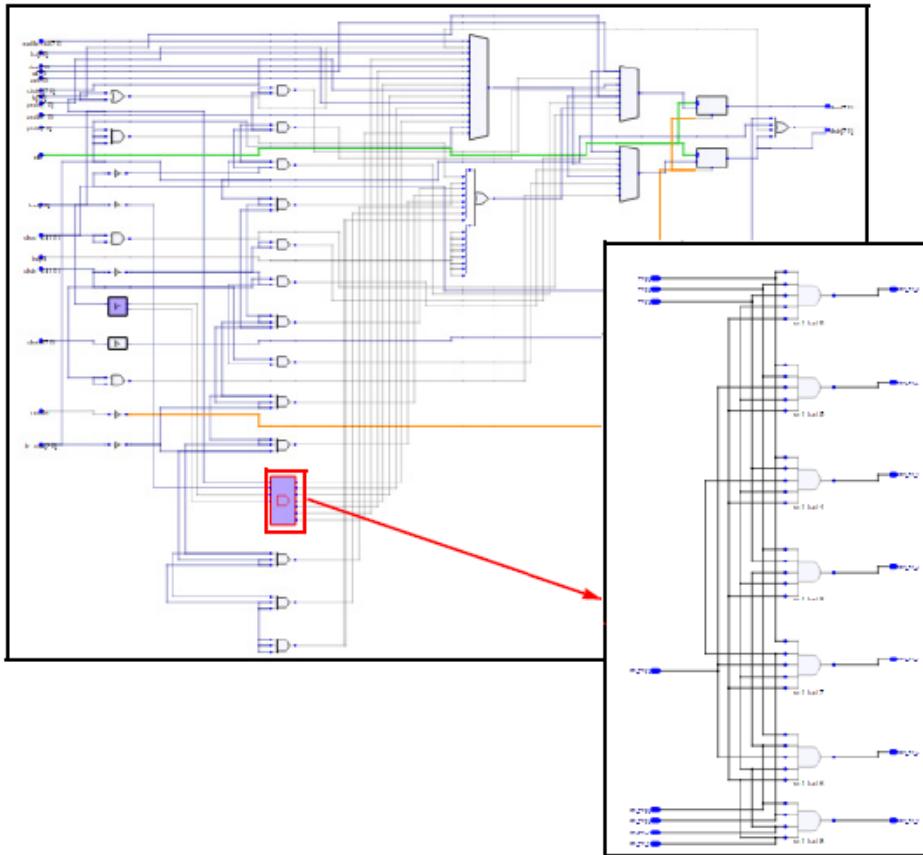
In the schematic, you can push into instances and view the lower-level hierarchy. You can use a mouse stroke or the command to push into objects:

1. To move down a level (push into an object) with a mouse stroke, put your cursor near the top of the object, hold down the right mouse button, and draw a vertical stroke from top to bottom. You can push into the following objects; see step 3 for examples of pushing into different types of objects.
  - Hierarchical instances. They can be displayed as pale yellow boxes (opaque instances).
  - Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the schematic, under `i:instanceNames ->Primitives`.
  - Instances formed into a group.

The remaining steps show you how to use the icon or command to push into an object.

2. Enable the Push/Pop command by doing one of the following:
  - Double-click on the object.
  - Right-click in the view and select Push/Pop from the drop-down menu.
  - Use the mouse strokes.

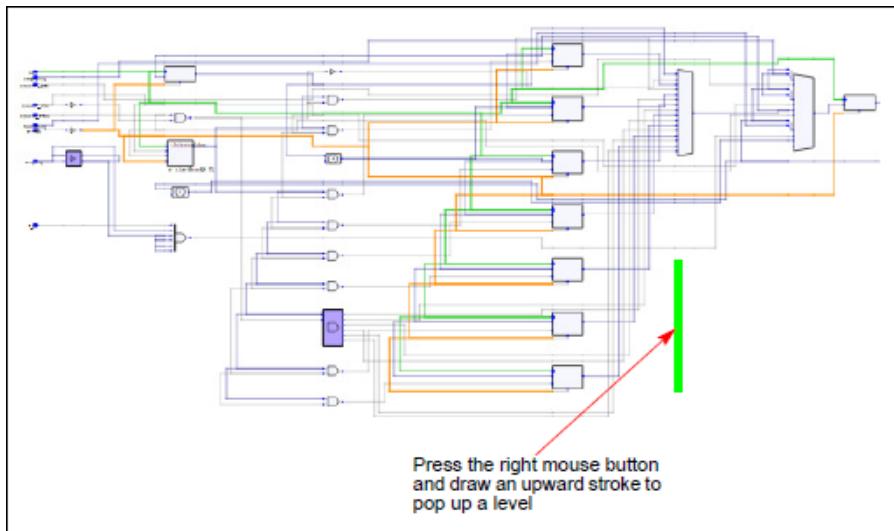
After pushing into an instance, the following schematic is displayed. Notice the purple block that groups gates with similar names together. You can push into this block as well.



3. To push (descend) into an object, double-click the hierarchical object.  
The following figure shows the result of pushing into a ROM.

## Popping up a Hierarchical Level

1. To move up a level (pop up a level), put your cursor anywhere in the design,
  - Use the Pop Hierarchy icon (  ).
  - Hold down the right mouse button, and draw a vertical mouse stroke, moving from the bottom upwards.



The software moves up a level, and displays the next level of hierarchy.

2. Alternatively, you can double-click on any whitespace in the view to pop up a level from where you pushed into the hierarchy.

# Finding Objects

In the schematics, you can use the Hierarchy Browser or the Find command to find objects, as explained in these sections:

- [Browsing to Find Objects in HDL Analyst Views, on page 227](#)
- [Using Wildcards with the Find Command, on page 237](#)

## Browsing to Find Objects in HDL Analyst Views

You can always zoom in to find an object in the schematic. Use Zoom Fit to quickly fit all objects into the schematic. The following procedure shows you how to browse through design objects and find an object at any level of the design hierarchy. You can use the Hierarchy Browser or the Find command to do this. If you are familiar with the design hierarchy, the Hierarchy Browser can be the quickest method to locate an object. The Find command is best used to graphically browse and locate the object you want.

### Browsing With the Hierarchy Browser

1. In the Hierarchy Browser, click the name of the net, port, or instance you want to select.

The object is highlighted in the schematic.

2. To select a range of objects, you can press and hold the Shift key while clicking the selected objects in the range.

The software selects and highlights all the objects in the range.

3. If the object is on a lower hierarchical level, do either of the following:

- Expand the appropriate higher-level object by clicking the collapsed symbol next to it, and then select the object you want.
- Push down into the higher-level object, and then select the object from the Hierarchy Browser.

The selected object is highlighted in the schematic. However, you may have to filter the object to view it in the design hierarchy.

4. To select all objects of the same type, select them from the Hierarchy Browser. For example, you can find all the nets in your design.

## Browsing With the New Hierarchy Browser

### *Beta*

Using the HDL Analyst tool, you can display a hierarchy of design objects in the Hierarchy Browser. Typically, the time taken to display a design hierarchy depends on the size of the design.

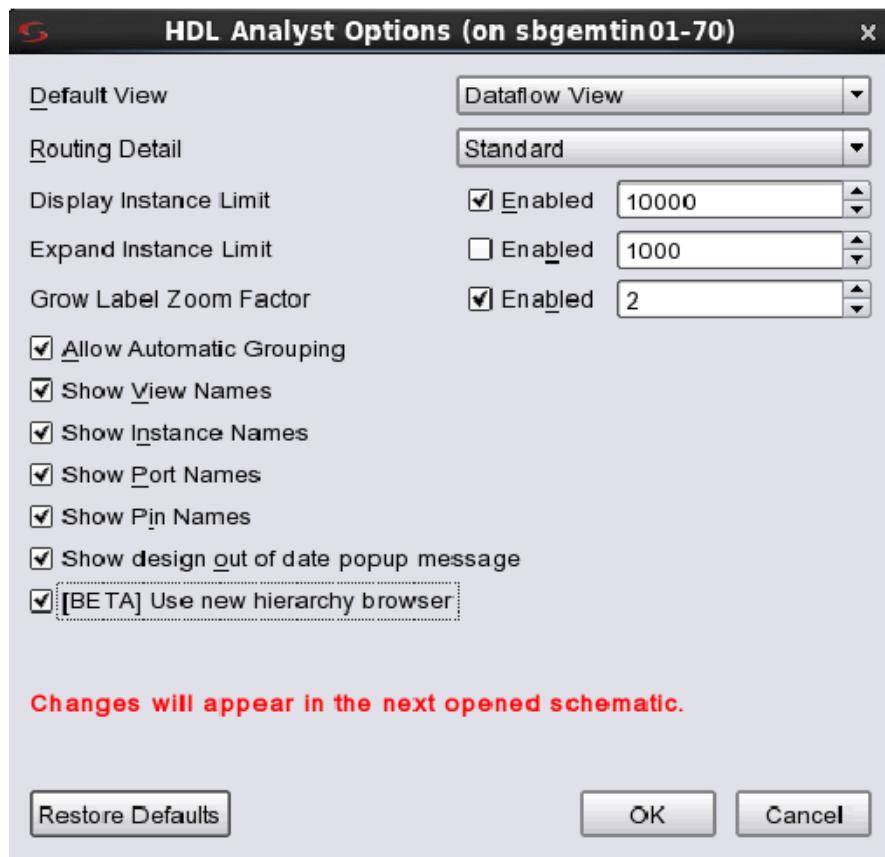
To speed up this process, the new Hierarchy Browser traverses the entire (text-based) netlist to quickly extract hierarchical instance data. This helps to display the entire netlist hierarchy quickly and also facilitates the viewing of custom instances on demand, instead of traversing down the design hierarchy. This flow is advantageous in large designs, to display the hierarchy and view any instance, quickly.

To enable the new Hierarchy Browser, follow these steps:

1. To set preference in the schematic from the user interface, select one of the following:
  - HDL Analyst->Schematic Options
  - Options->Schematics Options

The HDL Analyst Options dialog box is displayed.

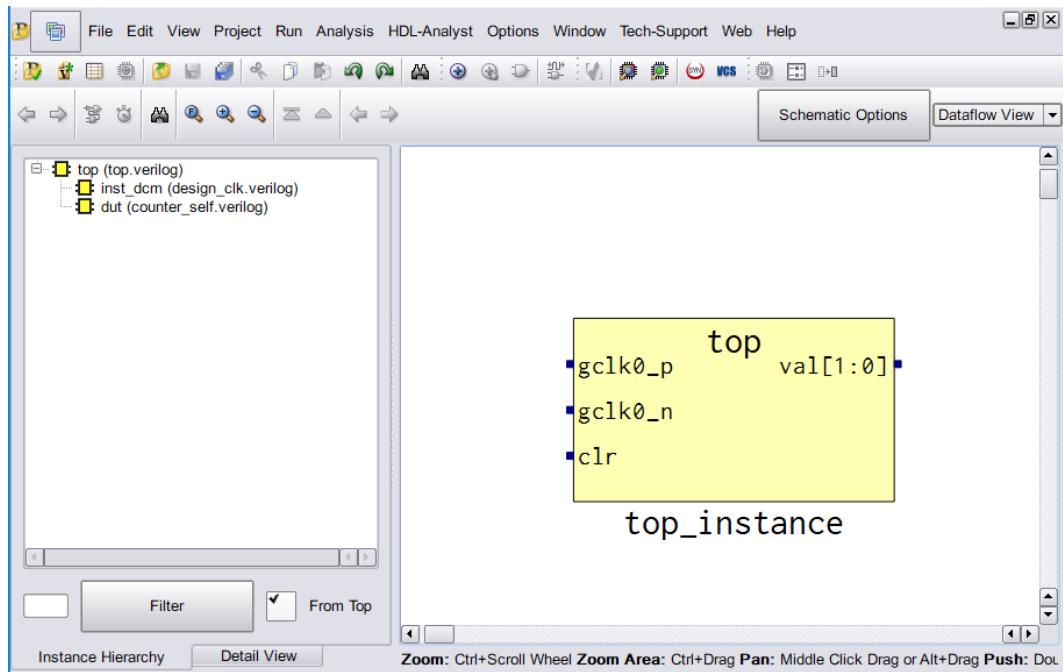
2. Enable the [BETA] Use new hierarchy browser option on the HDL Analyst Options dialog box. A warning is displayed, as shown below.



When you enable the Use new hierarchy browser option in the HDL Analyst Options dialog box, the HDL Analyst tool displays the RTL schematic at the instance level (Instance Hierarchy tab) and design level (Design View tab), when the next schematic displayed.

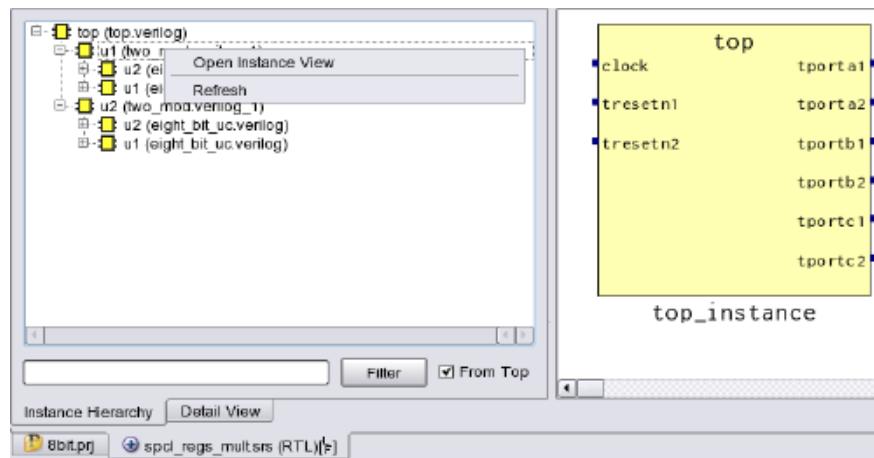
The Instance Hierarchy tab is visible only when the Use new hierarchy browser option is enabled in the Schematic Options (HDL Analyst Options) dialog box.

3. Click OK to close the dialog box. Now, each design you view will follow the updated option settings.
4. Select the RTL View icon or select RTL->Hierarchical View from the HDL-Analyst menu, to view the RTL view of the design.



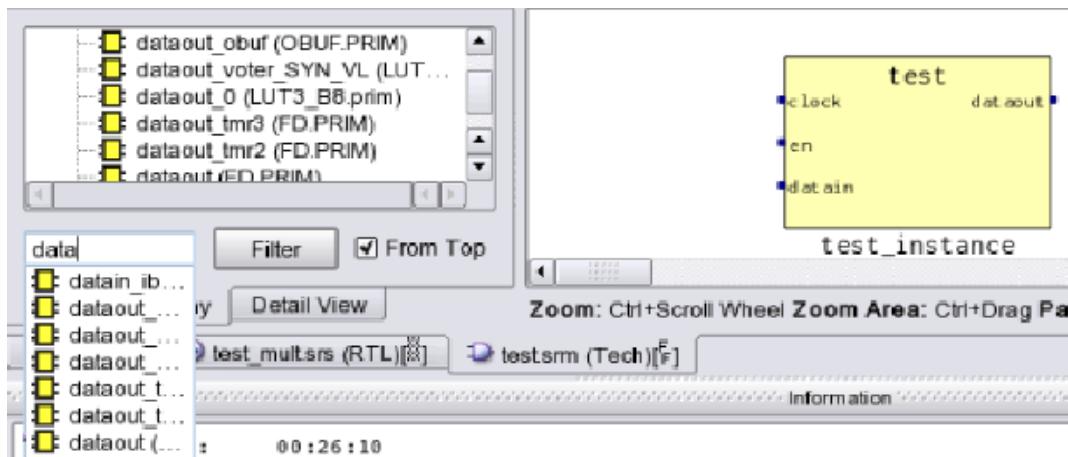
5. Open the instance by one of these methods:

- Double-click on the instance.
- Right-click and select Open Instance View.



## Browsing an Object by Filtering or Loading

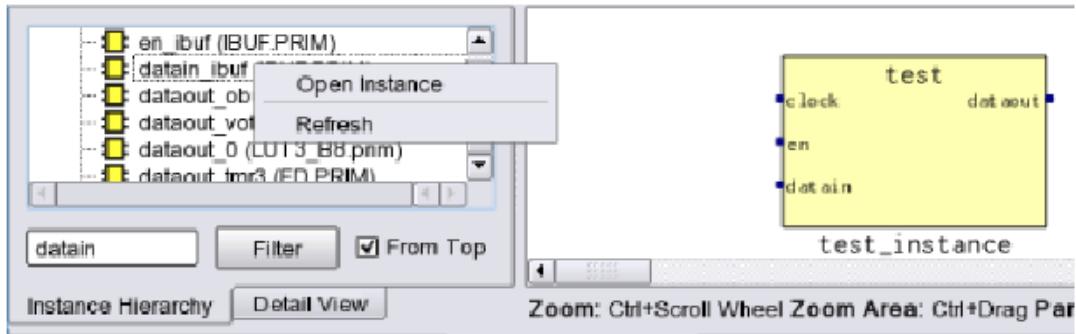
1. To filter a particular instance, type the instance name into the text box available below the Hierarchy Browser pane as shown below:



The instance names become visible in the drop-down as you continue to type.

2. Choose the instance and click Filter to display the instance in the RTL schematic view.
3. To search through a hierarchical path, select From Top. If this is not checked, the command searches the entire design.
4. To load an instance from the Hierarchy Browser, right-click the desired instance and select the Open Instance View ... option.

The instance is displayed in the RTL schematic view.



The Instance Hierarchy tab is visible only when the Use new hierarchy browser option is enabled in the Schematic Options (HDL Analyst Options) dialog box.

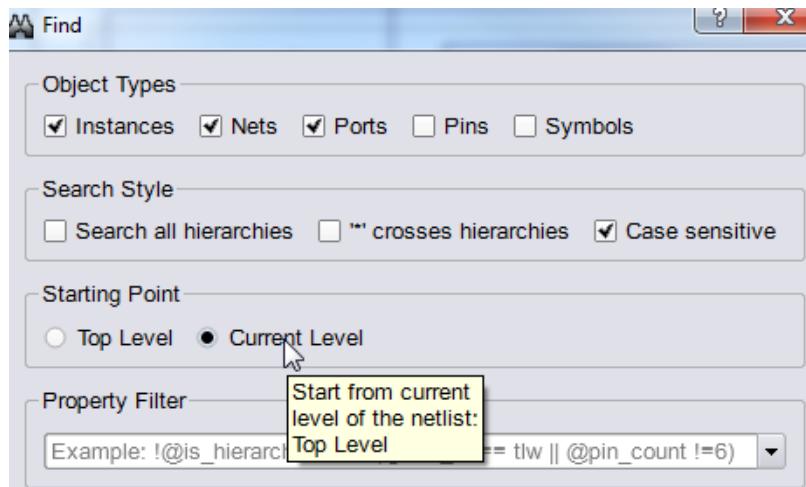
### New Hierarchy Browser Limitations:

Consider these limitations before using the new Hierarchy Browser flow:

- Currently, usable only with compiled-stage/mapped-stage netlists.
- In map designs, primitives and black boxes appear in the hierarchy but not in the RTL view.
- Certain view options while filtering or loading an instance from the new Hierarchy Browser do not work correctly.

## Browsing With the Find Command

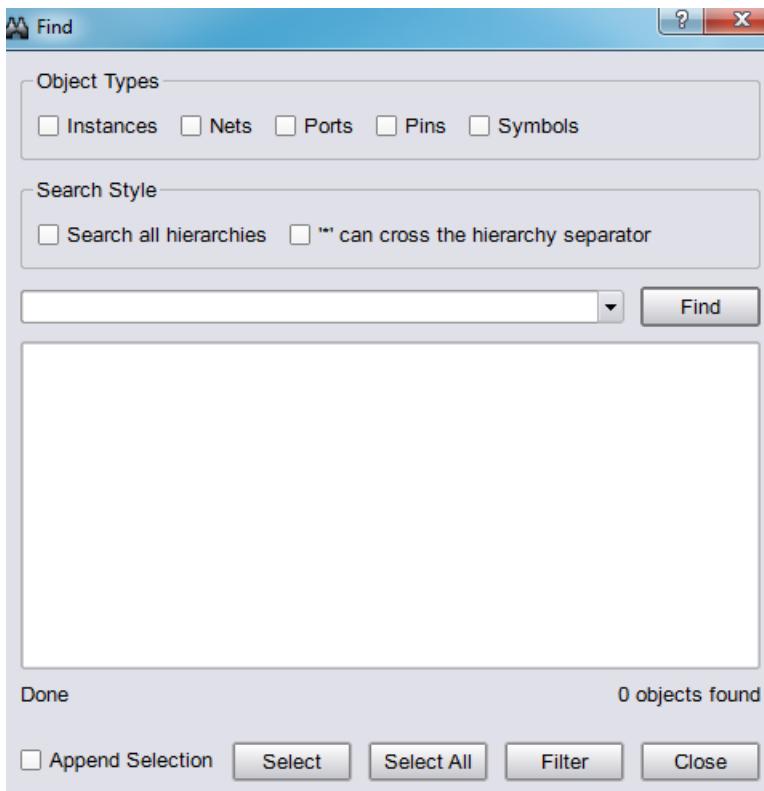
1. In a schematic, select the search icon ( ) or press Ctrl-f to open the Find dialog box.
2. Do the following in the dialog box:
  - Select the type of objects to find: instances, nets, ports, pins, and/or symbols.
  - Specify how you want the search to occur: for all hierarchies and/or allow \* to search across the hierarchy separator.
  - Specify whether to search using case-sensitive designations for objects.
  - Start the search either from the top level or current level of the schematic view. Use the tool tip in this dialog box to display the current level starting point.



When searching, note the following:

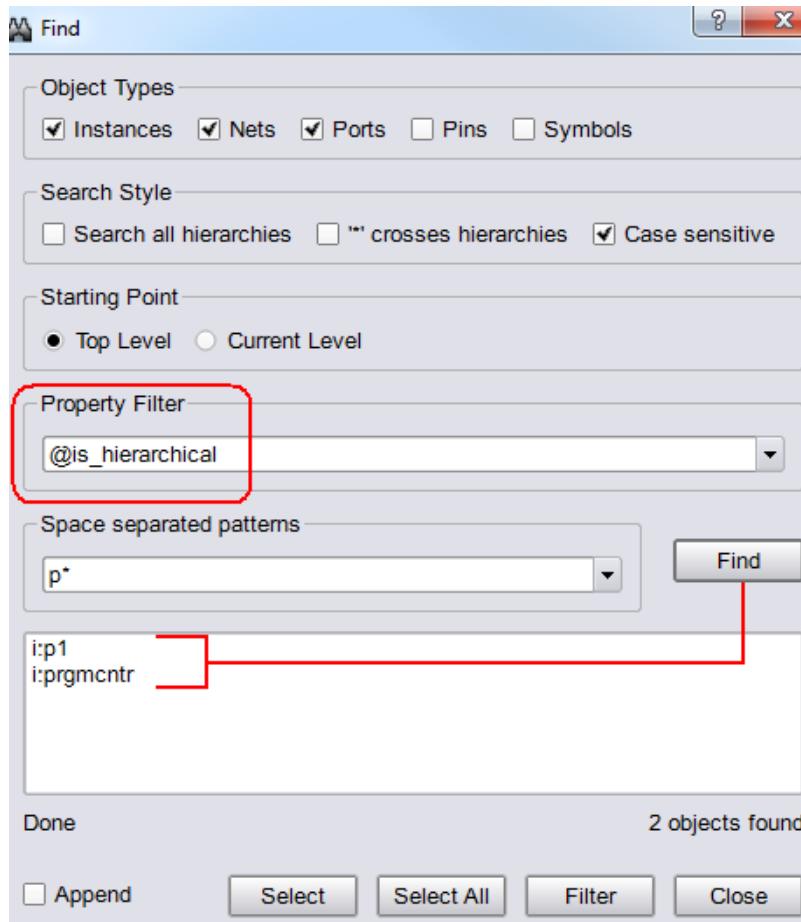
- Double-click a selected object from the dialog box to filter it in the schematic view.
- Use the Space separated patterns field to search for multiple patterns, specifying the patterns with spaces in the search field.

- Once multiple objects have been selected from the dialog box, you can highlight them, then copy and paste them to the Tcl window or in a text file.
3. Select an object displayed in the dialog box below, then click the Select button.
- Click the Filter button, to select the specified objects and filter them in the HDL Analyst view.
  - Click the Close button to end the Find search. Then, you can use the Filter command to display the objects.

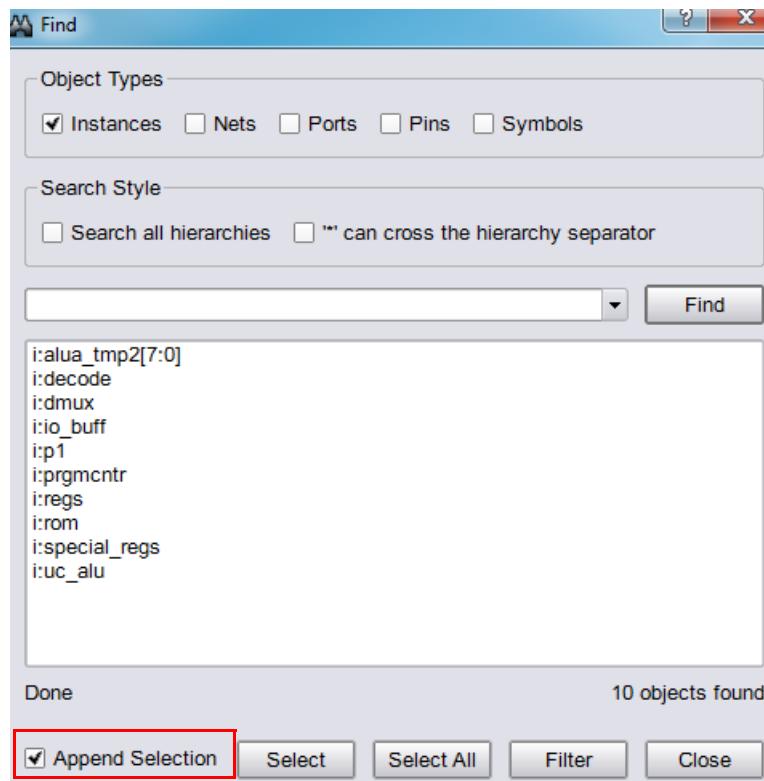


When the search style options (Search all hierarchies and ".," can cross the hierarchy separator) are not enabled, the software searches for objects at the top level.

4. You can filter on the results found for objects based on the Property Filter field and using the search patterns specified in the Space separated patterns field. For example, suppose you want to search for the @is\_hierarchical property for the design. Specify the pattern as shown in the following dialog box and click Find. The results are displayed below:



5. If you enable the Append Selection option, objects selected in the current display window are appended to each other when you click the Select or Select All button. Otherwise, objects will be overridden after each selection.



- When Append Selection is enabled, you can also search for multiple patterns, then filter them in the schematic view by clicking the Filter button.
- If you determine that the search is taking too long to run, notice that the Find button changes to Stop. Click Stop.

The multi-threaded Find command can be interrupted and canceled once the search term has been identified. If you let the search complete, you will see Finishing and Done appearing under the display window.

## Using Wildcards with the Find Command

Use the following wildcards when you search the schematics:

- \*     The asterisk matches any sequence of characters.
- ?     The question mark matches any single character, but not the hierarchy separator by default.
- .     The dot explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not a hierarchy separator, type a backslash before the dot: \.

# Crossprobing

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. Crossprobing helps you visualize where coding changes or timing constraints might help to reduce area or improve performance.

This section describes how to crossprobe from different views. It includes the following:

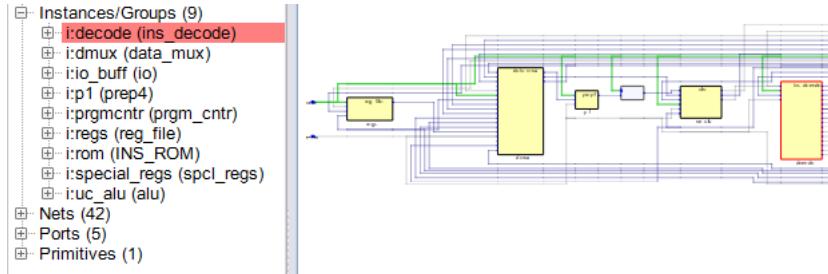
- [Crossprobing within a View, on page 238](#)
- [Crossprobing from an HDL Analyst View, on page 239](#)
- [Crossprobing to the Source Code, on page 240](#)
- [Crossprobing to the Source Code, on page 240](#)

## Crossprobing within a View

Selecting an object name in the Hierarchy Browser highlights the object in the schematic, and vice versa.

Selected Object	Highlighted Object
Instance in schematic (single-click)	Module icon in Hierarchy Browser
Net in schematic	Net icon in Hierarchy Browser
Port in schematic	Port icon in Hierarchy Browser
Logic icon in Hierarchy Browser	Instance in schematic
Net icon in Hierarchy Browser	Net in schematic
Port icon in Hierarchy Browser	Port in schematic

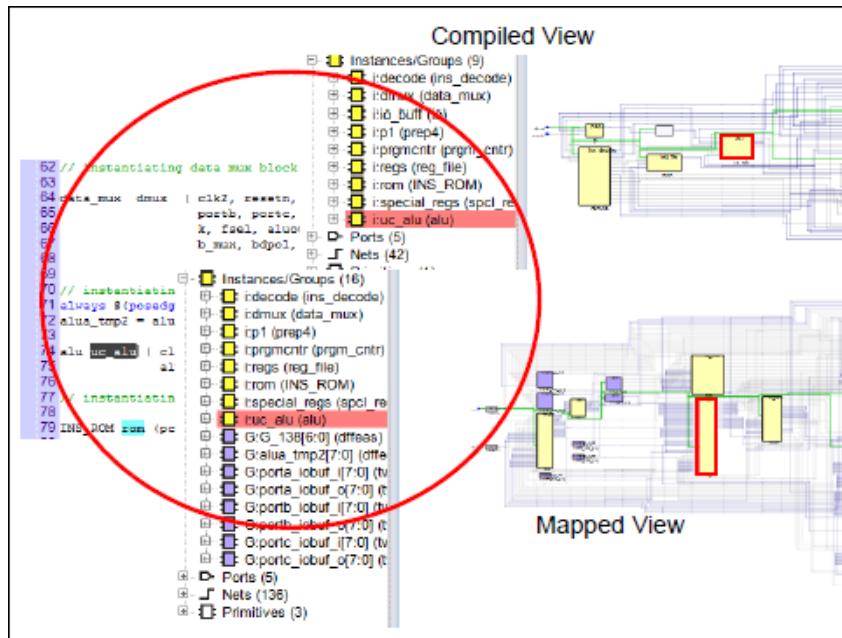
In this example, when you select the DECODE module in the Hierarchy Browser, the DECODE module is automatically selected in the view.



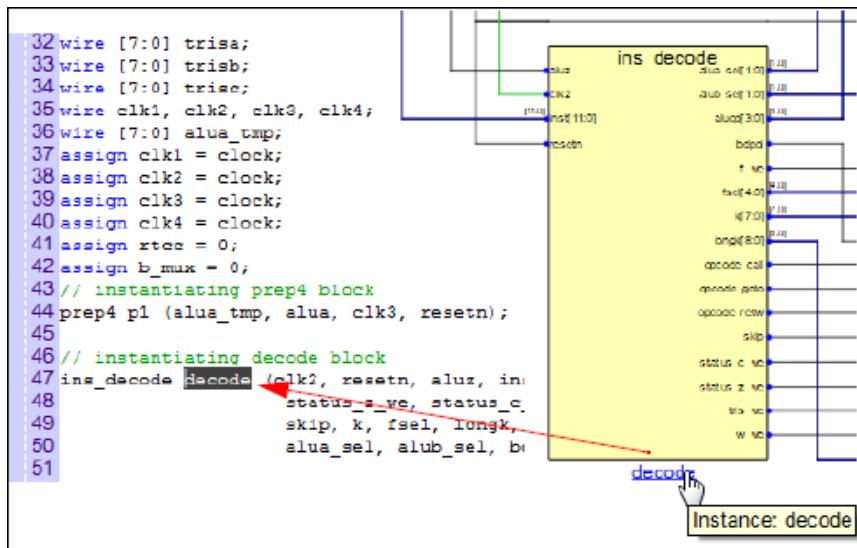
## Crossprobing from an HDL Analyst View

To crossprobe from the schematic to other open views or the source code files, select the object by clicking on it.

The software automatically highlights the object in all open views. If the open view is a schematic, the software highlights the object in the Hierarchy Browser on the left as well as in the schematic. If the highlighted object is in another hierarchy of a schematic, the view does not automatically track to the hierarchy. You may have to filter the schematic.



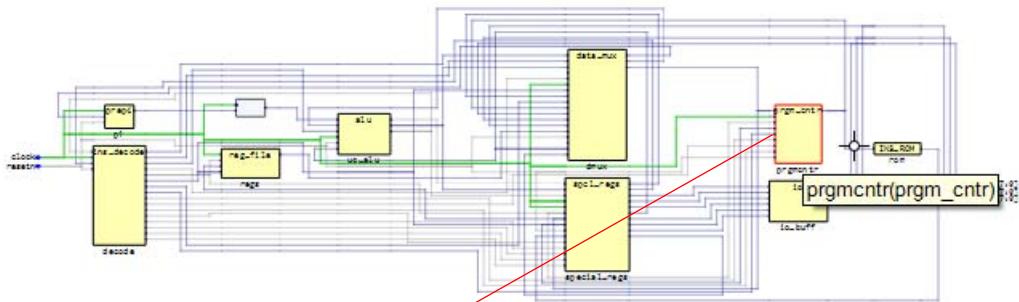
To crossprobe from the schematic to a source file when the source file is not open, the instance names must be the same. Notice that when you hover over an instance name in the schematic, it turns blue. You can click on this link, to automatically open the editor window of the source code file and highlight the appropriate code as shown below. A message is generated if a match cannot be found.



## Crossprobing to the Source Code

You can easily crossprobe instances or modules in the HDL Analyst view to the source code. To do this, choose either to:

- Highlight an instance in the HDL Analyst view, then right-click and select View Instance in Source from the drop-down menu. The tool automatically crossprobes to this instance in the source code.

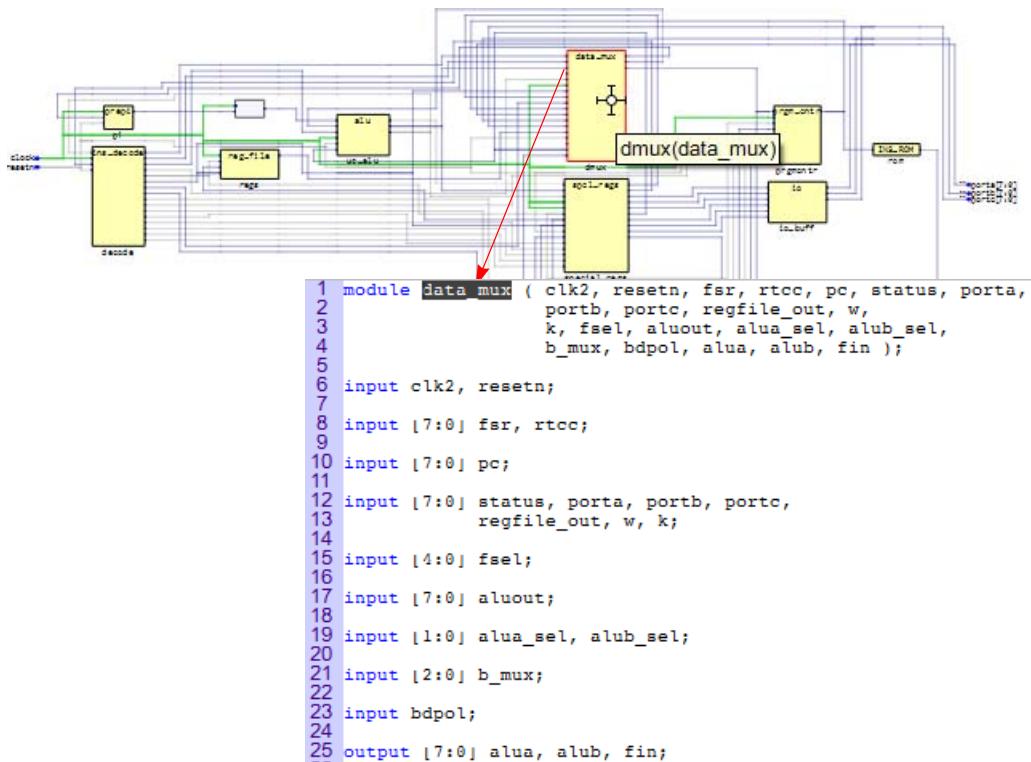


```

47 ins_decode decode (clk2, resetn, aluz, inst, f_we, w_we,
48                      status_z_we, status_c_we, tris_we,
49                      skip, k_fsel, longk, aluop,
50                      alua_sel, alub_sel, bdpol, opcode_goto, opcode_call, opcode_retlw );
51
52
53 // instantiating program counter block
54 prgm_cntr prgmcntr ( clk4, resetn, f_we, longk, fsel,
55                       opcode_goto, opcode_call, opcode_retlw, fin, pc );
56
57
58 // instantiating regs block
59 reg_file regs (clk1, f_we, fsel, fin, regfile_out);
60
61
62 // instantiating data mux block
63
64 data_mux dmux ( clk2, resetn, fsl, rtcc, pc[7:0], status, porta,
65                  portb, portc, regfile_out, w,
66                  k, fsel, aluout, alua_sel, alub_sel,
67                  b_mux, bdpol, alua, alub, fin );
68

```

- Highlight a module in the HDL Analyst view, then right-click and select View Module in Source from the drop-down menu. The tool automatically crossprobes to this module in the source code.



Note that you can crossprobe to instances and modules in the source code from the Hierarchy Browser as well. Highlight an object, then right-click and select View Instance in Source or View Module in Source from the drop-down menu.

## Crossprobing from the Text Editor Window

To crossprobe source in the text editor window or from the log file to a schematic, use this procedure. You can use this method to crossprobe from any text file with objects that have the same instance names as in the synthesis software.

1. Open the schematic to which you want to crossprobe.
2. Select the appropriate portion of the text in the Text Editor window. In some cases, it may be necessary to select an entire block of text to crossprobe.

3. You can choose either to:

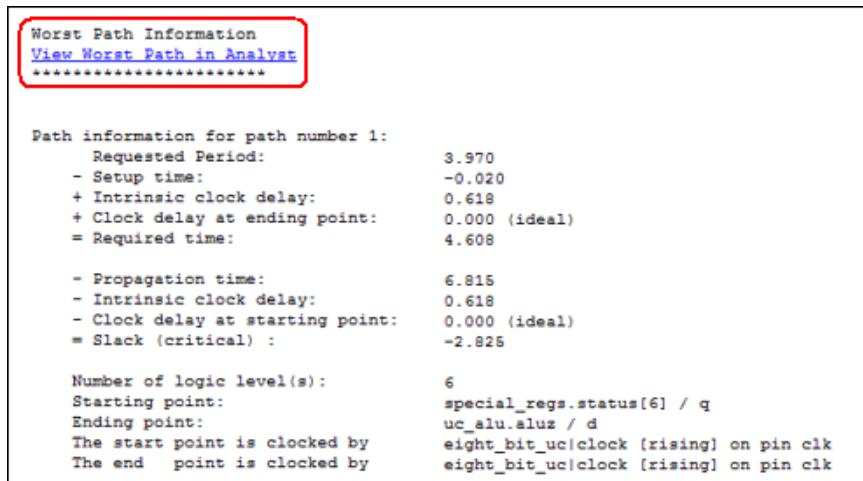
- Highlight the objects in the path, right-click and select Filter in Analyst from the drop-down menu. The tool automatically filters the schematic so that you see just the selected objects in the view.
- Highlight the objects in the path, right-click and select Select in Analyst from the drop-down menu. You might have to filter the selected objects to see them displayed in the schematic.

For example:

## Crossprobing from the Log File

The log file contains handy links, such as, clock trees driving clock pins of sequential elements and worst paths for the design to the HDL Analyst view. For example:

- Click on the View Worst Path in Analyst link in the log file.

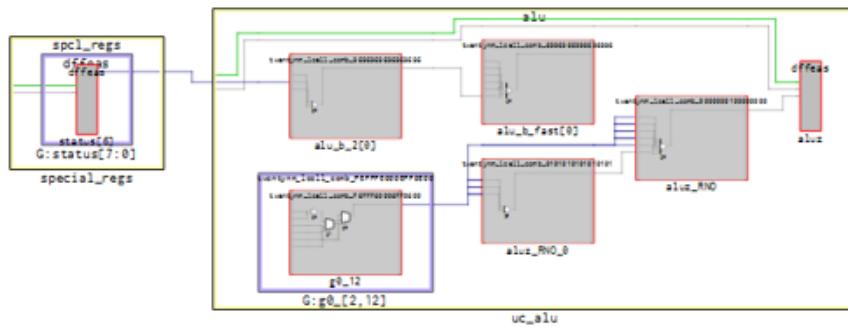


```
Worst Path Information
View Worst Path in Analyst
*****
Path information for path number 1:
  Requested Period:          3.970
  - Setup time:              -0.020
  + Intrinsic clock delay:   0.618
  + Clock delay at ending point: 0.000 (ideal)
  = Required time:           4.608

  - Propagation time:        6.815
  - Intrinsic clock delay:   0.618
  - Clock delay at starting point: 0.000 (ideal)
  = Slack (critical) :      -2.825

Number of logic level(s):      6
Starting point:               special_regs.status[6] / q
Ending point:                 uc_alu.aluz / d
The start point is clocked by eight_bit_uc|clock [rising] on pin clk
The end   point is clocked by eight_bit_uc|clock [rising] on pin clk
```

- The schematic for this critical path is automatically displayed in the mapped view shown below.



# Analyzing With the HDL Analyst Tool

The HDL Analyst tool is a graphical productivity tool that helps you visualize your synthesis results. It displays schematics of the design at different stages, allowing you to graphically view and analyze your design. At an early design stage, the schematic displays high-level structures like RAM, ROM, operators, and FSM as abstractions. Later in the cycle, these structures are converted to gates and mapped to technology-specific resources.

To analyze information or compare views with the log file, the FSM view, and the source code, you can use techniques like crossprobing, flattening, and filtering. See the following for more information about analysis techniques.

- [Viewing Design Hierarchy and Context, on page 245](#)
- [Filtering Schematics, on page 249](#)
- [Expanding Pin and Net Logic, on page 251](#)
- [Dissolving and Partial Dissolving of Buses and Pins, on page 255](#)
- [Flattening Schematic Hierarchy, on page 259](#)
- [Using the FSM Viewer, on page 261](#)

For additional information about navigating the HDL Analyst views or using other techniques like crossprobing, see the following:

- [Working in the Schematic, on page 198](#)
- [Exploring Design Hierarchy, on page 221](#)
- [Finding Objects, on page 227](#)
- [Crossprobing, on page 238](#)

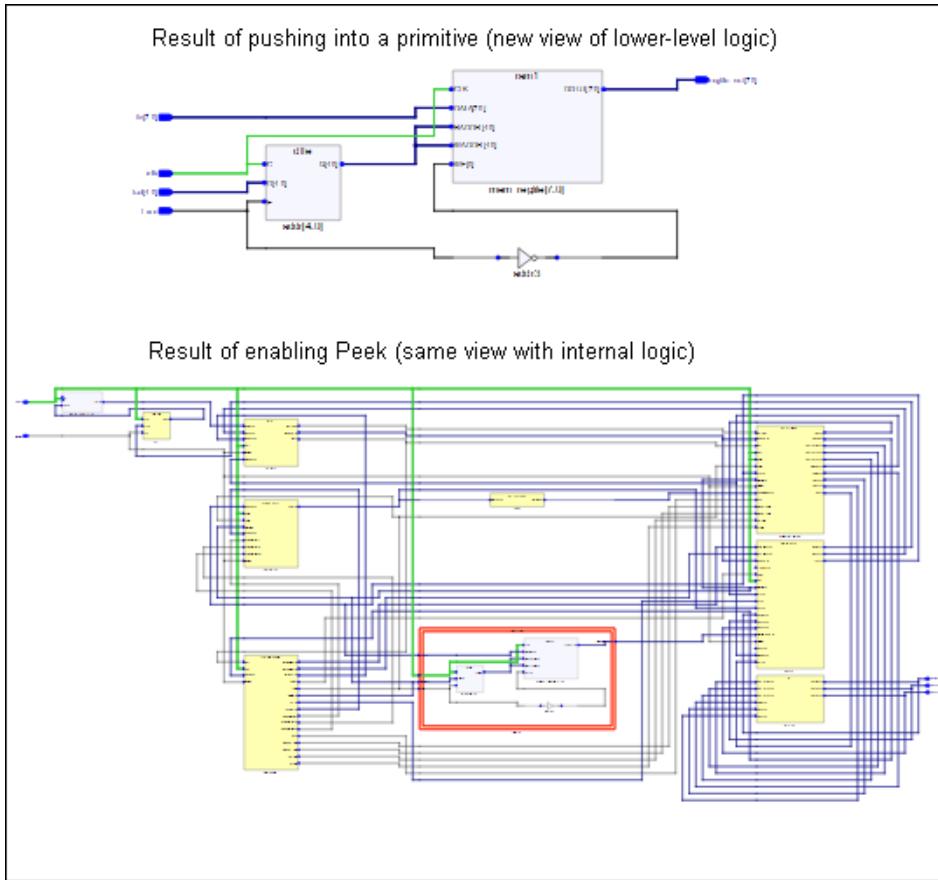
## Viewing Design Hierarchy and Context

Most large designs are hierarchical, so the software provides tools that help you view hierarchy details or put the details in context. Alternatively, you can browse and navigate hierarchy with the Push/Pop command, or flatten the design to view internal hierarchy.

This section describes how to use interactive hierarchical viewing operations to better analyze your design. Automatic hierarchy viewing operations that are built into other commands are described in the context in which they appear.

1. To view the internal logic of instances in your design, do either of the following:
  - To view the logic of an individual instance, push into it. This generates a new schematic with the internal details. Click the Back icon to return to the previous view.
  - To view the logic of all instances in the design, select all the required instances and right-click then select Peek. This command lets you see internal logic in content, by adding the internal details to the current schematic. If the view is too cluttered with this option on, filter the view (see [Filtering Schematics, on page 249](#)) or push into the primitive. Click the Back icon to return to the previous view after filtering or pushing into the object.

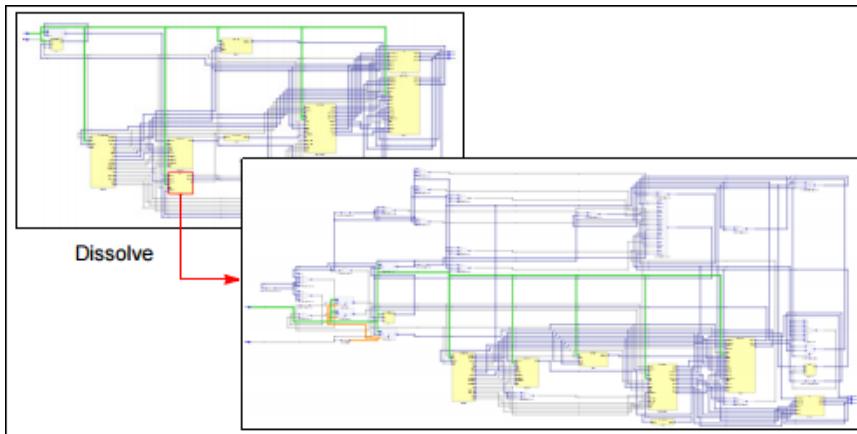
The following figure compares these two methods:



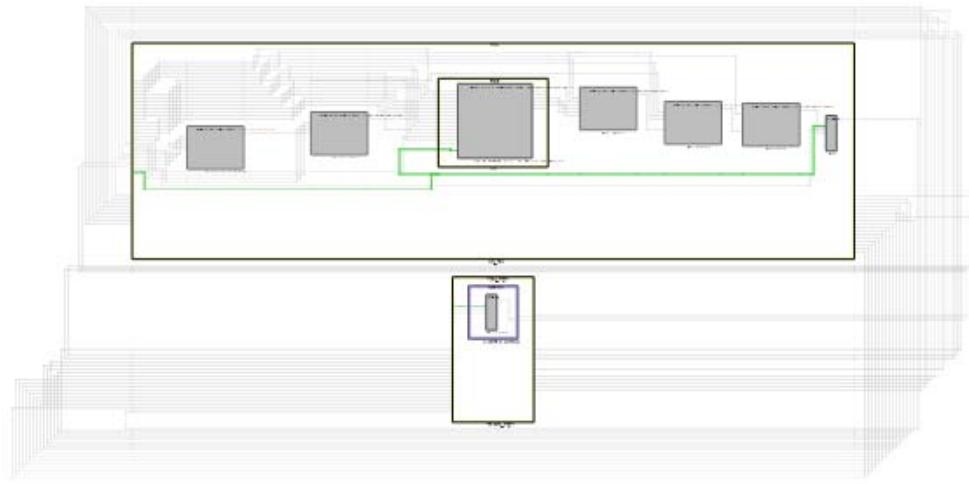
2. Suppose you just used the peek option to see the internal logic of an instance. To return back to the schematic state before using peek and while the peek objects are still highlighted, right-click and select Hide Contents from the drop-down menu.
3. To view the internal logic of a hierarchical instance, you can push into the instance, dissolve the selected instance with the Dissolve command, or flatten the design.

Pushing into an instance	Generates a view that shows only the internal logic. You do not see the internal hierarchy in context. To return to the previous view, click Back. See <a href="#">Exploring Object Hierarchy with Push/Pop Commands, on page 223</a> for details.
Flattening the entire design	Opens a view where the entire design is flattened. Large flattened designs can be overwhelming. See <a href="#">Flattening Schematic Hierarchy, on page 259</a> for details about flattening designs.
Flattening an instance by dissolving	Generates a view where the hierarchy of the selected instances is flattened, but the rest of the design is unaffected. This provides context. See <a href="#">Flattening Schematic Hierarchy, on page 259</a> for details about dissolving instances.

The following schematic shows an instance that has been dissolved in the view.



4. The software automatically traces a critical path through different hierarchical levels using hollow boxes with nested internal logic (transparent instances) to indicate levels in hierarchical instances.



## Filtering Schematics

Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter command to isolate selected objects.

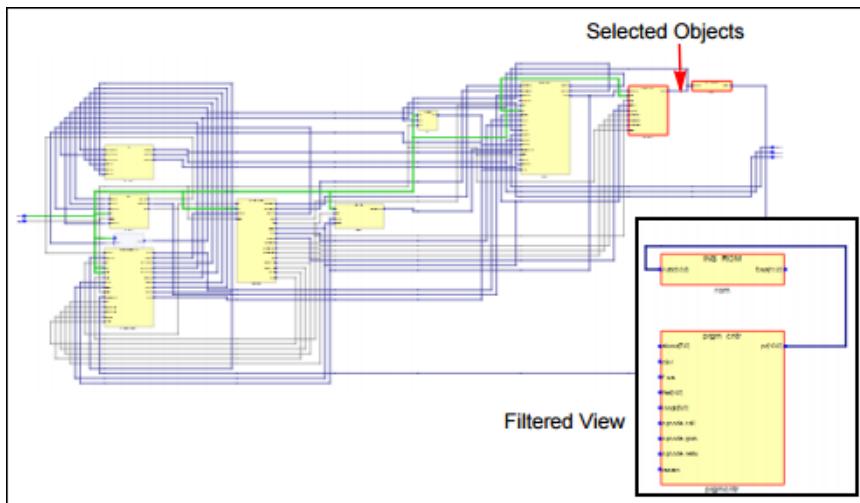
This table lists the advantages of using filtering over flattening:

Filter Schematic Command	Flatten Commands
Loads part of the design; better memory usage	Loads entire design
Combine filtering with the Push/Pop command, and history buttons (Back and Forward) to move freely between hierarchical levels	You can use the Back arrow or Show Top View icon to return to previous view that has been flattened.

1. Select the objects that you want to isolate. For example, you can select two connected objects.
2. Select the Filter command, using one of these methods:

- Right-click and select Filter from the popup menu.
- Click the Filter icon (buffer gate) (  ).

The software filters the design and displays the selected objects in a filtered view. These objects are isolated in the schematic displayed. Select Unfilter to take you back to the view where objects are at the same level.



You can now analyze the problem, and do operations like the following:

Trace paths, build up logic See [Expanding Pin and Net Logic, on page 251](#)

Filter further Select objects and filter again

Find objects See [Finding Objects, on page 227](#)

Flatten See [Flattening Schematic Hierarchy, on page 259](#). You can hide transparent or opaque instances.

Crossprobe from filtered view See [Crossprobing from an HDL Analyst View, on page 239](#)

3. To return to the previous schematic, click the Back arrow. If you flattened the hierarchy, right-click and select the Back arrow or the Show Top View icon to return to the top-level unflattened view.

For additional information about filtering schematics, see [Filtering Schematics, on page 249](#) and [Flattening Schematic Hierarchy, on page 259](#).

## Expanding Pin and Net Logic

When you are working in a filtered view, you might need to include more logic in your selected set to debug your design.

Use the Expand commands with the Filter and Flatten commands to isolate just the logic that you want to examine. Filtering isolates logic, flattening removes hierarchy. See [Filtering Schematics, on page 249](#) and [Flattening Schematic Hierarchy, on page 259](#) for details.

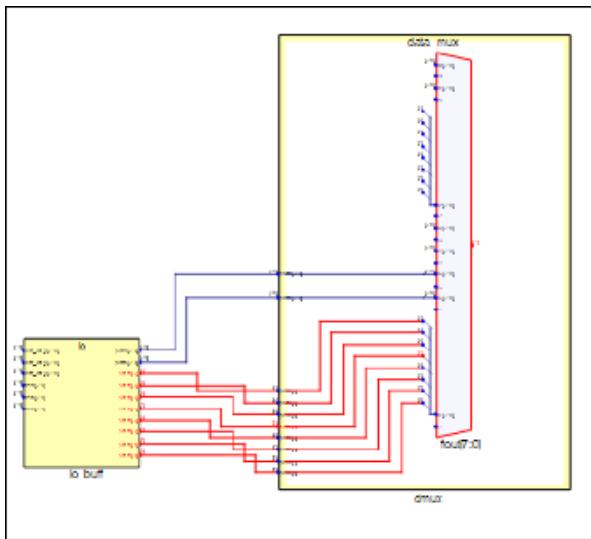
1. To expand logic from a pin hierarchically across boundaries, use the following commands.

To ...	Do this ...
See the first-level cells connected to a pin in the same hierarchy	Select a pin and select Expand. See <a href="#">Expanding Filtered Logic Example, on page 251</a> .
See the first-level cells connected to a pin at any level of hierarchy	Select a pin and select Hierarchical Expand Nets.
See all cells until a register or port is connected to the selected pin at the same level of hierarchy	Select a pin and select Expand to Reg/Port.
See all cells until a register or port is connected to the selected pin at any level of hierarchy	Select a pin and select Hierarchical Expand to Reg/Port.
See internal cells connected to a pin	Select a pin and select Expand Inwards. The software filters the schematic and displays the internal cells closest to the port. See <a href="#">Expanding Inwards Example, on page 251</a> .

### Expanding Filtered Logic Example

### Expanding Inwards Example

## Expand to One Object



## Expanding Hierarchically

2. To expand logic from a net, do the following:
  - Use the commands shown in the following table.
  - Select a net, then right-click and select the command from the right-click options.

---

To ...

Do this ...

See all instances connected to the selected net being filtered

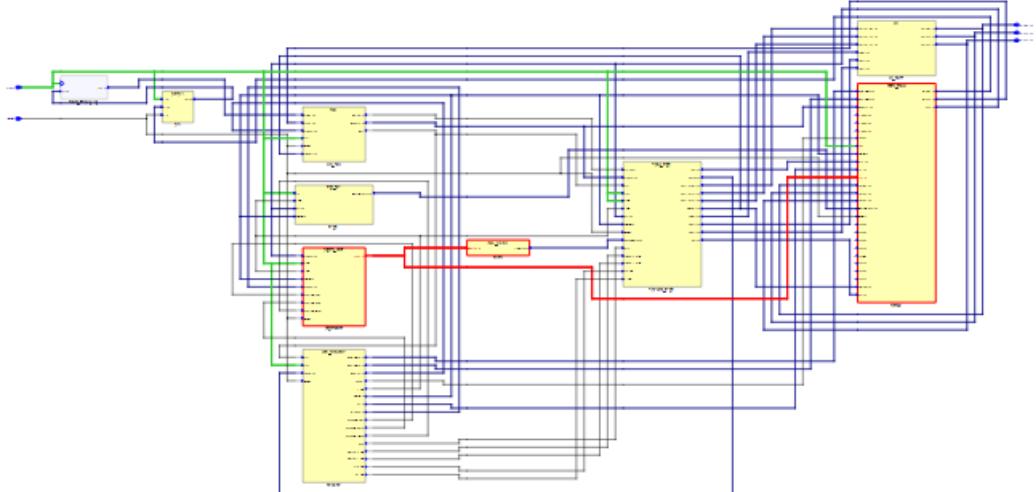
Select a net and select Filter by Nets.

To ...	Do this ...
Select the instances in the same hierarchy connected to the selected net	Select a net and select Expand Nets.
Select and show instances connected to the selected net at any level of hierarchy. The instance that drives the net and the instance which is driven by the net are shown.	Select a net and select Hierarchical Expand Nets.
Select and show instances connected to the selected net at any level of hierarchy. Instances that are not connected are removed from the view.	Select a net, then Filter by Net, and select Hierarchical Expand Nets.

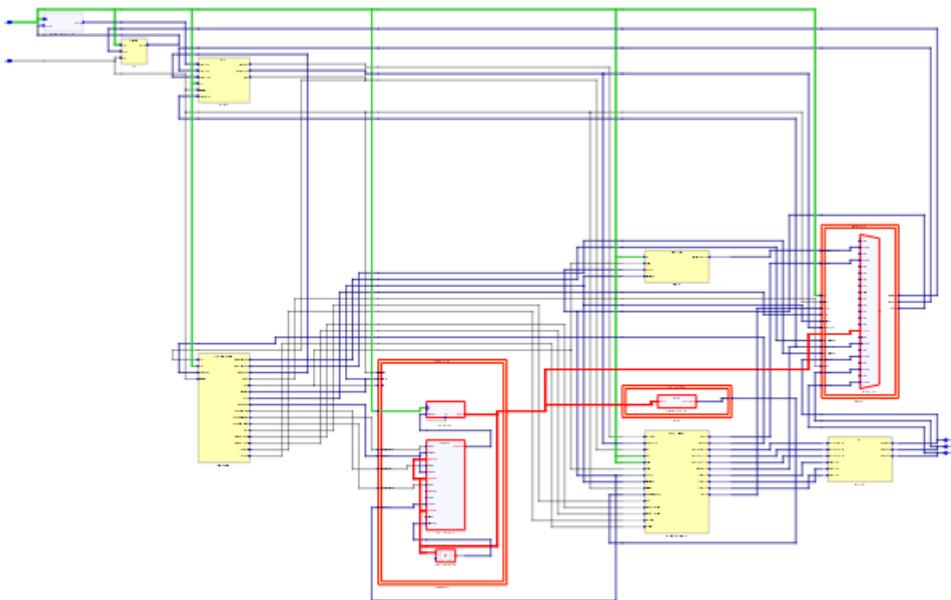
The following figures illustrate this.

## Filter by Nets

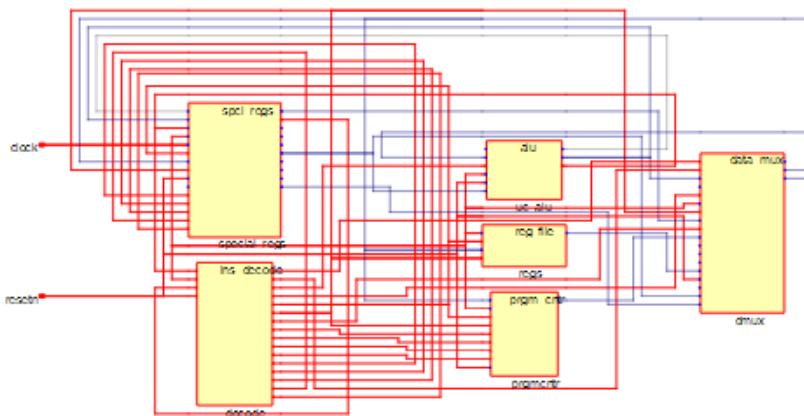
### Expand Nets



## Hierarchical Expand Nets



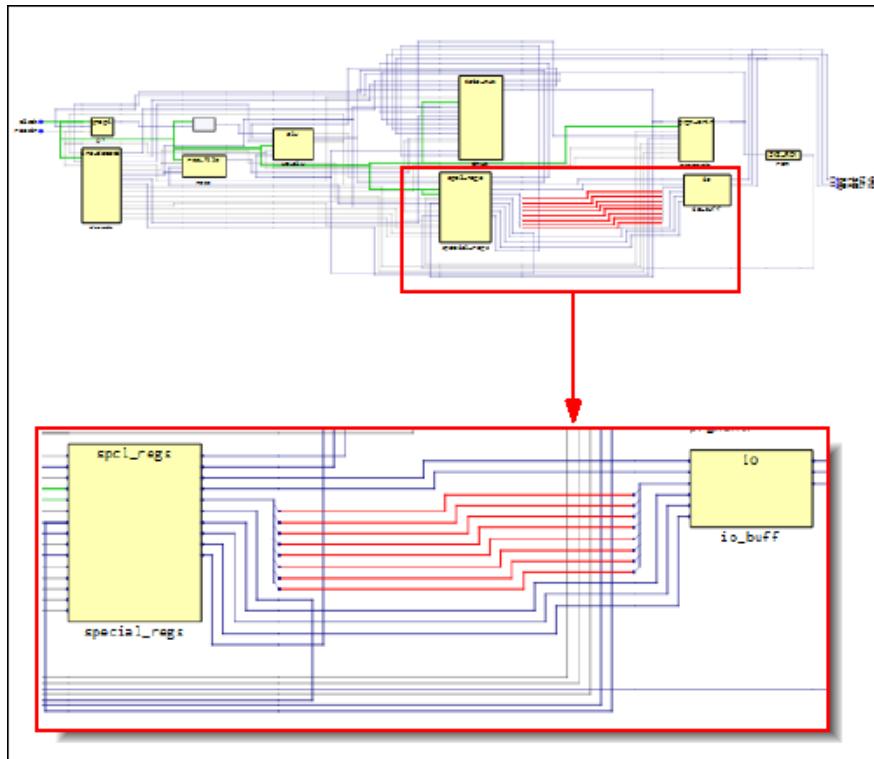
3. You can also isolate the paths to generate a schematic for a path between objects. To display connections to and from the selected instance, highlight it then right-click and select Isolate Paths from the drop-down menu.



## Dissolving and Partial Dissolving of Buses and Pins

The HDL Analyst tool has options for handling buses and pins in the display that can help you analyze your design easier. You can expand logic from a bus port or specific bits of a port.

1. To expand logic from a bus port to its bit ports:
  - Select a bus.
  - Right-click and select Dissolve from the drop-down menu.
  - Filter by net and choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 251](#)

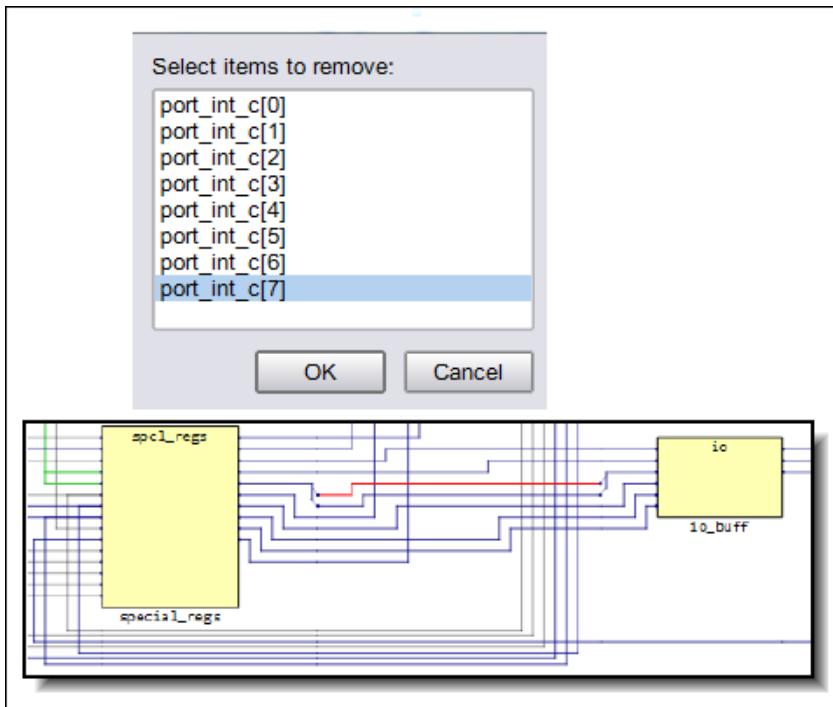


2. To expand logic from nets of a bus:
  - Select a bus.
  - Right-click and select Partial Dissolve from the drop-down menu.

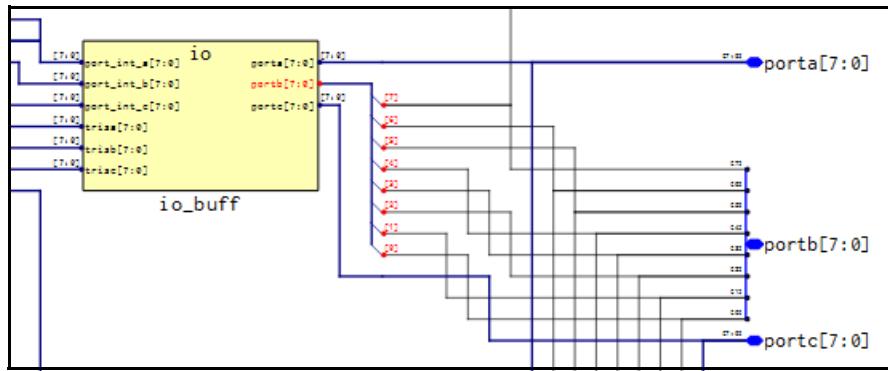
- Select the net (`port_int_c[7]`) to be removed.
- Click OK.

The selected net is now removed from the bus.

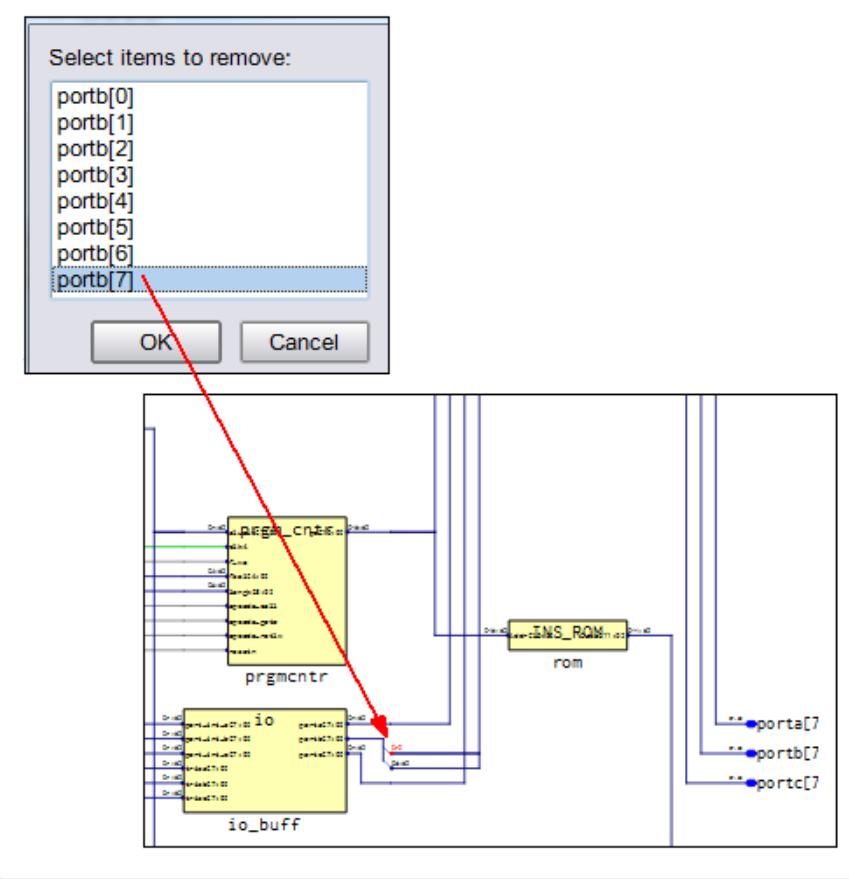
- Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 318](#).



3. To expand logic for all the pins of a bus pin:
  - Select a bus pin.
  - Right-click and select Dissolve from the drop-down menu.
  - Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 318](#).



4. To expand logic from specific bits pins of a bus pin:
  - Select a bus pin.
  - Right-click and select Partial Dissolve Pin from the drop-down menu.
  - Select the pin to be removed.
  - Click OK.  
The selected pin is removed from the bus pin.
  - Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 318](#).



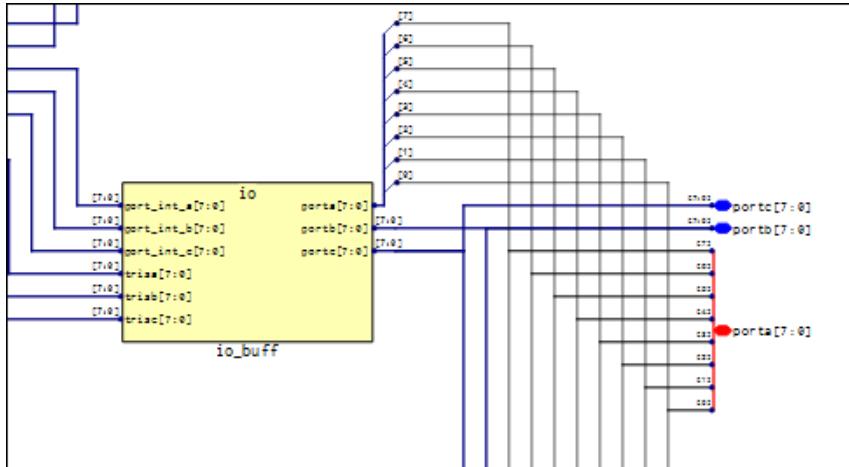
## Dissolving of Ports

The HDL Analyst tool has options for handling ports in the display that can help you analyze your design easier. You can expand logic for all bits of a port.

To expand logic for a port:

- Select a port.
- Right-click and select Dissolve from the drop-down menu.

- Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 251](#).



## Flattening Schematic Hierarchy

Flattening removes hierarchy so you can view the logic without hierarchical levels. In most cases, you do not have to flatten your hierarchical schematic to debug and analyze your design, because you can use a combination of filtering and expanding to view logic at different levels. However, if you must flatten the design use the following techniques, which include flattening and dissolving instances.

1. To flatten any level of hierarchy to logic cells below the current level, right-click and select Flatten Schematic from the drop-down menu.

The software flattens the design hierarchy and displays it in the window. To return to the previous level, select the Back arrow.

2. To selectively flatten some hierarchical instances in your design by dissolving them, do the following:
  - Select the instances to be flattened.
  - Right-click and select Dissolve.

The results differ slightly, depending on the kind of view from which you dissolve instances.

**Starting View    Software Generates a ...**

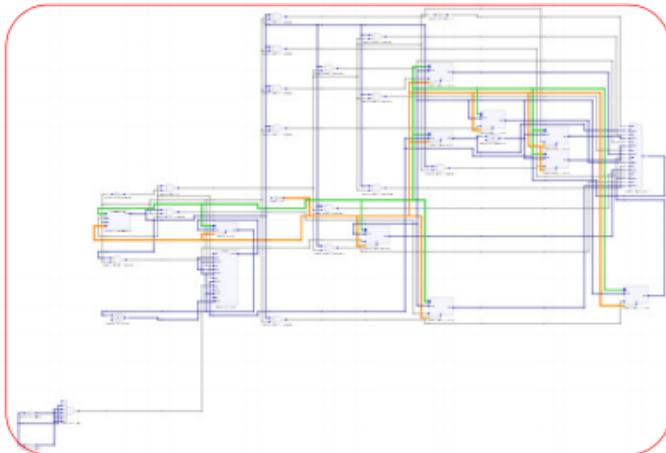
---

Filtered	Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of <a href="#">Viewing Design Hierarchy and Context, on page 245</a> . Use the Back button to return to the undissolved view.
Unfiltered	New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. You can use the Back button to return to previous or the top-level views.

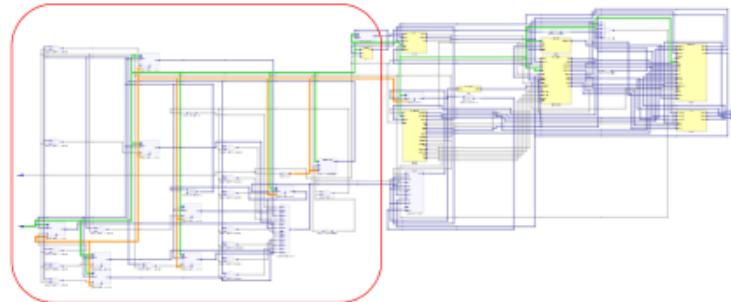
---

The following figure illustrates this.

Dissolved logic for prgmcntr shown filtered when started from filtered view



Dissolved logic for prgmcntr shown flattened in context when you start from an unfiltered view



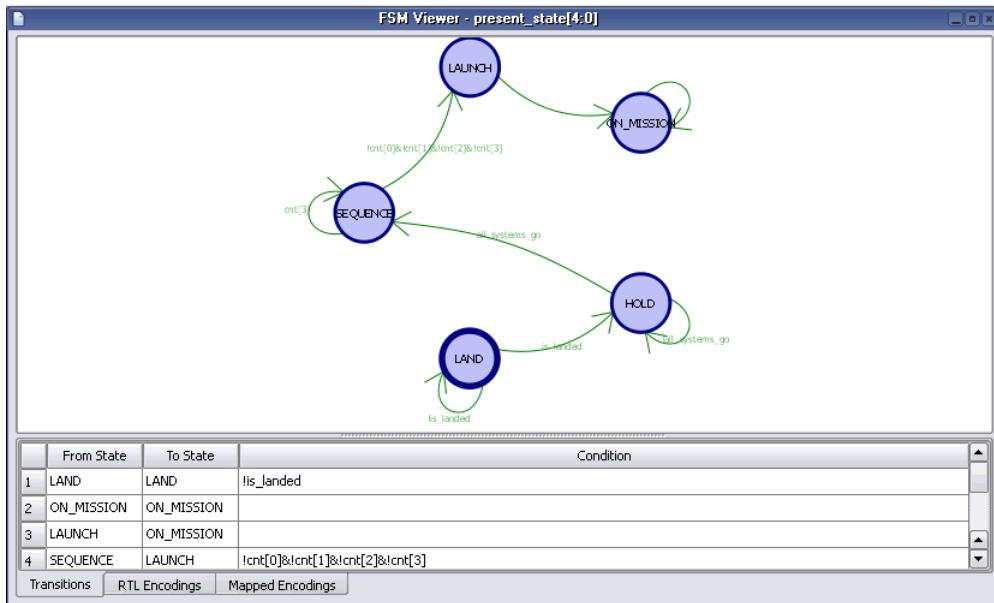
Use this technique if you only want to flatten part of your design while retaining the hierarchical context. If you want to flatten most of the design, use the technique described in the previous step. Instead of dissolving instances, you can use a combination of the filtering commands and the Push/Pop command.

## Using the FSM Viewer

The FSM viewer displays state transition bubble diagrams for FSMs in the design, along with additional information about the FSM. You can use this viewer to view state machines.

- To start the FSM viewer, open the compiled view and highlight the FSM instance, click the right mouse button and select View State Machine from the popup menu.

The FSM viewer opens. The viewer consists of a transition bubble diagram and a table for the encodings and transitions. If you used Verilog to define the FSMs, the viewer displays binary values for the state machines if you defined them with the 'define' keyword, and actual names if you used the parameter keyword.



- The following table summarizes basic viewing operations.

#### To view ...

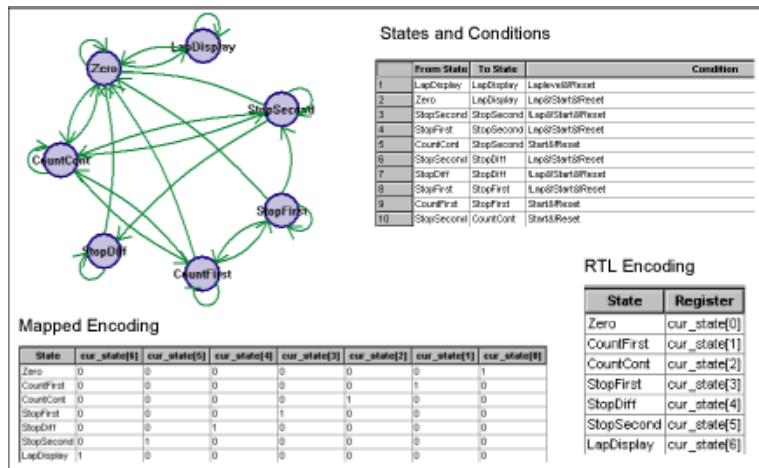
from and to states, and conditions for each transition

#### Do ...

Click the Transitions tab at the bottom of the table.

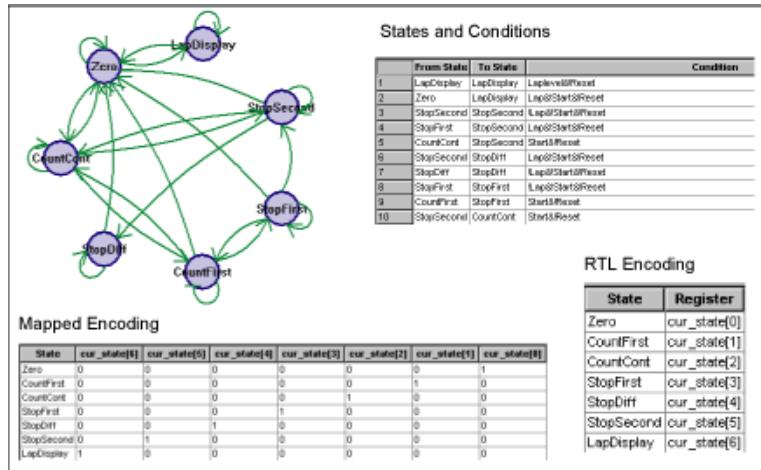
To view ...	Do ...
the correspondence between the states and the FSM registers in the RTL view	Click the RTL Encoding tab.
the correspondence between the states and the registers in the Technology View	Click the Mapped Encodings tab (available after synthesis).
only the transition diagram without the table	Select View->FSM table or click the FSM Table icon. You might have to scroll to the right to see it.

This figure shows you the mapping information for a state machine. The Transitions tab shows you simple equations for conditions for each state. The RTL Encodings tab has a State column that shows the state names in the source code, and a Registers column for the corresponding RTL encoding. The Mapped Encoding tab shows the state names in the code mapped to actual values.



3. To view just one selected state,
  - Select the state by clicking on its bubble. The state is highlighted.
  - Click the right mouse button and select the filtering criteria from the popup menu: output, input, or any transition.

The transition diagram now shows only the filtered states you set. The following figure shows filtered views for output and input transitions for one state.



Similarly, you can check the relationship between two or more states by selecting the states, filtering them, and checking their properties.

4. To view the properties for a state,
  - Select the state.
  - Click the right mouse button and select **Properties** from the popup menu. A form shows you the properties for that state.

To view the properties for the entire state machine like encoding style, number of states, and total number of transitions between states, deselect any selected states, click the right mouse button outside the diagram area, and select **Properties** from the popup menu.

# Working in the Standard Schematic

The HDL Analyst includes the RTL and Technology views, which are schematics used to graphically analyze your design. The RTL view is available after a design is compiled; the Technology view is available after a design has been synthesized and contains technology-specific primitives.

For detailed descriptions of these views, see the *HDL Analyst Tool* section of the *Reference Manual*. This section describes basic procedures you use in the RTL and Technology views. The information is organized into these topics:

- [Differentiating Between the HDL Analyst Views](#), on page 266
- [Opening the Views](#), on page 266
- [Viewing Object Properties](#), on page 267
- [Selecting Objects in the RTL/Technology Views](#), on page 273
- [Working with Multisheet Schematics](#), on page 274
- [Moving Between Views in a Schematic Window](#), on page 275
- [Setting Schematic Preferences](#), on page 276
- [Managing Windows](#), on page 278

For information on specific tasks like analyzing critical paths, see the following sections:

- [Exploring Object Hierarchy by Pushing/Popping](#), on page 281
- [Exploring Object Hierarchy of Transparent Instances](#), on page 286
- [Browsing to Find Objects in HDL Analyst Views](#), on page 288
- [Crossprobing \(Standard\)](#), on page 303
- [Analyzing With the Standard HDL Analyst Tool](#), on page 311

## Differentiating Between the HDL Analyst Views

RTL View	Technology View
Generated after compilation	Generated after mapping
Technology-independent components at a high level of abstraction, like adders, registers, large muxes, and state machines	Technology-specific primitives like look-up tables, cascade and carry chains, muxes and flip-flops
srs database (Synopsys proprietary)	srm database (Synopsys proprietary)

## Opening the Views

The procedure for opening an RTL or Technology view is similar; the main difference is the design stage at which these views are available.

1. Start at the appropriate design stage:

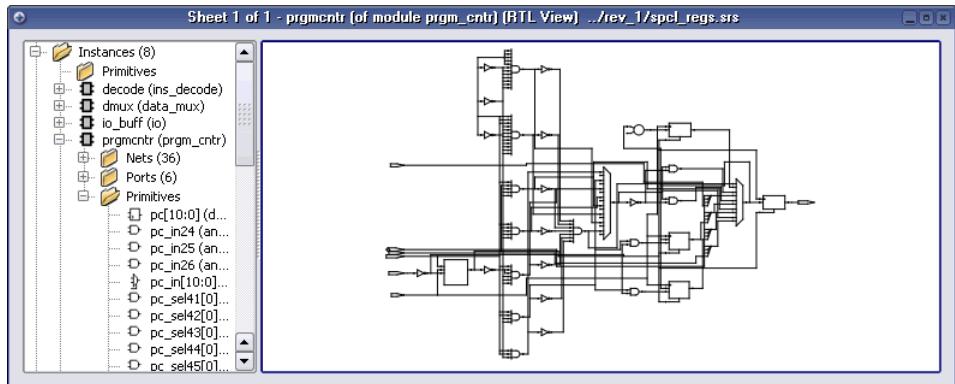
RTL view	Start with a compiled design.
Technology view	Start with a mapped (synthesized) design.

2. Open the view as described in this table:

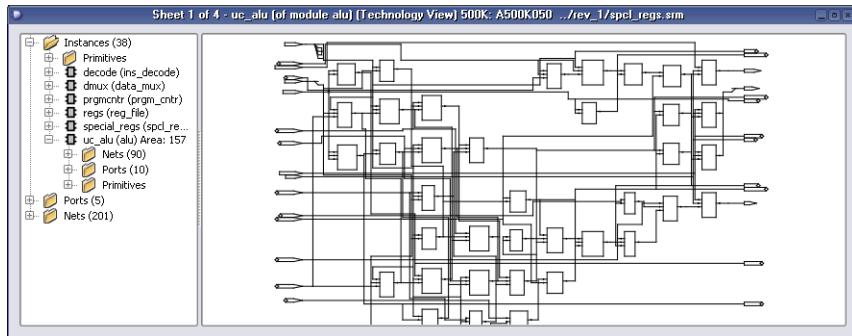
Hierarchical RTL view	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst-&gt;RTL-&gt;Hierarchical View.</li> <li>• Click the RTL View icon ().</li> <li>• Double-click the srs file in the Implementation Results view.</li> </ul> <p>To open a flattened RTL view, select HDL Analyst-&gt;RTL-&gt;Flattened View.</p>
Hierarchical Technology view	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst -&gt;Technology-&gt;Hierarchical View.</li> <li>• Click the Technology View icon ().</li> <li>• Double-click the srm file in the Implementation Results view.</li> </ul>
Flattened RTL or Technology view	Select HDL Analyst->RTL->Flattened View or HDL Analyst-> Technology->Flattened View

All RTL and Technology views have the schematic on the right and a pane on the left that contains a hierarchical list of the objects in the design. This pane is called the Hierarchy Browser. The bar at the top of contains additional information. in the *Reference Manual*

#### RTL View



#### Technology View



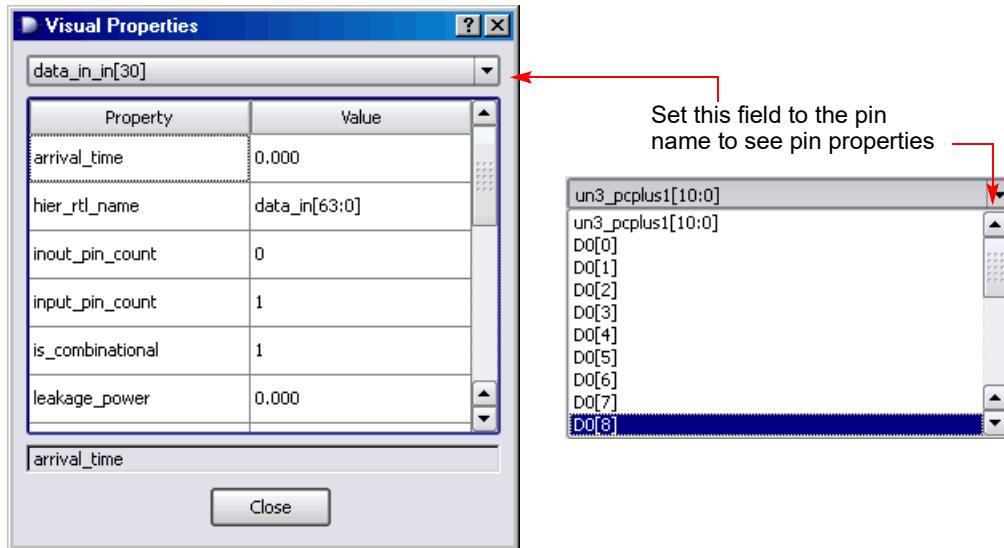
## Viewing Object Properties

There are a few ways in which you can view the properties of objects.

1. To temporarily display the properties of a particular object, hold the cursor over the object. A tooltip temporarily displays the information at the cursor and in the status bar at the bottom of the tool window.

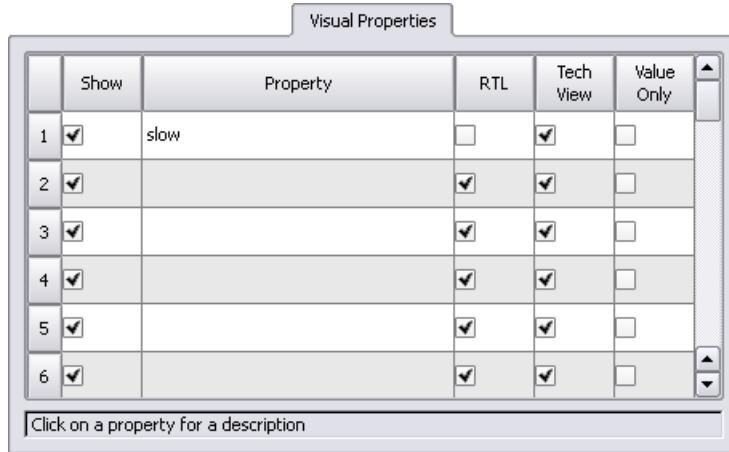
2. Select the object, right-click, and select Properties. The properties and their values are displayed in a table.

If you select an instance, you can view the properties of the associated pins by selecting the pin from the list. Similarly, if you select a port, you can view the properties on individual bits.



3. To flag objects by property, follow these steps:

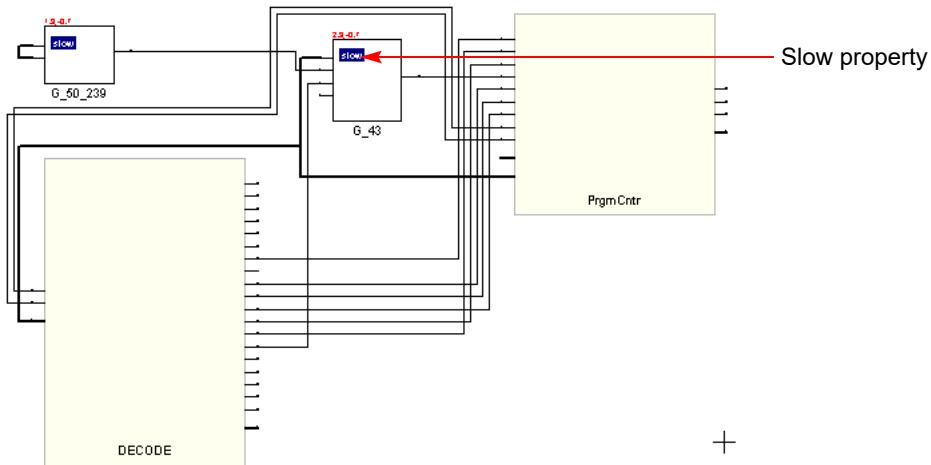
- Open an RTL or Technology view.
- Select Options->HDL Analyst Options->Visual Properties, and select the properties you want to view from the pull-down list. Some properties are only available in certain views.



- Close the HDL Analyst Options dialog box.
- Enable View->Visual Properties. If you do not enable this, the software does not display the property flags in the schematics. The tool uses a rectangular flag with the property name and value to annotate all objects in the current view that have the specified property. Different properties use different colors, so you can enable and view many properties at the same time.

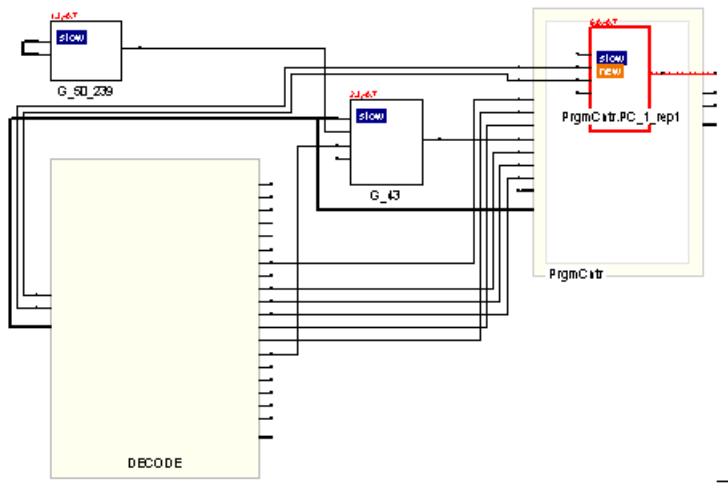
## Example: Slow and New Properties

The slow property is useful for analyzing your critical path, because it denotes objects that do not meet the timing criteria. The following figure shows a filtered view of a critical path, with slow instances flagged in blue.



The New property helps with debugging because it quickly identifies objects that have been added to the current schematic with commands like Expand. You can step through successive filtered views to determine what was added at each step.

The next figure expands one of the pins from the previous filtered view. The new instance added to the view has two flags: new and slow.



## Using the orig\_inst\_of Property for Parameterized Modules

The compiler automatically unifies parameterized modules or instances. Properties are available to identify the RTL names of both unified and original modules or instances.

- inst\_of property – identifies module or instance by unified name
- orig\_inst\_of property – identifies module or instance by its original name before it was unified

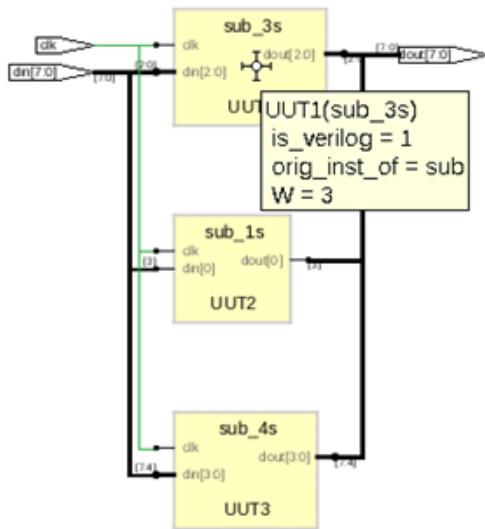
In the following example, the top-level module (top) instantiates the module sub multiple times using different parameter values. The compiler unifies the module sub as sub\_3s, sub\_1s, and sub\_4s.

### Top.v

```
module top (input clk, [7:0] din, output [7:0] dout);
    sub #(.(W(3))) UUT1 (.clk, .din(din[2:0]), .dout(dout[2:0]));
    sub #(.(W(1))) UUT2 (.clk, .din(din[3]), .dout(dout[3]));
    sub #(.(W(4))) UUT3 (.clk, .din(din[7:4]), .dout(dout[7:4]));
endmodule

module sub #(parameter W = 0) (
    input clk,
    input [W-1:0] din,
    output logic [W-1:0] dout );
    always@(posedge clk)
        begin
            dout <= din;
        end
endmodule
```

### RTL View



### TCL Command Example

Use the `get_prop` command with the `orig_inst_of` property to identify the original RTL name for the module:

```
% get_prop -prop orig_inst_of {v:sub_3s}  
sub  
  
% get_prop -prop orig_inst_of {i:UUT3}  
sub
```

## Selecting Objects in the RTL/Technology Views

For mouse selection, standard object selection rules apply. In selection mode, the pointer is shaped like a crosshair.

To select...	Do this...
Single objects	Click on the object in the RTL or Technology schematic, or click the object name in the Hierarchy Browser.
Multiple objects	<p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>• Draw a rectangle around the objects.</li> <li>• Select an object, press Ctrl, and click other objects you want to select.</li> <li>• Select multiple objects in the Hierarchy Browser. See <a href="#">Browsing With the Hierarchy Browser, on page 288</a>.</li> <li>• Use Find to select the objects you want. See <a href="#">Using Find for Hierarchical and Restricted Searches, on page 291</a>.</li> </ul>
Objects by type (instances, ports, nets)	Use Edit->Find to select the objects (see <a href="#">Browsing With the Find Command, on page 289</a> ), or use the Hierarchy Browser, which lists objects by type.
All objects of a certain type (instances, ports, nets)	To select all objects of a certain type, do either of the following: <ul style="list-style-type: none"> <li>• Right-click and choose the appropriate command from the Select All Schematic/Current Sheet popup menus.</li> <li>• Select the objects in the Hierarchy Browser.</li> </ul>
No objects (deselect all currently selected objects)	Click the left mouse button in a blank area of the schematic or click the right mouse button to bring up the pop-up menu and choose Unselect All. Deselected objects are no longer highlighted.

The HDL Analyst view highlights selected objects in red. If the object you select is on another sheet of the schematic, the schematic tracks to the appropriate sheet. If you have other windows open, the selected object is highlighted in the other windows as well (crossprobing), but the other windows do not track to the correct sheet. Selected nets that span different hierarchical levels are highlighted on all the levels. See [Crossprobing \(Standard\), on page 303](#) for more information about crossprobing.

Some commands affect selection by adding to the selected set of objects: the Expand commands, the Select All commands, and the Select Net Driver and Select Net Instances commands.

## Working with Multisheet Schematics

The title bar of the RTL or Technology view indicates the number of sheets in that schematic. In a multisheet schematic, nets that span multiple sheets are indicated by sheet connector symbols, which you can use for navigation.

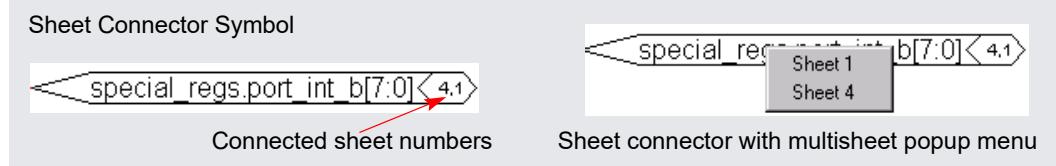
1. To reduce the number of sheets in a schematic, select Options->HDL Analyst Options and increase the values set for Sheet Size Options - Instances and Sheet Size Options - Filtered Instances. To display fewer objects per sheet (increase the number of sheets), increase the values.

These options set a limit on the number of objects displayed on an unfiltered and filtered schematic sheet, respectively. A low Filtered Instances value can cause lower-level logic inside a transparent instance to be displayed on a separate sheet. The sheet numbers are indicated inside the empty transparent instance.

2. To navigate through a multisheet schematic, refer to this table. It summarizes common operations and ways to navigate.

To view...	Use one of these methods...
Next sheet or previous sheet	Select View->Next/Previous Sheet. Press the right mouse button and draw a horizontal mouse stroke (left to right for next sheet, right to left for previous sheet). Click the icons: Next Sheet (  ) or Previous Sheet (  ). Press Shift-right arrow (Next Sheet) or Shift-left arrow (Previous sheet). Navigate with View->Back and View ->Forward if the next/previous sheets are part of the display history.
A specific sheet number	Select View->View Sheets and select the sheet. Click the right mouse button, select View Sheets from the popup menu, and then select the sheet you want. Press Ctrl-g and select the sheet you want.
Lower-level logic of a transparent instance on separate sheets	Check the sheet numbers indicated inside the empty transparent instance. Use the sheet navigation commands like Next Sheet or View Sheets to move to the sheet you need.

To view...	Use one of these methods...
All objects of a certain type	To highlight all the objects of the same type in the schematic, right-click and select the appropriate command from the Select All Schematic popup menu. To highlight all the objects of the same type on the current sheet, right-click and select the appropriate command from the Select All Sheet popup menu.
Selected items only	Filter the schematic as described in <a href="#">Filtering Schematics, on page 315</a> .
A net across sheets	If there are no sheet numbers displayed in a hexagon at the end of the sheet connector, select Options ->HDL Analyst Options and enable Show Sheet Connector Index. Right-click the sheet connector and select the sheet number from the popup as shown in the following figure.

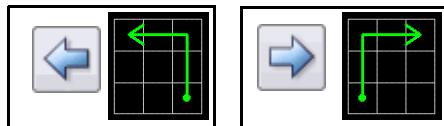


## Moving Between Views in a Schematic Window

When you filter or expand your design, you move through a number of different design views in the same schematic window. For example, you might start with a view of the entire design, zoom in on an area, then filter an object, and finally expand a connection in the filtered view, for a total of four views.

1. To move back to the previous view, click the Back icon or draw the appropriate mouse stroke.

The software displays the last view, including the zoom factor. This does not work in a newly generated view (for example, after flattening) because there is no history.



2. To move forward again, click the Forward icon or draw the appropriate mouse stroke.

The software displays the next view in the display history.

## Setting Schematic Preferences

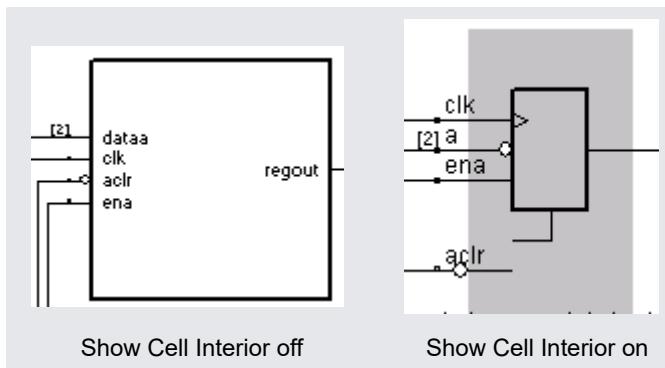
You can set various preferences for the RTL and Technology views from the user interface.

1. Select Options->HDL Analyst Options. For a description of all the options on this form, see [HDL Analyst Options Command, on page 395](#) in the *Reference Manual*.
2. The following table details some common operations:

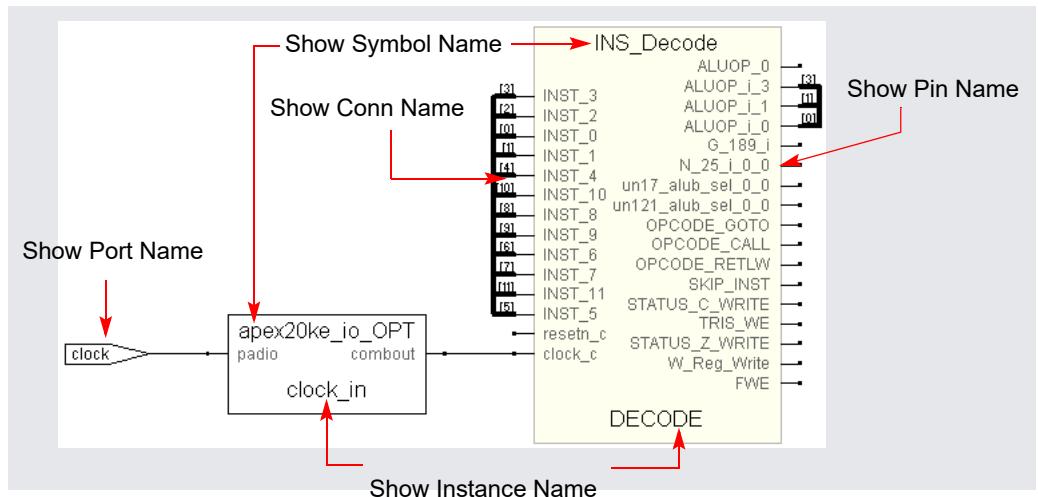
To...	Do this...
Display the Hierarchy Browser	Enable Show Hierarchy Browser (General tab).
Control crossprobing from an object to a P&R text file	Enable Enhanced Text Crossprobing. (General tab)
Determine the number of objects displayed on a sheet.	Set the value with Maximum Instances on the Sheet Size tab. Increase the value to display more objects per sheet.
Determine the number of objects displayed on a sheet in a filtered view.	Set the value with Maximum Filtered Instances on the Sheet Size tab. Increase the number to display more objects per sheet. You cannot set this option to a value less than the Maximum Instances value.

Some of these options do not take effect in the current view, but are visible in the next schematic you open.

3. To view hierarchy within a cell, enable the General->Show Cell Interiors option.



- To control the display of labels, first enable the Text->Show Text option, and then enable the Label Options you want. The following figure illustrates the label that each option controls.



For a more detailed information about some of these options, see [Schematic Display, on page 93](#) in the *Reference Manual*.

- Click OK on the HDL Analyst Options form.

The software writes the preferences you set to the ini file, and they remain in effect until you change them.

## Managing Windows

As you work on a project, you open different windows. For example, you might have two Technology views, an RTL view, and a source code window open. The following guidelines help you manage the different windows you have open. For information about cycling through the display history in a single schematic, see *Moving Between Views in a Schematic Window, on page 275*.

1. Toggle on View->Workbook Mode.

Below the Project view, you see tabs like the following for each open view. The tab for the current view is on top. The symbols in front of the view name on the tab help identify the kind of view.



2. To bring an open view to the front, if the window is not visible, click its tab. If part of the window is visible, click in any part of the window.

If you previously minimized the view, it will be in minimized form. Double-click the minimized view to open it.

3. To bring the next view to the front, click Ctrl-F6 in that window.
4. Order the display of open views with the commands from the Window menu. You can cascade the views (stack them, slightly offset), or tile them horizontally or vertically.
5. To close a view, press Ctrl-F4 in that window or select File->Close.

# Exploring Design Hierarchy (Standard)

Schematics generally have a certain amount of design hierarchy. You can move between hierarchical levels using the Hierarchy Browser or Push/Pop mode. For additional information, see [Analyzing With the Standard HDL Analyst Tool, on page 311](#). The topics include:

- [Traversing Design Hierarchy with the Hierarchy Browser, on page 280](#)
- [Exploring Object Hierarchy by Pushing/Popping, on page 281](#)
- [Exploring Object Hierarchy of Transparent Instances, on page 286](#)

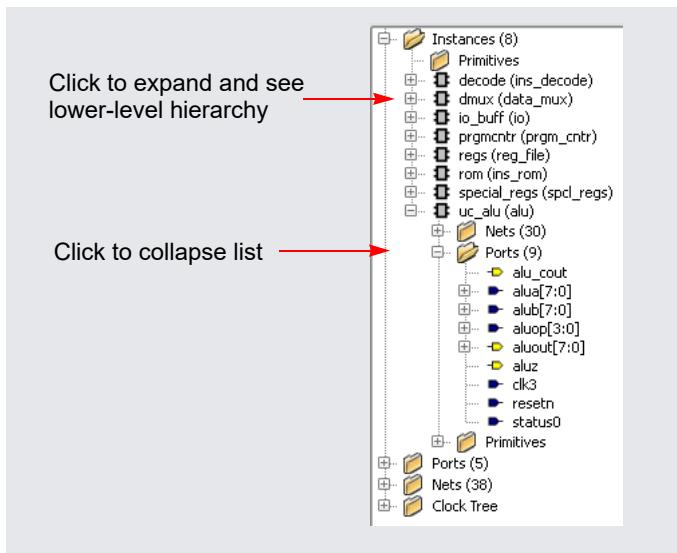
## Traversing Design Hierarchy with the Hierarchy Browser

The Hierarchy Browser is the list of objects on the left side of the RTL and Technology views. It is best used to get an overview, or when you need to browse and find an object. If you want to move between design levels of a particular object, Push/Pop mode is more direct. Refer to [Exploring Object Hierarchy by Pushing/Popping, on page 281](#) for details.

The hierarchy browser allows you to traverse and select the following:

- Instances and submodules
- Ports
- Internal nets
- Clock trees (in an RTL view)

The browser lists the objects by type. A plus sign in a square icon indicates that there is hierarchy under that object and a minus sign indicates that the design hierarchy has been expanded. To see lower-level hierarchy, click the plus sign for the object. To ascend the hierarchy, click the minus sign.



Refer to [Hierarchy Browser Symbols](#), on page 82 in the *Reference Manual* for an explanation of the symbols.

## Exploring Object Hierarchy by Pushing/Popping

To view the internal hierarchy of a specific object, it is best to use Push/Pop mode or examine transparent instances, instead of using the Hierarchy Browser described in [Traversing Design Hierarchy with the Hierarchy Browser, on page 280](#). You can access Push/Pop mode with the Push/Pop Hierarchy icon, the Push/Pop Hierarchy command, or mouse strokes.

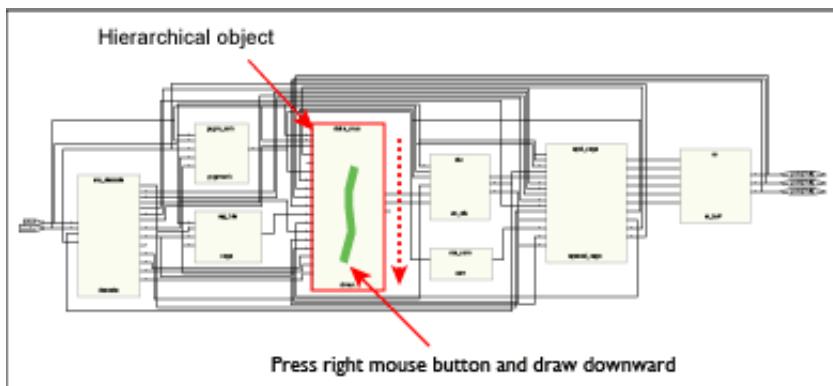
When combined with other commands like filtering and expansion commands, Push/Pop mode can be a very powerful tool for isolating and analyzing logic. See [Filtering Schematics, on page 315](#), [Expanding Pin and Net Logic, on page 318](#), and [Expanding and Viewing Connections, on page 321](#) for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

- [Pushing into Objects, on page 282, next](#)
- [Popping up a Hierarchical Level, on page 285](#)

## Pushing into Objects

In the schematic, you can push into objects and view the lower-level hierarchy. You can use a mouse stroke, the command, or the icon to push into objects:

1. To move down a level (push into an object) with a mouse stroke, put your cursor near the top of the object, hold down the right mouse button, and draw a vertical stroke from top to bottom. You can push into the following objects; see step 3 for examples of pushing into different types of objects.
  - Hierarchical instances. They can be displayed as pale yellow boxes (opaque instances) or hollow boxes with internal logic displayed (transparent instances). You cannot push into a hierarchical instance that is hidden with the Hide Instance command (internal logic is hidden).



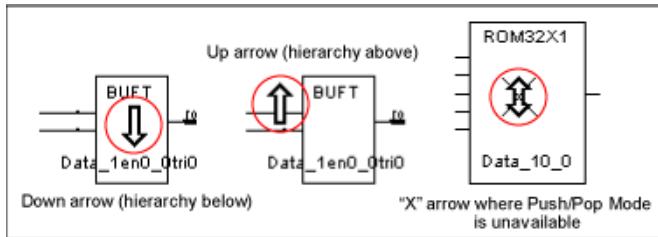
- Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the Technology view, under Instances - Primitives.
- Inferred ROMs and state machines.

The remaining steps show you how to use the icon or command to push into an object.

2. Enable Push/Pop mode by doing one of the following:
  - Select View->Push/Pop Hierarchy.

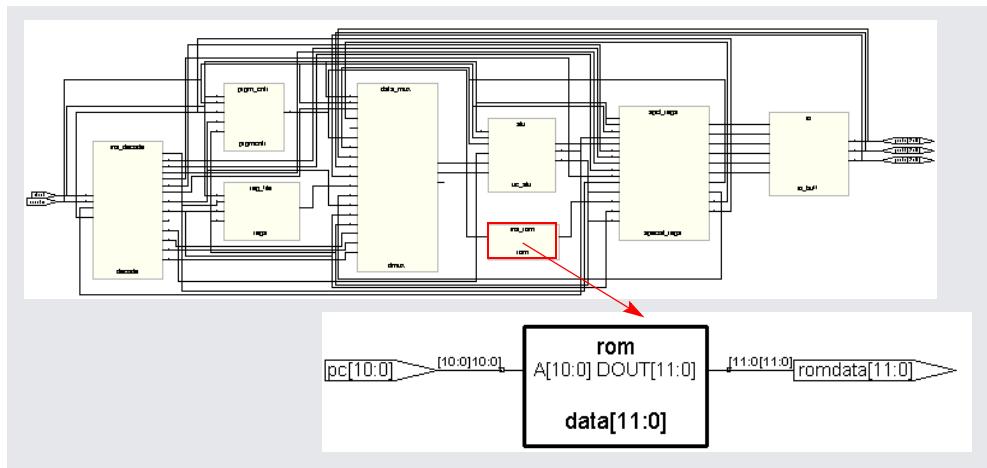
- Right-click in the Technology view and select Push/Pop Hierarchy from the popup menu.
- Click the Push/Pop Hierarchy icon ( in the toolbar (two arrows pointing up and down).
- Press F2.

The cursor changes to an arrow. The direction of the arrow indicates the underlying hierarchy, as shown in the following figure. The status bar at the bottom of the window reports information about the objects over which you move your cursor.

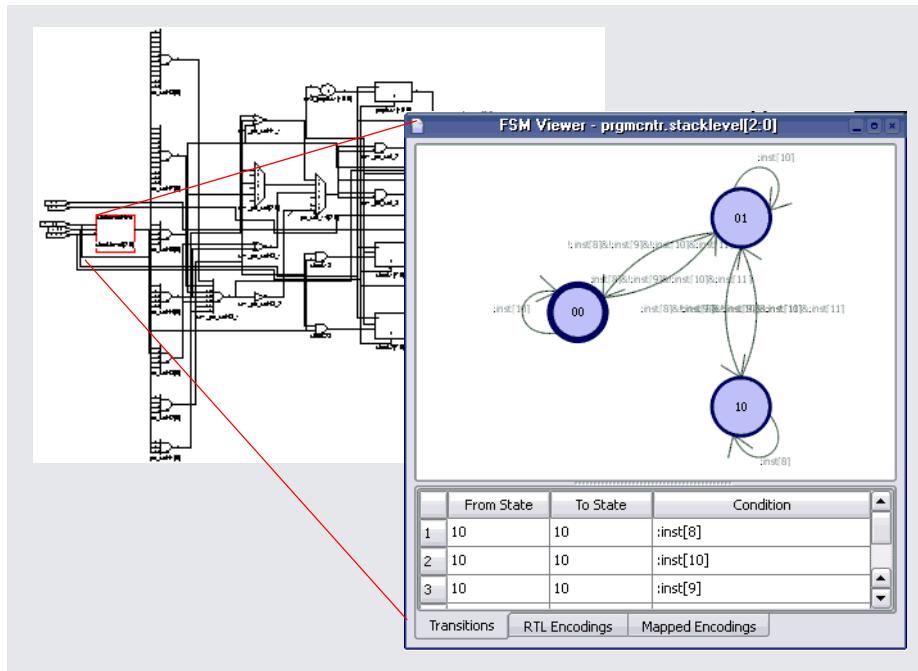


3. To push (descend) into an object, click on the hierarchical object. For a transparent instance, you must click on the pale yellow border. The following figure shows the result of pushing into a ROM.

When you descend into a ROM, you can push into it one more time to see the ROM data table. The information is in a view-only text file called `rom.info`.

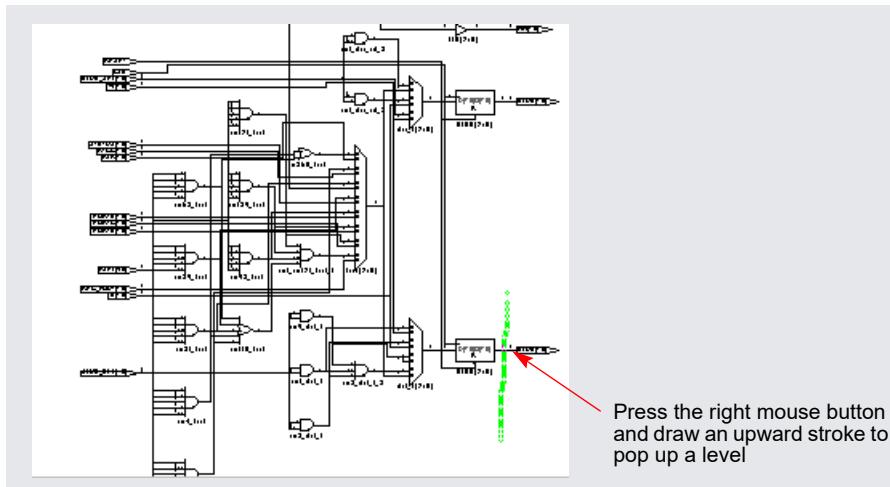


Similarly, you can push into a state machine. When you push into an FSM from the RTL view, you open the FSM viewer where you can graphically view the transitions. For more information, see [Using the FSM Viewer \(Standard\), on page 329](#). If you push into a state machine from the Technology view, you see the underlying logic.



## Popping up a Hierarchical Level

1. To move up a level (pop up a level), put your cursor anywhere in the design, hold down the right mouse button, and draw a vertical mouse stroke, moving from the bottom upwards.



The software moves up a level, and displays the next level of hierarchy.

2. To pop (ascend) a level using the commands or icon, do the following:
  - Select the command or icon if you are not already in Push/Pop mode. See [Pushing into Objects, on page 282](#) for details.
  - Move your cursor to a blank area and click.
3. To exit Push/Pop mode, do one of the following:
  - Click the right mouse button in a blank area of the view.
  - Deselect View->Push/Pop Hierarchy.
  - Deselect the Push/Pop Hierarchy icon.
  - Press F2.

## Exploring Object Hierarchy of Transparent Instances

Examining a transparent instance is one way of exploring the design hierarchy of an object. The following table compares this method with pushing (described in [Exploring Object Hierarchy by Pushing/Popping, on page 281](#)).

	<b>Pushing</b>	<b>Transparent Instance</b>
User control	You initiate the operation through the command or icon.	You have no direct control; the transparent instance is automatically generated by some commands that result in a filtered view.
Design context	Context lost; the lower-level logic is shown in a separate view	Context maintained; lower-level logic is displayed inside a hollow yellow box at the hierarchical level of the parent.

# Finding Objects (Standard)

In the schematic, you can use the Hierarchy Browser or the Find command to find objects, as explained in these sections:

- [Browsing to Find Objects in HDL Analyst Views, on page 288](#)
- [Using Find for Hierarchical and Restricted Searches, on page 291](#)
- [Using Wildcards with the Find Command, on page 295](#)
- [Using Find to Search the Output Netlist, on page 300](#)

For information about the Tcl Find command, which you use to locate objects, and create collections, see [find, on page 119](#) in the *Reference Manual*.

## Browsing to Find Objects in HDL Analyst Views

You can always zoom in to find an object in the RTL and Technology schematics. The following procedure shows you how to browse through design objects and find an object at any level of the design hierarchy. You can use the Hierarchy Browser or the Find command to do this. If you are familiar with the design hierarchy, the Hierarchy Browser can be the quickest method to locate an object. The Find command is best used to graphically browse and locate the object you want.

### Browsing With the Hierarchy Browser

1. In the Hierarchy Browser, click the name of the net, port, or instance you want to select.

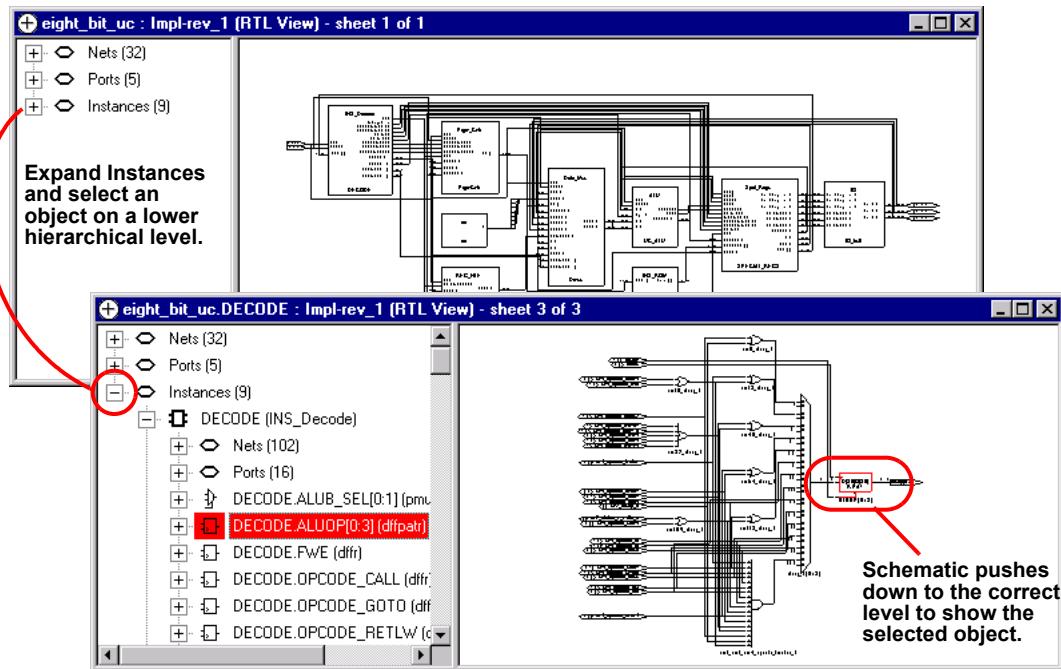
The object is highlighted in the schematic.

2. To select a range of objects, select the first object in the range. Then, scroll to display the last object in the range. Press and hold the Shift key while clicking the last object in the range.

The software selects and highlights all the objects in the range.

3. If the object is on a lower hierarchical level, do either of the following:
  - Expand the appropriate higher-level object by clicking the plus symbol next to it, and then select the object you want.
  - Push down into the higher-level object, and then select the object from the Hierarchy Browser.

The selected object is highlighted in the schematic. The following example shows how moving down the object hierarchy and selecting an object causes the schematic to move to the sheet and level that contains the selected object.



4. To select all objects of the same type, select them from the Hierarchy Browser. For example, you can find all the nets in your design.

## Browsing With the Find Command

1. In a schematic, select HDL Analyst->Find or press Ctrl-f to open the Object Query dialog box.
2. Do the following in the dialog box:

- Select objects in the selection box on the left. You can select all the objects or a smaller set of objects to browse. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the dialog box.
- Click the arrow to move the selected objects over to the box on the right.

The software highlights the selected objects.

3. In the Object Query dialog box, click on an object in the box on the right.

The software tracks to the schematic page with that object.

## Using Find for Hierarchical and Restricted Searches

You can always zoom in to find an object in the RTL and Technology schematics or use the Hierarchy Browser (see [Browsing to Find Objects in HDL Analyst Views, on page 288](#)). This procedure shows you how to use the Find command to do hierarchical object searches or restrict the search to the current level or the current level and its underlying hierarchy.

Note that Find only adds to the current selection; it does not deselect anything that is already selected. You can use successive searches to build up exactly the selection you need, before filtering.

1. If needed, restrict the range of the search by filtering the view.

See [Viewing Design Hierarchy and Context, on page 312](#) and [Filtering Schematics, on page 315](#) for details. With a filtered view, the software only searches the filtered instances, unless you set the scope of the search to Entire Design, as described below, in which case Find searches the entire design.

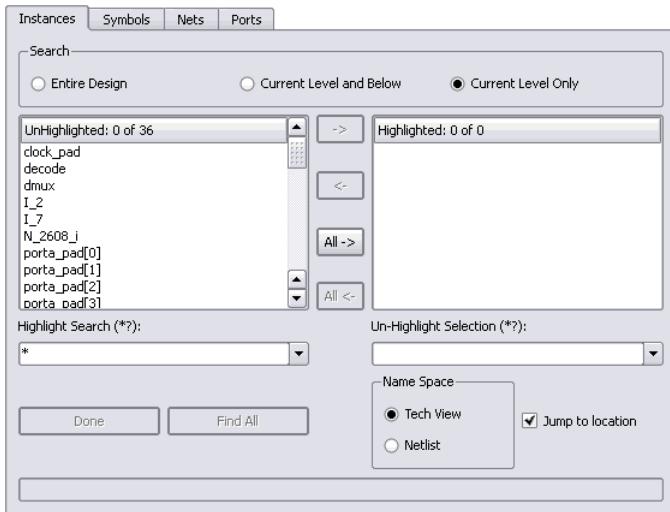
You can use the filtering technique to restrict your search to just one schematic sheet. Select all the objects on one sheet and filter the view. Continue with the procedure.

2. To further restrict the range of the search, hide instances you do not need.

You can do this in addition to filtering the view, or instead of filtering the view. Hidden instances and their hierarchy are excluded from the search. When you have finished the search, use the Unhide Instances command to make the hierarchy visible again.

3. Open the Object Query dialog box.

- Do one of the following: right click in the RTL or Technology view and select Find from the popup menu, press Ctrl-f, or click the Find icon ().
- Reposition the dialog box so you can see both your schematic and the dialog box.



4. Select the tab for the type of object. The Unhighlighted box on the left lists all objects of that type (instances, symbols, nets, or ports).

For fastest results, search by Instances rather than Nets. When you select Nets, the software loads the whole design, which could take some time.

5. Click one of these buttons to set the hierarchical range for the search: Entire Design, Current Level & Below, or Current Level Only, depending on the hierarchical level of the design to which you want to restrict your search.

The range setting is especially important when you use wildcards. See [Effect of Hierarchy and Range on Wildcard Searches, on page 295](#) for details. Current Level Only or Current Level & Below are useful for searching filtered schematics or critical path schematics.

The lower-level details of a transparent instance appear at the current level and are included in the search when you set it to Current Level Only. To exclude them, temporarily hide the transparent instances, as described in step 2.

Use Entire Design to hierarchically search the whole design. For large hierarchical designs, reduce the scope of the search by using the techniques described in the first step.

The Unhighlighted box shows available objects within the scope you set. Objects are listed in alphabetical order, not hierarchical order.

6. To search for objects in the mapped database or the output netlist, set the Name Space option.

The name of an object might be changed because of synthesis optimizations or to match the place-and-route tool conventions, so that the object name may no longer match the name in the original netlist. Setting the Name Space option ensures that the Find command searches the correct database for the object. For example, if you set this option to Tech View, the tool searches the mapped database (srm) for the object name you specify. For information about using this feature to find objects from an output netlist, see [Using Find to Search the Output Netlist, on page 300](#).

7. Do the following to select objects from the list. To use wildcards in the selection, see the next step.
  - Click on the objects you want from the list. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the dialog box.
  - Click Find 200 or Find All. The former finds the first 200 matches, and then you can click the button again to find the next 200.
  - Click the right arrow to move the objects into the box on the right, or double-click individual names.

The schematic displays highlighted objects in red.

8. Do the following to select objects using patterns or wildcards.

- Type a pattern in the Highlight Wildcard field. See [Using Wildcards with the Find Command, on page 295](#) for a detailed discussion of wildcards.

The Unhighlighted list shows the objects that match the wildcard criteria. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the form.

- Click the right arrow to move the selections to the box on the right, or double-click individual names. The schematic displays highlighted objects in red.

You can use wildcards to avoid typing long pathnames. Start with a general pattern, and then make it more specific. The following example browses and uses wildcards successively to narrow the search.

Find all instances three levels down	*.*.*
Narrow search to find instances that begin with i_	i_*.*.*
Narrow search to find instances that begin with un2 after the second hierarchy separator	i_*.*.un2*

Note that there are some differences when you specify the `find` command in the RTL view, Technology view, or the constraint file.

9. You can leave the dialog box open to do successive Find operations. Click OK or Cancel to close the dialog box when you are done.

For detailed information about the Find command and the Object Query dialog box, see [Find Command \(HDL Analyst\), on page 285](#) of the *Reference Manual*.

## Using Wildcards with the Find Command

Use the following wildcards when you search the schematics:

- \* The asterisk matches any sequence of characters.
- ? The question mark matches any single character.
- . The dot explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not a hierarchy separator, type a backslash before the dot: \.

### Effect of Hierarchy and Range on Wildcard Searches

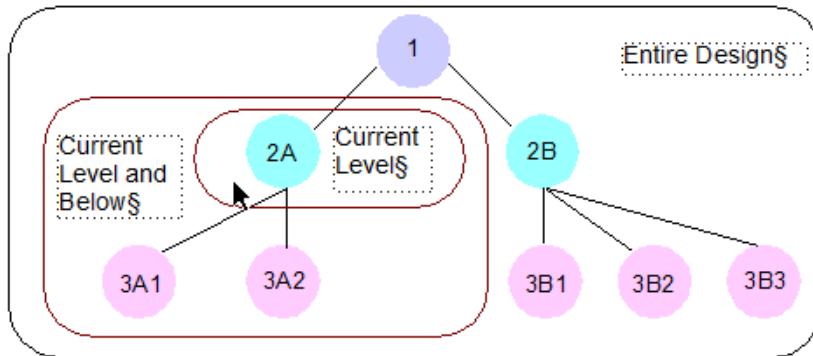
The asterisk and question mark wildcards do not cross hierarchical boundaries, but search each level of hierarchy individually with the search pattern. This default is affected by the following:

- Hierarchical separators

Dots match hierarchy separators, unless you use the backslash escape character in front of the dot (\.). Hierarchical search patterns with a dot (l\*.\*.) are repeated at each level included in the scope. If you use the \*.\* pattern with Current Level, the software matches non-hierarchical names at the current level that include a dot.

- Search range

The scope of the search determines the starting point for the searches. Sometimes the starting point might make it appear as if the wildcards cross hierarchical boundaries. If you are at 2A in the following figure and the scope of the search is set to Current Level and Below, separate searches start at 2A, 3A1, and 3A2. Each search does not cross hierarchical boundaries. If the scope of the search is Entire Design, the wildcard searches run from each hierarchical point (1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3). The result of an asterisk search (\*) with Entire Design is a list of all matches in the design, regardless of the current level.



See [Wildcard Search Examples, on page 297](#) for examples.

## How a Wildcard Search Works

- The starting point of a wildcard search depends on the range set for the search.

**Entire Design** Starts at top level and uses the pattern to search from that level. It then moves to any child levels below the top level and searches them. The software repeats the search pattern at each hierarchical point in the design until it searches the entire design.

**Current Level** Starts at the current hierarchical level and searches that level only. A search started at 2A only covers 2A.

**Current Level and Below** Starts at the current hierarchical level and searches that level. It then moves to any child levels below the starting point and conducts separate searches from each of these starting points.

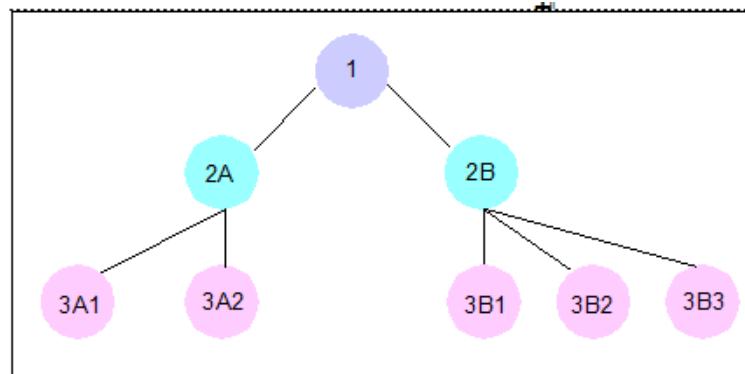
- The software applies the wildcard pattern to all applicable objects within the range. For Current Level and Current Level and Below, the current level determines the starting point.

Dots match hierarchy separators, unless you use the backslash escape character in front of the dot (\.). Hierarchical search patterns with a dot (l\*.\*) are repeated at each level included in the scope. See [Effect of Hierarchy and Range on Wildcard Searches, on page 295](#) and [Wildcard Search Examples, on page 297](#) for details and examples, respectively. If

if you use the `*.*` pattern with Current Level, the software matches non-hierarchical names at the current level that include a dot.

## Wildcard Search Examples

The figure shows a design with three hierarchical levels, and the table shows the results of some searches on this design.



Scope	Pattern	Starting Point	Finds Matches in...
Entire Design	*	3A1	1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3 (* at all levels)
	*.*	2B	2A and 2B (*.*) from 1) 3A1, 3A2, 3B1, 3B2, and 3B3 (*.*) from 2A and 2B) No matches in 1 (because of the hierarchical dot), unless a name includes a non-hierarchical dot.
Current Level	*	1	1 only (no hierarchical boundary crossing)
	*.*	2B	2B only. No search of lower levels even though the dot is specified, because the scope is Current Level. No matches, unless a 2B name includes a non-hierarchical dot.

Scope	Pattern	Starting Point	Finds Matches in...
Current Level and Below	*	2A	2A only (no hierarchical boundary crossing)
	*.*	1	2A and 2B (*.* from 1) 3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B) No matches from 1, because the dot is specified.
	*.*	2B	3B1, 3B2, and 3B3 (*.* from 2B)
	*.*	3A2	No matches (no hierarchy below 3A2)
	*.*.*	1	3A1, 3A2, 3B1, 3B2, and 3B3 (*.*.* from 1) Search ends because there is no hierarchy two levels below 2A and 2B.

## Difference from Tcl Search

The FPGA synthesis tools and Synopsys TimeQuest and Design Compiler products confine the simple search to within one level of hierarchy. The following command searches each level of hierarchy individually for the specified pattern:

```
find -hier *abc*addr_reg[*]
```

If you want to go through the hierarchy, you must add the hierarchy separators to the search pattern:

```
find {*.*.abc.*.*.addr_reg[*]}
```

## Find Command Differences in HDL Analyst Views and Constraint File

There are some slight differences when you use the Find command in the RTL view, Technology view, and the constraint files:

- You cannot use find to search for bit registers of a bit array in the RTL or Technology views, but you can specify it in a constraint file, where the following command will work:

```
find -seq {i:modulex_inst.qb[7]}
```

In a HDL Analyst view, you cannot find {i:modulex\_inst.qb[7]}, but you can specify and find {i:modulex\_inst.qb[7:0]}.

- By default, the following Tcl command does not find objects in the RTL view, although it does find objects in the Technology view:

```
-hier -seq * -filter @clock == clk75
```

To make this work in an RTL view, you must turn on Annotated Properties for Analyst in the Device tab of the Implementation Options dialog box, recompile the design, and then open a new RTL view.

## Combining Find with Filtering to Refine Searches

You can combine the Find command with the filtering commands to better effect. Depending on what you want to do, use the Find command first, or a filtering command.

1. To limit the range of a search, do the following:
  - Filter the design.
  - Use the Find command on the filtered view, but set the search range to Current Level Only.
2. Select objects for a filtered view.
  - Use the Find command to browse and select objects.
  - Filter the objects to display them.

## Using Find to Search the Output Netlist

When the synthesis tool creates an output netlist like a vqm or edf file, some names are optimized for use in the P&R tool. When you debug your design for place and route looking for a particular object, use the Name Space option in the Object Query dialog box to locate the optimized names in the output netlist. The following procedure shows you how to locate an object, highlight and filter it in the Technology view, and crossprobe to the source code for editing.

1. Select the output netlist file option in the Implementations Results tab of the Implementation Options dialog box.
2. After you synthesize your design, open your output netlist file and select the name of the object you want to find.

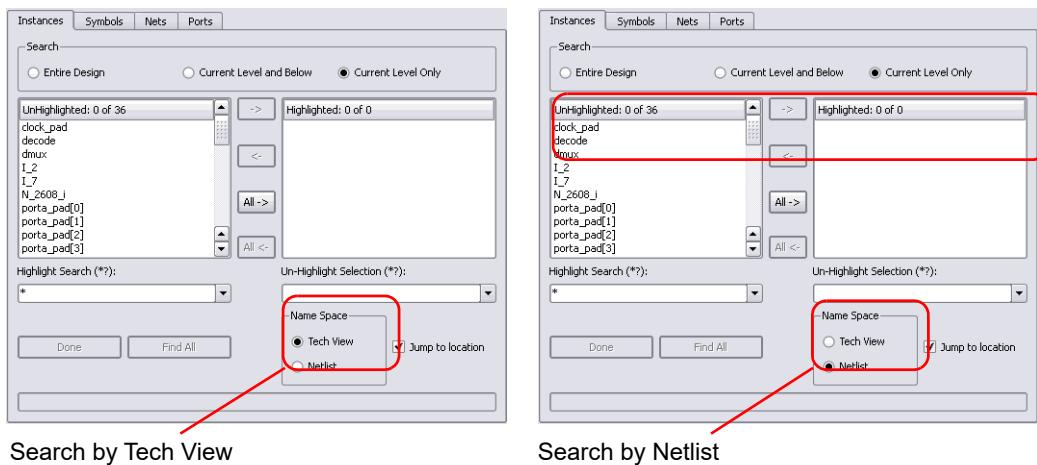
```

455 );
456 defparam DATA0_c1.lut_mask=64'h0000ffff0000ff00;
457 defparam DATA0_c1.shared_arith="off";
458 defparam DATA0_c1.extended_lut="off";
459 // @5:46
460     stratixiii_lcell_comb DATA0_c3 (
461         .sumout(DATA0_c3_sumout),
462         .cout(DATA0_c3_cout),
463         .dataaf(VCC),
464         .dataae(VCC),
465         .datad(DATA0_internal_3),
466         .dataac(GND),
467         .dataab(VCC),
468         .dataaa(VCC),
469         .dataag(VCC),
470         .cin(DATA0_c2_cout)

```

Copy Name

3. Copy the name and open a Technology view.
4. In the Technology view, press Ctrl-f or select Edit->Find to open the Object Query dialog box and do the following:
  - Paste the object name you copied into the Highlight Search field.
  - Set the Name Space option to Netlist and click Find All.

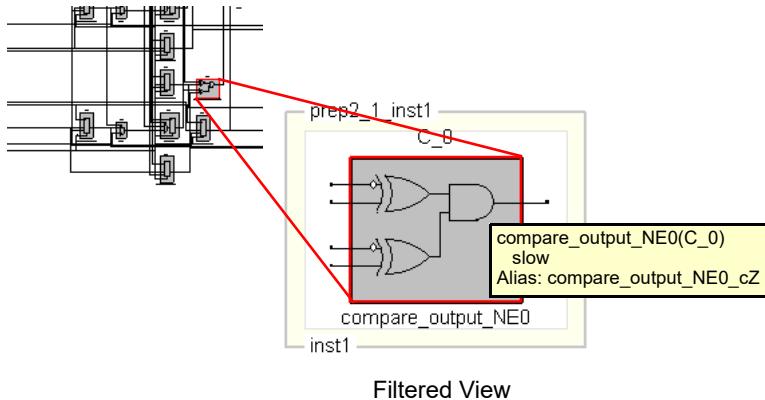


If you leave the Name Space option set to the default of Tech View, the tool does not find the name because it is searching the mapped database instead of the output netlist.

- Double click the name to move it into the Highlighted field and close the dialog box.

In the Technology view, the name is highlighted in the schematic.

5. Select HDL Analyst->Filter Schematic to view only the highlighted portion of the schematic.



Filtered View

The tooltip shows the equivalent name in the Technology view.

6. Double click the filtered schematic to crossprobe to the corresponding code in the HDL file.

# Crossprobing (Standard)

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. Highlighting a line of text, for example, highlights the corresponding logic in the schematic. Crossprobing helps you visualize where coding changes or timing constraints might help to reduce area or improve performance.

You can crossprobe between the RTL view, Technology view, the FSM Viewer, the log file, the source files, and some external text files from place-and-route tools. However, not all objects or source code crossprobe to other views, because some source code and RTL view logic is optimized away during the compilation or mapping processes.

This section describes how to crossprobe from different views. It includes the following:

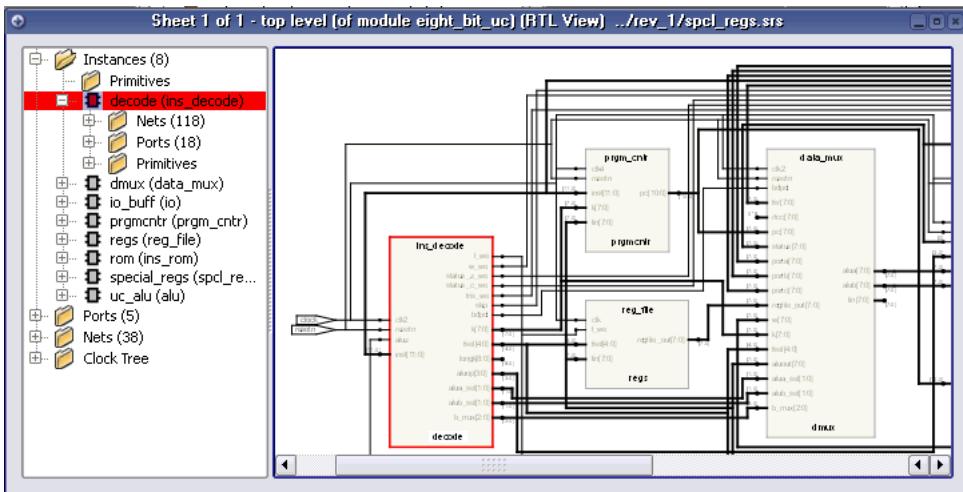
- [Crossprobing within an RTL/Technology View, on page 303](#)
- [Crossprobing from the RTL/Technology View, on page 304](#)
- [Crossprobing from the Text Editor Window, on page 306](#)
- [Crossprobing from the Tcl Script Window, on page 309](#)
- [Crossprobing from the FSM Viewer, on page 310](#)

## Crossprobing within an RTL/Technology View

Selecting an object name in the Hierarchy Browser highlights the object in the schematic, and vice versa.

Selected Object	Highlighted Object
Instance in schematic (single-click)	Module icon in Hierarchy Browser
Net in schematic	Net icon in Hierarchy Browser
Port in schematic	Port icon in Hierarchy Browser
Logic icon in Hierarchy Browser	Instance in schematic
Net icon in Hierarchy Browser	Net in schematic
Port icon in Hierarchy Browser	Port in schematic

In this example, when you select the DECODE module in the Hierarchy Browser, the DECODE module is automatically selected in the RTL view.

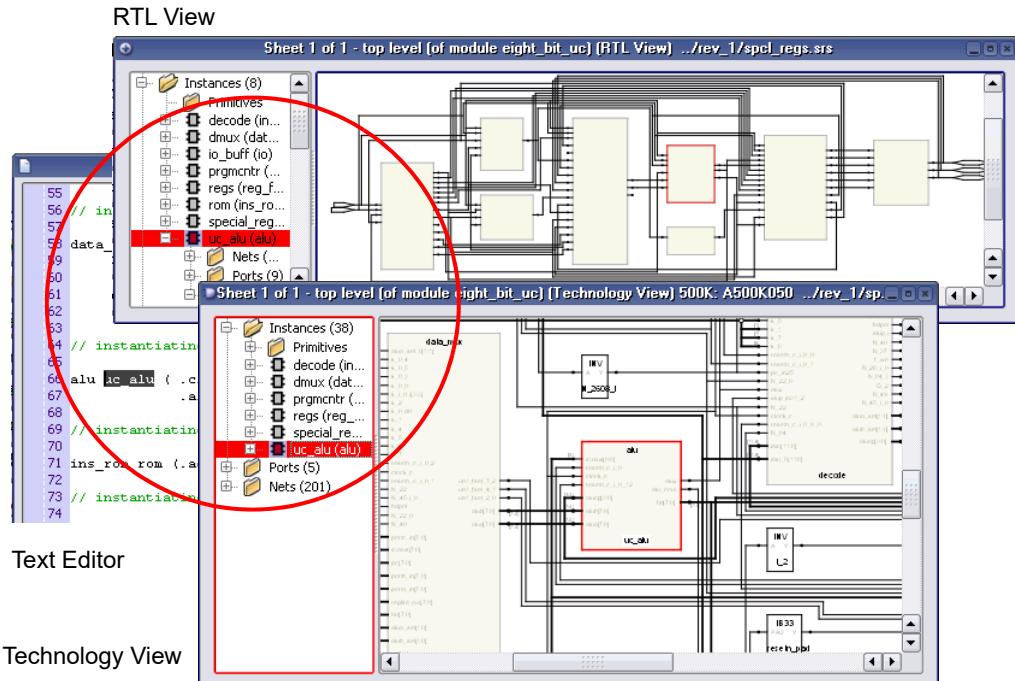


## Crossprobing from the RTL/Technology View

1. To crossprobe from an RTL or Technology views to other open views, select the object by clicking on it.

The software automatically highlights the object in all open views. If the open view is a schematic, the software highlights the object in the Hierarchy Browser on the left as well as in the schematic. If the highlighted object is on another sheet of a multi-sheet schematic, the view does not automatically track to the page. If the crossprobed object is inside a hidden instance, the hidden instance is highlighted in the schematic.

If the open view is a source file, the software tracks to the appropriate code and highlights it. The following figure shows crossprobing between the RTL, Technology, and Text Editor (source code) views.



2. To crossprobe from the RTL or Technology view to the source file when the source file is not open, double-click on the object in the RTL or Technology view.

Double-clicking automatically opens the appropriate source code file and highlights the appropriate code. For example, if you double-click an object in a Technology view, the HDL Analyst tool automatically opens an editor window with the source code and highlights the code that contains the selected register.

The following table summarizes the crossprobing capability from the RTL or Technology view.

<b>From</b>	<b>To</b>	<b>Procedure</b>
RTL	Source code	Double-click an object. If the source code file is not open, the software opens the Text Editor window to the appropriate section of code. If the source file is already open, the software scrolls to the correct section of the code and highlights it.
RTL	Technology	The Technology view must be open. Click the object to highlight and crossprobe.
RTL	FSM Viewer	The FSM view must be open. The state machine must be coded with a onehot encoding style. Click the FSM to highlight and crossprobe.
Technology	Source code	If the source code file is already, open, the software scrolls to the correct section of the code and highlights it. If the source code file is not open, double-click an object in the Technology view to open the source code file.
Technology	RTL	The RTL view must be open. Click the object to highlight and crossprobe.

## Crossprobing from the Text Editor Window

To crossprobe from a source code window or from the log file to an RTL, Technology, or FSM view, use this procedure. You can use this method to crossprobe from any text file with objects that have the same instance names as in the synthesis software. For example, you can crossprobe from place-and-route files. See [Example of Crossprobing a Path from a Text File, on page 307](#) for a practical example of how to use crossprobing.

1. Open the RTL, FSM, or Technology view to which you want to crossprobe.
2. To crossprobe from an error, warning, or note in the html log file, click on the file name to open the corresponding source code in another Text Editor window; to crossprobe from a text log file, double-click on the text of the error, warning, or note.
3. To crossprobe from a third-party text file (not source code or a log file), select Options->HDL Analyst Options->General, and enable Enhanced text crossprobing.

4. Select the appropriate portion of text in the Text Editor window. In some cases, it may be necessary to select an entire block of text to crossprobe.

The software highlights the objects corresponding to the selected code in all the open windows. For example, if you select a state name in the code, it highlights the state in the FSM viewer. If an object is on another schematic sheet or on another hierarchical level, the highlighting might not be obvious. If you filter the RTL or schematic (right-click in the source code window with the selected text and select Filter Schematic from the popup menu), you can isolate the highlighted objects for easy viewing.

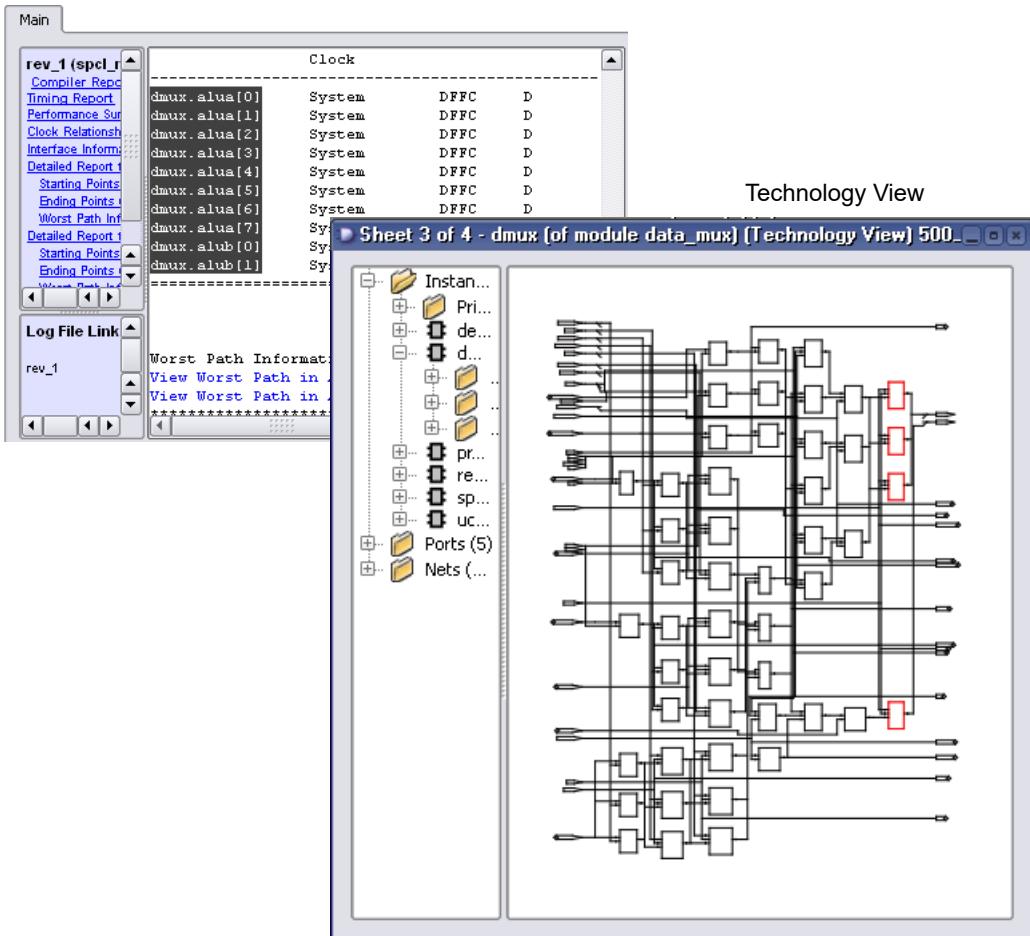
### Example of Crossprobing a Path from a Text File

This example selects a path in a log file and crossprobes it in the Technology view. You can use the same technique to crossprobe from other text files like place-and-route files, as long as the instance names in the text file match the instance names in the synthesis tool.

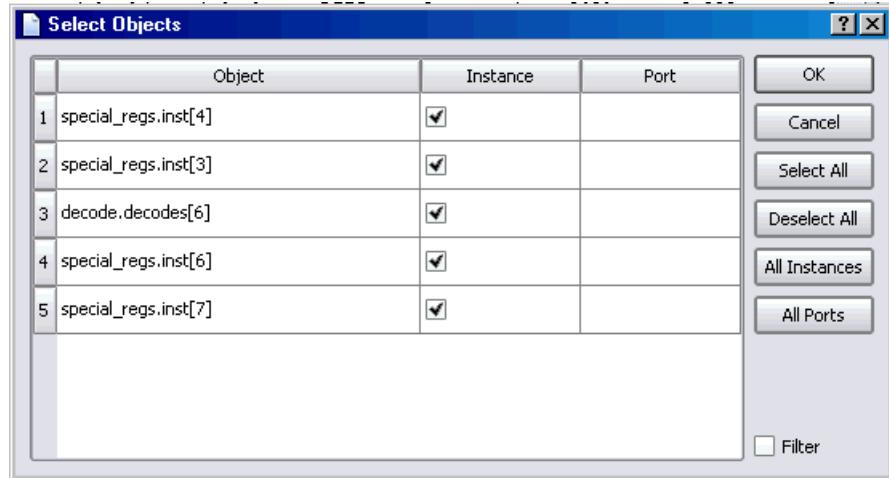
1. Open the log file, the RTL, and Technology views.
2. Select the path objects in the log file.
  - Select the column by pressing Alt and dragging the cursor to the end of the column. On the Linux platform, use the key to which the Alt function is mapped; this is usually the Ctrl-Alt key combination.
  - To select all the objects in the path, right-click and choose Select in Analyst from the popup menu. Alternatively, you can select certain objects only, as described next.

The software selects the objects in the column, and highlights the path in the open RTL and Technology views.

## Text Editor



- To further filter the objects in the path, right-click and choose Select From from the popup menu. On the form, check the objects you want, and click OK. Only the corresponding objects are highlighted.



3. To isolate and view only the selected objects, do this in the Technology view: press F12, or right-click and select the Filter Schematic command from the popup menu.

You see just the selected objects.

## Crossprobing from the Tcl Script Window

Crossprobing from the Tcl script window is useful for debugging error messages.

To crossprobe from the Tcl Script window to the source code, double-click a line in the Tcl window. To crossprobe a warning or error, first click the Messages tab and then double-click the warning or error. The software opens the relevant source code file and highlights the corresponding code.

## Crossprobing from the FSM Viewer

You can crossprobe to the FSM Viewer if you have the FSM view open. You can crossprobe from an RTL, Technology, or source code window.

To crossprobe from the FSM Viewer, do the following:

1. Open the view to which you want to crossprobe: RTL/Technology view, or the source code file.
2. Do the following in the open FSM view:
  - For FSMs with a onehot encoding style, click the state bubbles in the bubble diagram or the states in the FSM transition table.
  - For all other FSMs, click the states in the bubble diagram. You cannot use the transition table because with these encoding styles, the number of registers in the RTL or Technology views do not match the number of registers in the FSM Viewer.

The software highlights the corresponding code or object in the open views. You can only crossprobe from a state in the FSM table if you used a onehot encoding style.

# Analyzing With the Standard HDL Analyst Tool

The HDL Analyst tool is a graphical productivity tool that helps you visualize your synthesis results. It consists of RTL-level and technology-primitive level schematics that let you graphically view and analyze your design.

- **RTL View**

Using BEST® (Behavior Extracting Synthesis Technology) in the RTL view, the software keeps a high-level of abstraction and makes the RTL view easy to view and debug. High-level structures like RAMs, ROMs, operators, and FSMs are kept as abstractions in this view instead of being converted to gates. You can examine the high-level structure, or push into a component and view the gate-level structure.

- **Technology View**

The software uses module generators to implement the high-level structures from the RTL view, and maps them to technology-specific resources.

To analyze information, compare the current view with the information in the RTL/Technology view, the log file, the FSM view, and the source code, you can use techniques like crossprobing, flattening, and filtering. See the following for more information about analysis techniques.

- [Viewing Design Hierarchy and Context, on page 312](#)
- [Filtering Schematics, on page 315](#)
- [Expanding Pin and Net Logic, on page 318](#)
- [Expanding and Viewing Connections, on page 321](#)
- [Flattening Schematic Hierarchy, on page 322](#)
- [Minimizing Memory Usage While Analyzing Designs, on page 327](#)

For additional information about navigating the HDL Analyst views or using other techniques like crossprobing, see the following:

- [Working in the Standard Schematic, on page 265](#)
- [Exploring Design Hierarchy \(Standard\), on page 280](#)
- [Finding Objects \(Standard\), on page 288](#)
- [Crossprobing \(Standard\), on page 303](#)

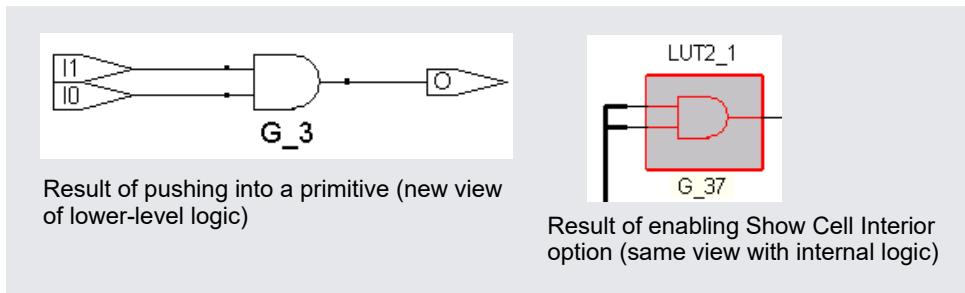
## Viewing Design Hierarchy and Context

Most large designs are hierarchical, so the synthesis software provides tools that help you view hierarchy details or put the details in context. Alternatively, you can browse and navigate hierarchy with Push/Pop mode, or flatten the design to view internal hierarchy.

This section describes how to use interactive hierarchical viewing operations to better analyze your design. Automatic hierarchy viewing operations that are built into other commands are described in the context in which they appear. For example, [Viewing Critical Paths, on page 339](#) describes how the software automatically traces a critical path through different hierarchical levels using hollow boxes with nested internal logic (transparent instances) to indicate levels in hierarchical instances.

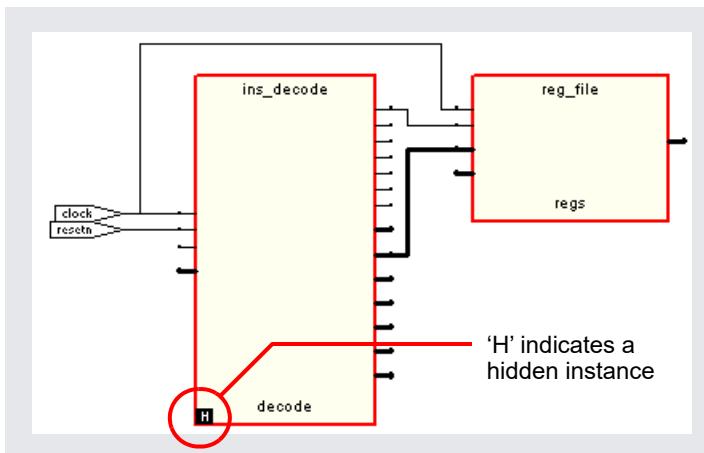
1. To view the internal logic of primitives in your design, do either of the following:
  - To view the logic of an individual primitive, push into it. This generates a new schematic with the internal details. Click the Back icon to return to the previous view.
  - To view the logic of all primitives in the design, select Options->HDL Analyst Options->General, and enable Show Cell Interior. This command lets you see internal logic in context, by adding the internal details to the current schematic and all subsequent views. If the view is too cluttered with this option on, filter the view (see [Filtering Schematics, on page 315](#)) or push into the primitive. Click the Back icon to return to the previous view after filtering or pushing into the object.

The following figure compares these two methods:



2. To hide selected hierarchy, select the instance whose hierarchy you want to exclude, and then select Hide Instances from the HDL Analyst menu or the right-click popup menu in the schematic.

You can hide opaque (solid yellow) or transparent (hollow) instances. The software marks hidden instances with an H in the lower left. Hidden instances are like black boxes; their hierarchy is excluded from filtering, expanding, dissolving, or searching in the current window, although they can be crossprobed. An instance is only hidden in the current view window; other view windows are not affected. Temporarily hiding unnecessary hierarchy focuses analysis and saves time in large designs.



Before you save a design with hidden instances, select Unhide Instances from the HDL Analyst menu or the right-click popup menu and make the hidden internal hierarchy accessible again. Otherwise, the hidden instances are saved as black boxes, without their internal logic. Conversely, you can use this feature to reduce the scope of analysis in a large design by hiding instances you do not need, saving the reduced design to a new name, and then analyzing it.

3. To view the internal logic of a hierarchical instance, you can push into the instance, dissolve the selected instance with the Dissolve Instances command, or flatten the design. You cannot use these methods to view the internal logic of a hidden instance.

Pushing into an instance	Generates a view that shows only the internal logic. You do not see the internal hierarchy in context. To return to the previous view, click Back. See <a href="#">Exploring Object Hierarchy by Pushing/Popping, on page 281</a> for details.
Flattening the entire design	Opens a new view where the entire design is flattened, except for hidden hierarchy. Large flattened designs can be overwhelming. See <a href="#">Flattening Schematic Hierarchy, on page 322</a> for details about flattening designs.  Because this is a new view, you cannot use Back to return to the previous view. To return to the top-level unflattened schematic, right-click in the view and select Unflatten Schematic.
Flattening an instance by dissolving	Generates a view where the hierarchy of the selected instances is flattened, but the rest of the design is unaffected. This provides context. See <a href="#">Flattening Schematic Hierarchy, on page 322</a> for details about dissolving instances.

4. If the result of filtering or dissolving is a hollow box with no internal logic, try either of the following, as appropriate, to view the internal hierarchy:
  - Select Options->HDL Analyst Options->Sheet Size and increase the value of Maximum Filtered Instances. Use this option if the view is not too cluttered.
  - Use the sheet navigation commands to go to the sheets indicated in the hollow box.

If there is too much internal logic to display in the current view, the software puts the internal hierarchy on separate schematic sheets. It displays a hollow box with no internal logic and indicates the schematic sheets that contain the internal logic.

5. To view the design context of an instance in a filtered view, select the instance, right-click, and select Show Context from the popup menu.

The software displays an unfiltered view of the hierarchical level that contains the selected object, with the instance highlighted. This is useful when you have to go back and forth between different views during analysis. The context differs from the Expand commands, which show connections. To return to the original filtered view, click Back.

## Filtering Schematics

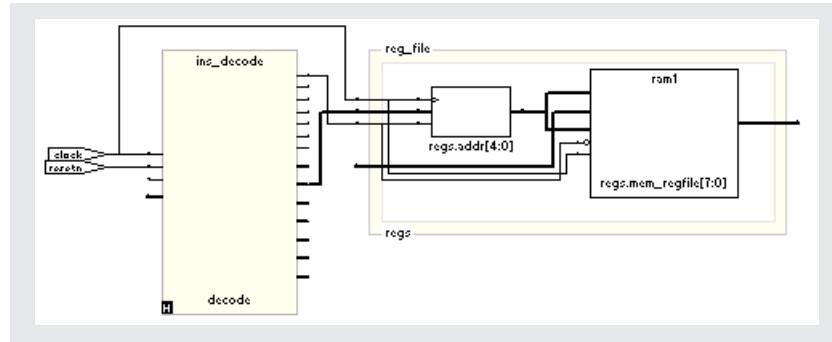
Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter Schematic command to isolate selected objects. See Chapter 3 of the *Reference Manual* for details about these commands.

This table lists the advantages of using filtering over flattening:

Filter Schematic Command	Flatten Commands
Loads part of the design; better memory usage	Loads entire design
Combine filtering with Push/Pop mode, and history buttons (Back and Forward) to move freely between hierarchical levels	Must use Unflatten Schematic to return to top level, and flatten the design again to see lower levels. Cannot return to previous view if the previous view is not the top-level view.

1. Select the objects that you want to isolate. For example, you can select two connected objects.

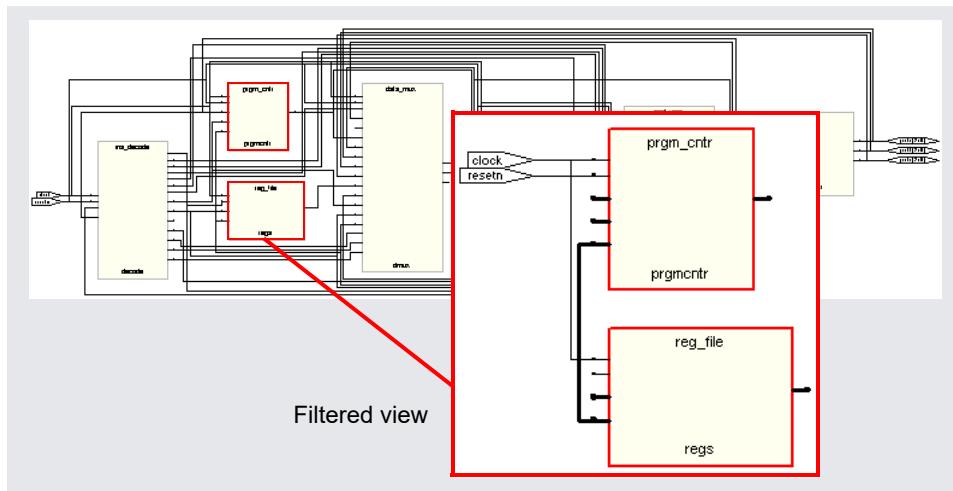
If you filter a hidden instance, the software does not display its internal hierarchy when you filter the design. The following example illustrates this.



2. Select the Filter Schematic command, using one of these methods:

- Select Filter Schematic from the HDL Analyst menu or the right-click popup menu.
- Click the Filter Schematic icon (buffer gate) ().
- Press F12.
- Press the right mouse button and draw a narrow V-shaped mouse stroke in the schematic window. See Help->Mouse Stroke Tutor for details.

The software filters the design and displays the selected objects in a filtered view. The title bar indicates that it is a filtered view. Hidden instances have an H in the lower left. The view displays other hierarchical instances as hollow boxes with nested internal logic (transparent instances). For descriptions of filtered views and transparent instances, see [Filtered and Unfiltered Schematic Views, on page 85](#) and [Transparent and Opaque Display of Hierarchical Instances, on page 91](#) in the *Reference Manual*. If the transparent instance does not display internal logic, use one of the alternatives described in [Viewing Design Hierarchy and Context, on page 312](#), step 4.



3. If the filtered view does not display the pin names of technology primitives and transparent instances that you want to see, do the following:
  - Select Options->HDL Analyst Options->Text and enable Show Pin Name.
  - To temporarily display a pin name, move the cursor over the pin. The name is displayed as long as the cursor remains over the pin. Alternatively, select a pin. The software displays the pin name until you make another selection. Either of these options can be applied to individual pins. Use them to view just the pin names you need and keep design clutter to a minimum.
  - To see all the hierarchical pins, select the instance, right-click, and select Show All Hier Pins.

You can now analyze the problem, and do operations like the following:

Trace paths, build up logic	See <a href="#">Expanding Pin and Net Logic, on page 318</a> and <a href="#">Expanding and Viewing Connections, on page 321</a>
Filter further	Select objects and filter again
Find objects	See <a href="#">Finding Objects (Standard), on page 288</a>
Flatten, or hide and flatten	See <a href="#">Flattening Schematic Hierarchy, on page 322</a> . You can hide transparent or opaque instances.
Crossprobe from filtered view	See <a href="#">Crossprobing from the RTL/Technology View, on page 304</a>

4. To return to the previous schematic, click the Back icon. If you flattened the hierarchy, right-click and select Unflatten Schematic to return to the top-level unflattened view.

For additional information about filtering schematics, see [Filtering Schematics, on page 315](#) and [Flattening Schematic Hierarchy, on page 322](#).

## Expanding Pin and Net Logic

When you are working in a filtered view, you might need to include more logic in your selected set to debug your design. This section describes commands that expand logic fanning out from pins or nets; to expand paths, see [Expanding and Viewing Connections, on page 321](#).

Use the Expand commands with the Filter Schematic, Hide Instances, and Flatten commands to isolate just the logic that you want to examine. Filtering isolates logic, flattening removes hierarchy, and hiding instances prevents their internal hierarchy from being expanded. See [Filtering Schematics, on page 315](#) and [Flattening Schematic Hierarchy, on page 322](#) for details.

1. To expand logic from a pin hierarchically across boundaries, use the following commands.

To...	Do this (HDL Analyst->Hierarchical/Popup menu)...
See all cells connected to a pin	Select a pin and select Expand. See <a href="#">Expanding Filtered Logic Example, on page 320</a> .
See all cells that are connected to a pin, up to the next register	Select a pin and select Expand to Register/Port. See <a href="#">Expanding Filtered Logic to Register/Port Example, on page 320</a> .
See internal cells connected to a pin	Select a pin and select Expand Inwards. The software filters the schematic and displays the internal cells closest to the port. See <a href="#">Expanding Inwards Example, on page 321</a> .

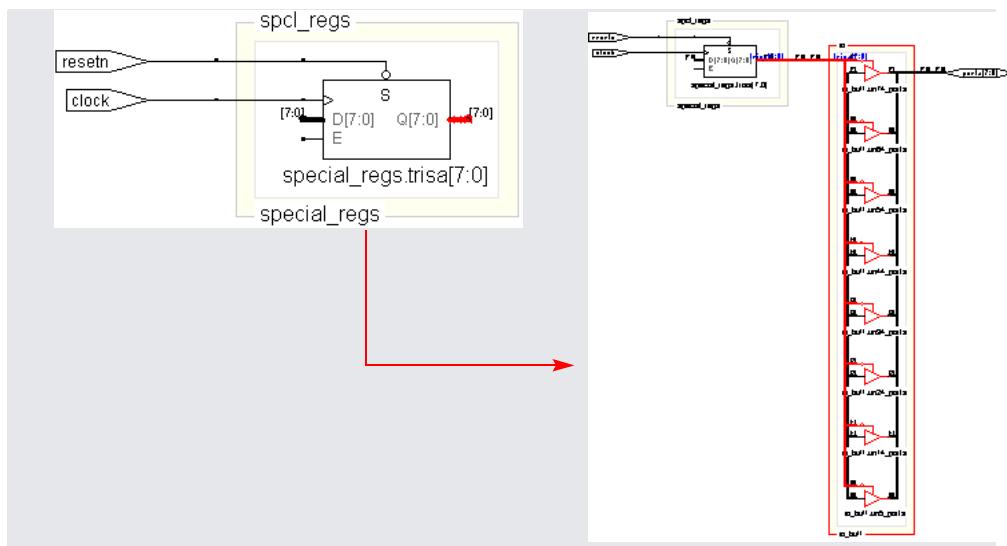
The software expands the logic as specified, working on the current level and below or working up the hierarchy, crossing hierarchical boundaries as needed. Hierarchical levels are shown nested in hollow bounding boxes. The internal hierarchy of hidden instances is not displayed.

For descriptions of the Expand commands, see [HDL Analyst Menu, on page 372](#) of the *Reference Manual*.

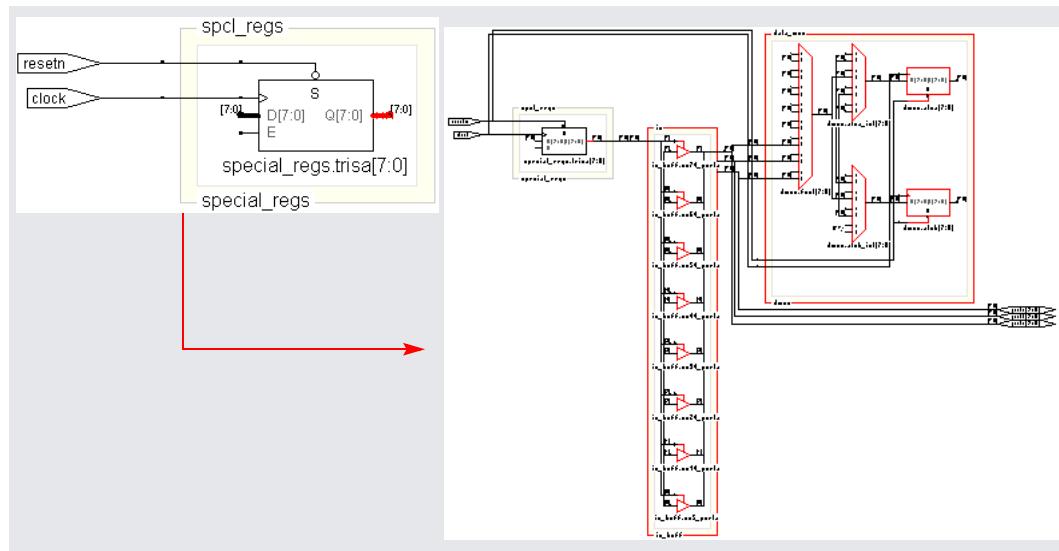
2. To expand logic from a pin at the current level only, do the following:
  - Select a pin, and go to the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.
  - Select Expand or Expand to Register/Ports. The commands work as described in the previous step, but they do not cross hierarchical boundaries.
3. To expand logic from a net, use the commands shown in the following table.
  - To expand at the current level and below, select the commands from the HDL Analyst->Hierarchical menu or the right-click popup menu.
  - To expand at the current level only, select the commands from the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.

To...	Do this...
Select the driver of a net	Select a net and select Select Net Driver. The result is a filtered view with the net driver selected ( <a href="#">Selecting the Net Driver Example, on page 321</a> ).
Trace the driver, across sheets if needed	Select a net and select Go to Net Driver. The software shows a view that includes the net driver.
Select all instances on a net	Select a net and select Select Net Instances. You see a filtered view of all instances connected to the selected net.

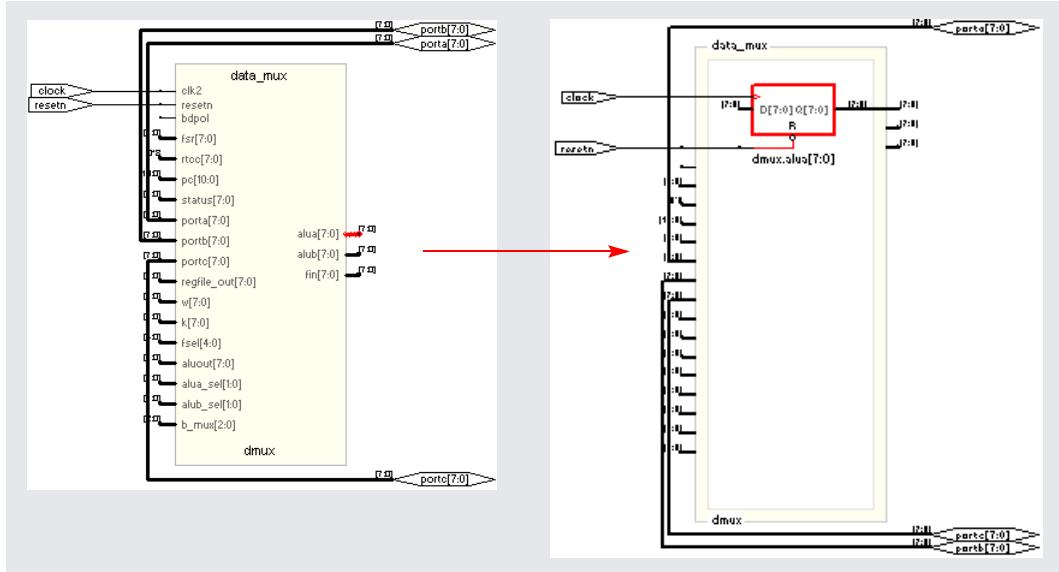
## Expanding Filtered Logic Example



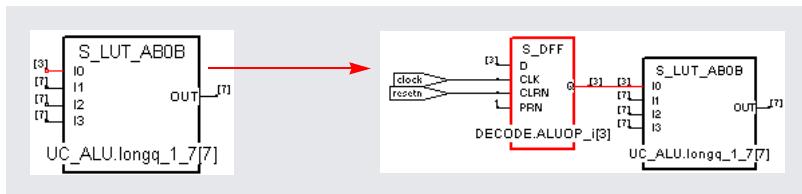
## Expanding Filtered Logic to Register/Port Example



## Expanding Inwards Example



## Selecting the Net Driver Example



## Expanding and Viewing Connections

This section describes commands that expand logic between two or more objects; to expand logic out from a net or pin, see [Expanding Pin and Net Logic, on page 318](#). You can also isolate the critical path or use the Timing Analyst to generate a schematic for a path between objects, as described in [Analyzing Timing in Schematic Views, on page 336](#).

Use the following path commands with the Filter Schematic and Hide Instances commands to isolate just the logic that you want to examine. The two techniques described here differ: Expand Paths expands connections between selected objects, while Isolate Paths pares down the current view to only display connections to and from the selected instance.

For detailed descriptions of the commands mentioned here, see [Commands That Result in Filtered Schematics, on page 113](#) in the *Reference Manual*.

1. To expand and view connections between selected objects, do the following:
  - Select two or more points.
  - To expand the logic at the current level only, select HDL Analyst->Current Level->Expand Paths or popup menu->Current Level Expand Paths.
  - To expand the logic at the current level and below, select HDL Analyst->Hierarchical->Expand Paths or popup menu->Expand Paths.
2. To view connections from all pins of a selected instance, right-click and select Isolate Paths from the popup menu.

Starting Point    The Filtered View Traces Paths (Forward and Back) From All Pins of the Selected Instance...

---

Filtered view    Traces through all sheets of the filtered view, up to the next port, register, hierarchical instance, or black box.

---

Unfiltered view    Traces paths on the current schematic sheet only, up to the next port, register, hierarchical instance, or black box.

Unlike the Expand Paths command, the connections are based on the schematic used as the starting point; the software does not add any objects that were not in the starting schematic.

## Flattening Schematic Hierarchy

Flattening removes hierarchy so you can view the logic without hierarchical levels. In most cases, you do not have to flatten your hierarchical schematic to debug and analyze your design, because you can use a combination of filtering, Push/Pop mode, and expanding to view logic at different levels. However, if you must flatten the design, use the following techniques., which include flattening, dissolving, and hiding instances.

1. To flatten an entire design down to logic cells, use one of the following commands:
  - For an RTL view, select HDL Analyst->RTL->Flattened View. This flattens the design to generic logic cells.
  - For a Technology view, select Flattened View or Flattened to Gates View from the HDL Analyst->Technology menu. Use the former command to flatten the design to the technology primitive level, and the latter command to flatten it further to the equivalent Boolean logic.

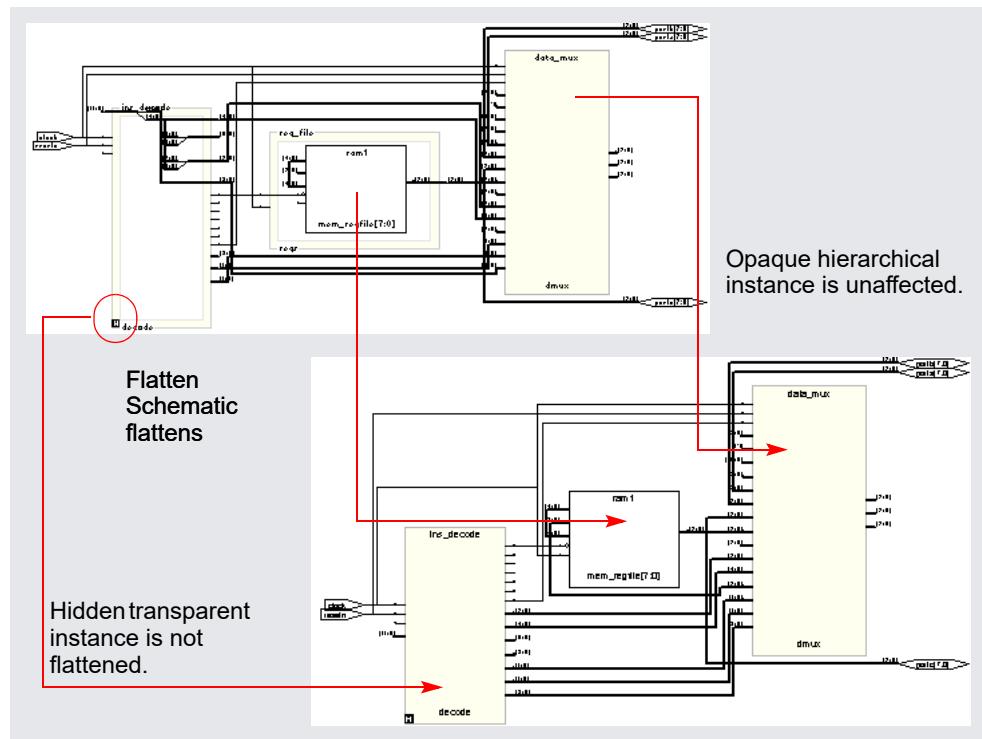
The software flattens the top-level design and displays it in a new window. To return to the top-level design, right-click and select Unflatten Schematic.

Unless you really require the entire design to be flattened, use Push/Pop mode and the filtering commands ([Filtering Schematics, on page 315](#)) to view the hierarchy. Alternatively, you can use one of the selective flattening techniques described in subsequent steps.

- To selectively flatten transparent instances when you analyze critical paths or use the Expand commands, select Flatten Current Schematic from the HDL Analyst menu, or select Flatten Schematic from the right-click popup menu.

The software generates a new view of the current schematic in the same window, with all transparent instances at the current level and below flattened. RTL schematics are flattened down to generic logic cells and Technology views down to technology primitives. To control the number of hierarchical levels that are flattened, use the Dissolve Instances command described in step 4.

If your view only contains hidden hierarchical instances or pale yellow (opaque) hierarchical instances, nothing is flattened. If you flatten an unfiltered (usually the top-level design) view, the software flattens all hierarchical instances (transparent and opaque) at the current level and below. The following figure shows flattened transparent instances.



Because the flattened view is a new view, you cannot use Back to return to the unflattened view or the views before it. Use Unflatten Schematic to return to the unflattened top-level view.

3. To selectively flatten the design by hiding instances, select hierarchical instances whose hierarchy you do not want to flatten, right-click, and select Hide Instances. Then flatten the hierarchy using one of the Flatten commands described above.

Use this technique if you want to flatten most of your design. If you want to flatten only part of your design, use the approach described in the next step.

When you hide instances, the software generates a new view where the hidden instances are not flattened, but marked with an H in the lower left corner. The rest of the design is flattened. If unhidden hierarchical instances are not flattened by this procedure, use the Flattened View or Flattened to Gates View commands described in step 1 instead of the Flatten Current Schematic command described in step 2, which only flattens transparent instances in filtered views.

You can select the hidden instances, right-click, and select Unhide Instances to make their hierarchy accessible again. To return to the unflattened top-level view, right-click in the schematic and select Unflatten Schematic.

4. To selectively flatten some hierarchical instances in your design by dissolving them, do the following:

- If you want to flatten more than one level, select Options->HDL Analyst Options and change the value of Dissolve Levels. If you want to flatten just one level, leave the default setting.
- Select the instances to be flattened.
- Right-click and select Dissolve Instances.

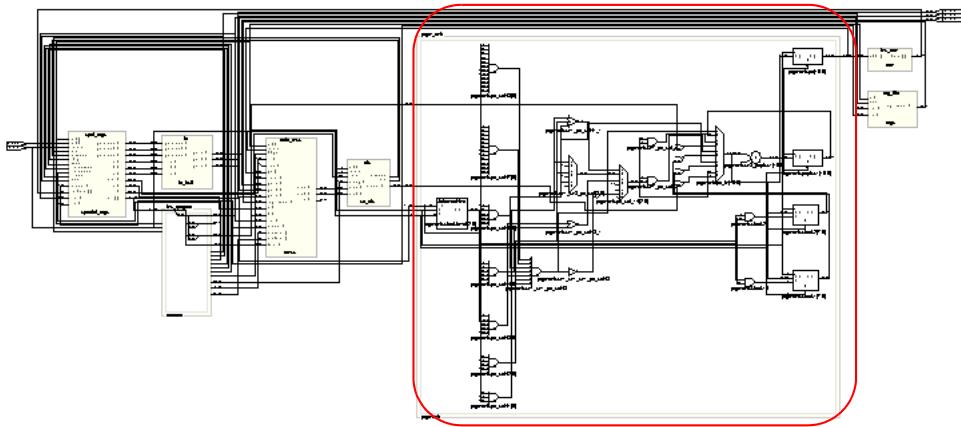
The results differ slightly, depending on the kind of view from which you dissolve instances.

**Starting View Software Generates a...**

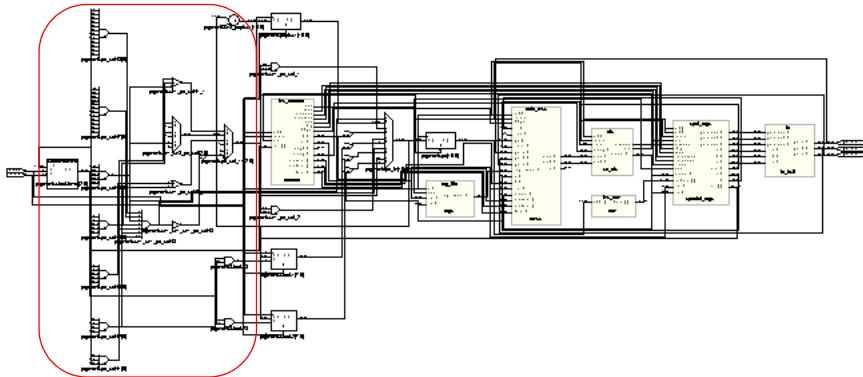
Filtered	Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of <a href="#">Viewing Design Hierarchy and Context, on page 312</a> . Use the Back button to return to the undissolved view.
Unfiltered	New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. Select Unflatten Schematic to return to the top-level unflattened view. You cannot use the Back button to return to previous views because this is a new view.

The following figure illustrates this.

Dissolved logic for prgmcntr shown nested when started from filtered view



Dissolve



Use this technique if you only want to flatten part of your design while retaining the hierarchical context. If you want to flatten most of the design, use the technique described in the previous step. Instead of dissolving instances, you can use a combination of the filtering commands and Push/Pop mode.

## Minimizing Memory Usage While Analyzing Designs

When working with large hierarchical designs, use the following techniques to use memory resources efficiently.

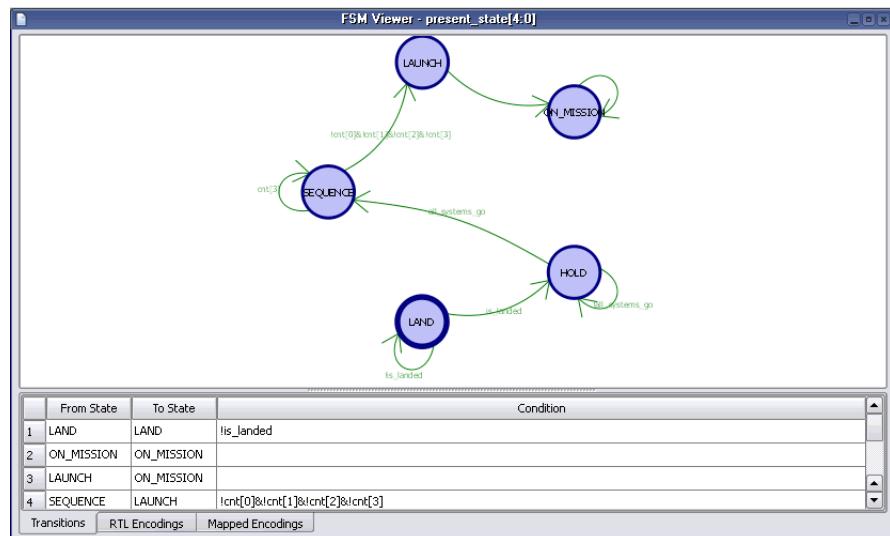
- Before you do any analysis operations such as searching, flattening, expanding, or pushing/popping, hide (HDL Analyst->Hide Instances) the hierarchical instances you do not need. This saves memory resources, because the software does not load the hierarchy of the hidden instances.
- Temporarily divide your design into smaller working files. Before you do any analysis, hide the instances you do not need. Save the design. The srs and srm files generated are smaller because the software does not save the hidden hierarchy. Close any open HDL Analyst windows to free all memory from the large design. In the Implementation Results view, double-click one of the smaller files to open the RTL or Technology schematic. Analyze the design using the smaller, working schematics.
- Filter your design instead of flattening it. If you must flatten your design, hide the instances whose hierarchy you do not need before flattening, or use the Dissolve Instances command. See [Flattening Schematic Hierarchy, on page 322](#) for details. For more information on the Expand Paths and Isolate Paths commands, see [RTL and Technology Views Popup Menus, on page 428](#) of the *Reference Manual*.
- When searching your design, search by instance rather than by net. Searching by net loads the entire design, which uses memory.
- Limit the scope of a search by hiding instances you do not need to analyze. You can limit the scope further by filtering the schematic in addition to hiding the instances you do not want to search.

# Using the FSM Viewer (Standard)

The FSM viewer displays state transition bubble diagrams for FSMs in the design, along with additional information about the FSM. You can use this viewer to view state machines implemented by the FSM Compiler. For more information, see [Running the FSM Compiler, on page 408](#) and [Inserting Probes, on page 413](#), respectively.

1. To start the FSM viewer, open the RTL view and either
  - Select the FSM instance, click the right mouse button and select View FSM from the popup menu.
  - Push down into the FSM instance (Push/Pop icon).

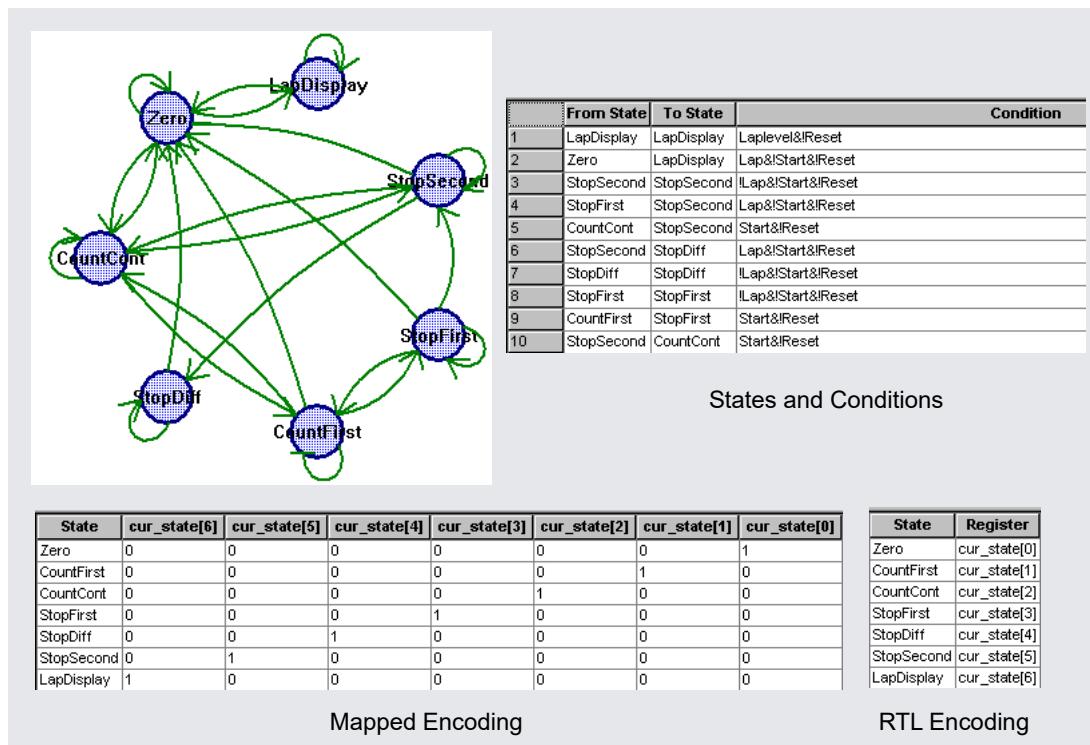
The FSM viewer opens. The viewer consists of a transition bubble diagram and a table for the encodings and transitions. If you used Verilog to define the FSMs, the viewer displays binary values for the state machines if you defined them with the ‘define’ keyword, and actual names if you used the parameter keyword.



2. The following table summarizes basic viewing operations.

To view...	Do...
from and to states, and conditions for each transition	Click the Transitions tab at the bottom of the table.
the correspondence between the states and the FSM registers in the RTL view	Click the RTL Encoding tab.
the correspondence between the states and the registers in the Technology View	Click the Mapped Encodings tab (available after synthesis).
only the transition diagram without the table	Select View->FSM table or click the FSM Table icon. You might have to scroll to the right to see it.

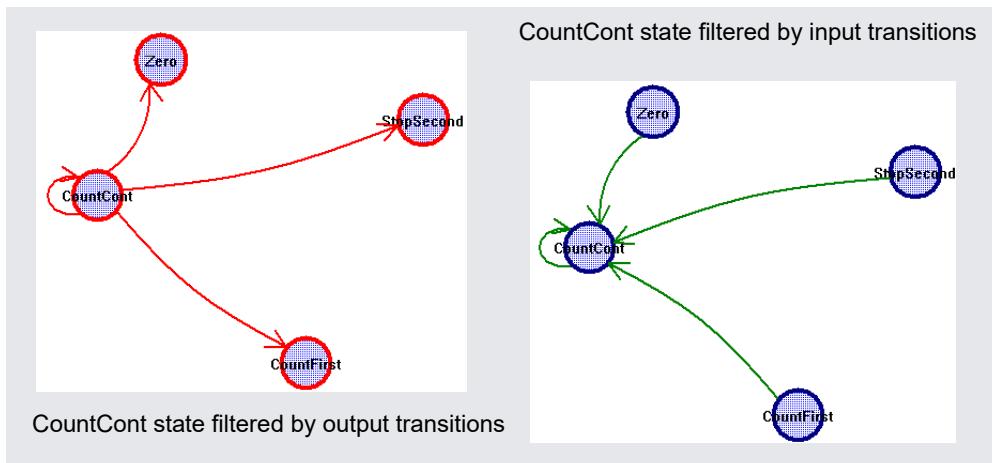
This figure shows you the mapping information for a state machine. The Transitions tab shows you simple equations for conditions for each state. The RTL Encodings tab has a State column that shows the state names in the source code, and a Registers column for the corresponding RTL encoding. The Mapped Encoding tab shows the state names in the code mapped to actual values.



3. To view just one selected state,

- Select the state by clicking on its bubble. The state is highlighted.
- Click the right mouse button and select the filtering criteria from the popup menu: output, input, or any transition.

The transition diagram now shows only the filtered states you set. The following figure shows filtered views for output and input transitions for one state.



Similarly, you can check the relationship between two or more states by selecting the states, filtering them, and checking their properties.

4. To view the properties for a state,

- Select the state.
- Click the right mouse button and select Properties from the popup menu. A form shows you the properties for that state.

To view the properties for the entire state machine like encoding style, number of states, and total number of transitions between states, deselect any selected states, click the right mouse button outside the diagram area, and select Properties from the popup menu.

5. To view the FSM description in text format, select the state machine in the RTL view and View FSM Info File from the right mouse popup. This is an example of the FSM Info File, *statemachine.info*.

```

State Machine: work.Control(verilog)-cur_state[6:0]
No selected encoding - Synplify will choose
Number of states: 7
Number of inputs: 4
Inputs:
  0: Laplevel
  1: Lap
  2: Start
  3: Reset
Clock: Clk

```

```
Transitions: (input, start state, destination state)
-100 S0 S6
--10 S0 S2
---1 S0 S0
-00- S0 S0
--10 S1 S3
-100 S1 S2
-000 S1 S1
---1 S1 S0
--10 S2 S5
-000 S2 S2
-100 S2 S1
---1 S2 S0
-100 S3 S5
-000 S3 S3
--10 S3 S1
---1 S3 S0
-000 S4 S4
--1- S4 S0
-1-- S4 S0
---1 S4 S0
-000 S5 S5
-100 S5 S4
--10 S5 S2
---1 S5 S0
1--0 S6 S6
---1 S6 S0
0--- S6 S0
```



## CHAPTER 8

# Analyzing Timing

---

This chapter describes typical analysis tasks. It describes graphical analysis with the HDL Analyst tool as well as interpretation of the text log file. It covers the following:

- [Analyzing Timing in Schematic Views, on page 336](#)
- [Generating Custom Timing Reports with STA, on page 344](#)
- [Using Analysis Design Constraints, on page 347](#)
- [Using Auto Constraints, on page 354](#)

# Analyzing Timing in Schematic Views

You can use the HDL Analyst and Timing Analyst functionality to analyze timing. This section describes the following:

- [Viewing Timing Information](#), on page 336
- [Annotating Timing Information in the Schematic Views](#), on page 337
- [Analyzing Clock Trees in the RTL View](#), on page 339
- [Viewing Critical Paths](#), on page 339
- [Handling Negative Slack](#), on page 342
- [Generating Custom Timing Reports with STA](#), on page 344

## Viewing Timing Information

Some commands, like Show Critical Path, Hierarchical Critical Path, Flattened Critical Path, automatically enable Show Timing Information and display the timing information. The following procedure shows you how to do so manually.

1. To analyze timing, enable HDL Analyst->Show Timing Information.

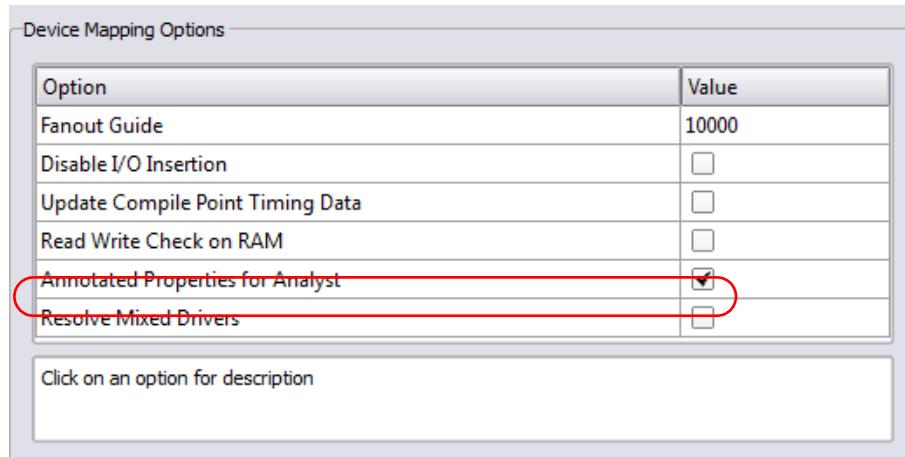
This displays the timing numbers for all instances in a Technology view. It shows the following:

<b>Delay</b>	This is the first number displayed. <ul style="list-style-type: none"><li>• Combinational logic This first number is the cumulative path delay to the output of the instance, which includes the net delay of the output.</li><li>• Flip-flops This first number is the path delay attributed to the flip-flop. The delay can be associated with either the input or output path, whichever is worse, because the flip-flop is the end of one path and the start of another.</li></ul>
<b>Slack Time</b>	This is the second number, and it is the slack time of the worst path that goes through the instance. A negative value indicates that timing constraints can not be met.

## Annotating Timing Information in the Schematic Views

You can annotate the schematic views with timing information for the components in the design. Once the design is annotated, you can search for these properties and their associated instances.

On the Device tab of the Implementation Options dialog box, enable Annotated Properties for Analyst.



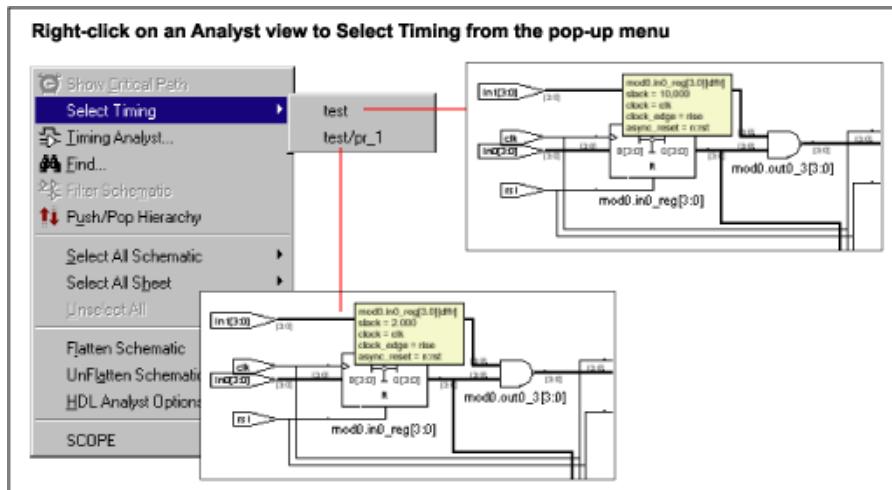
For each synthesis implementation and each place-and-route implementation, the tool generates properties and stores them in two files located in the project folder:

- .sap Synplify Annotated Properties  
Contains the annotated design properties generated after compilation, like clock pins.
- .tap Timing Annotated Properties  
Contains the annotated timing properties generated after compilation.

2. To view the annotated timing, open an RTL or Technology view.
3. To view the timing information from another associated implementation, do the following:

- Open an RTL or Technology view. It displays the timing information for that implementation.
- Select HDL Analyst->Select Timing, and select another implementation from the list. The list contains the main implementation and all associated place-and-route implementations. The timing numbers in the current Analyst view change to reflect the numbers from the selected implementation.

In the following example, an RTL View shows timing data from the test implementation and the test/pr\_1 (place and route) implementation.



- Once you have annotated your design, you can filter searches using these properties with the find command.
  - Use the `find -filter {@propName}>=propValue` command for the searches. See [find -filter, on page 127](#) in the *Reference Manual* for a list of properties. For information about the find command, see [find, on page 119](#) in the *Command Reference Manual*.
  - Precede the property name with the @ symbol.

For example to find fanouts larger than 60, specify `find -filter {@fanout}>=60`.

## Analyzing Clock Trees in the RTL View

To analyze clock trees in the RTL view, do the following:

1. In the Hierarchy Browser, expand **Clock Tree**, select all the clocks, and filter the design.

The Hierarchy Browser lists all clocks and the instances that drive them under **Clock Tree**. The filtered view shows the selected objects.

2. If necessary, use the filter and expand commands to trace clock connections back to the ports and check them.

For details about the commands for filtering and expanding paths, see [Filtering Schematics, on page 315](#), [Expanding Pin and Net Logic, on page 318](#) and [Expanding and Viewing Connections, on page 321](#).

3. Check that your defined clock constraints cover the objects in the design.

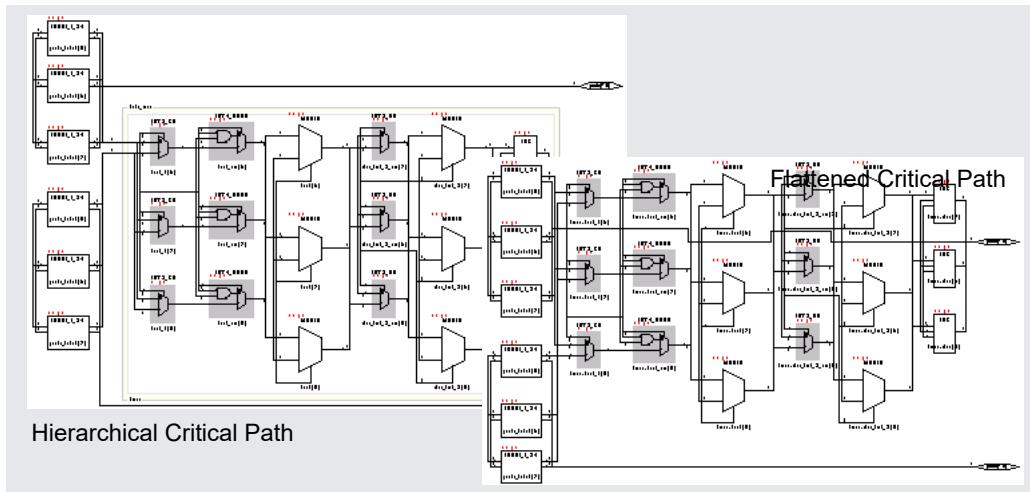
If you do not define your clock constraints accurately, you might not get the best possible synthesis optimizations.

## Viewing Critical Paths

The HDL Analyst tool makes it simple to find and examine critical paths and the relevant source code. The following procedure shows you how to filter and analyze a critical path. You can also use the procedure described in [Generating Custom Timing Reports with STA, on page 344](#) to view this and other paths.

1. If needed, set the slack time for your design.
  - Select **HDL Analyst->Set Slack Margin**.
  - To view only instances with the worst-case slack time, enter a zero.
  - To set a slack margin range, type a value for the slack margin, and click **OK**. The software gets a range by subtracting this number from the slack time, and the Technology view displays instances within this range. For example, if your slack time is -10 ns, and you set a slack margin of 4 ns, the command displays all instances with slack times between -6 ns and -10 ns. If your slack margin is 6 ns, you see all instances with slack times between -4 ns and -10 ns.

2. Display the critical path using one of the following methods. The Technology view displays a hierarchical view that highlights the instances and nets in the most critical path of your design.
- To generate a hierarchical view of the critical path, click the Show Critical Path icon (stopwatch icon (⌚)), select HDL Analyst->Technology->Hierarchical Critical Path, or select the command from the popup menu. This is a filtered view in the same window, with hierarchical logic shown in transparent instances. History commands apply, so you can return to the previous view by clicking Back.
  - To flatten the hierarchical critical path described above, right-click and select Flatten Schematic. The software generates a new view in the current window, and flattens only the transparent instances needed to show the critical path; the rest of the design remains hierarchical. Click Back to go to the top-level design.
  - To generate a flattened critical path in a new window, select HDL Analyst->Technology->Flattened Critical Path. This command uses more memory because it flattens the entire design and generates a new view for the flattened critical path in a new window. Click Back in this window to go to the flattened top-level design or to return to the previous window.



3. Use the timing numbers displayed above each instance to analyze the path. If no numbers are displayed, enable HDL Analyst->Show Timing Information. Interpret the numbers as follows:

**Delay**

For combinational logic, it is the cumulative delay to the output of the instance, including the net delay of the output. For flip-flops, it is the portion of the path delay attributed to the flip-flop. The delay can be associated with either the input path or output path, whichever is worse, because the flip-flop is the end of one path and the start of another.

**Slack time**

Slack of the worst path that goes through the instance. A negative value indicates that timing has not been met.

:8.8, 1.2

4. View instances in the critical path that have less than the worst-case slack time. For additional information on handling slack times, see [Handling Negative Slack, on page 342](#).  
If necessary change the slack margin and regenerate the critical path.
5. Crossprobe and check the RTL view and source code. Analyze the code and the schematic to determine how to address the problem. You can add more constraints or make code changes.
6. Click the Back icon to return to the previous view. If you flattened your design during analysis, select Unflatten Schematic to return to the top-level design.

There is no need to regenerate the critical path, unless you flattened your design during analysis or changed the slack margin. When you flatten your design, the view is regenerated so the history commands do not apply and you must click the Critical Path icon again to see the critical path view.

7. Rerun synthesis, and check your results.

If you have fixed the path, the window displays the next most critical path when you click the icon.

Repeat this procedure and fix the design for the remaining critical paths. When you are within 5-10 percent of your desired results, place and route your design to see if you meet your goal. If so, you are done. If your vendor provides timing-driven place and route, you might improve your results further by adding timing constraints to place and route.

## Handling Negative Slack

Positive slack time values (greater than or equal to 0 ns) are good, while negative slack time values (less than 0 ns) indicate the design has not met timing requirements. The negative slack value indicates the amount by which the timing is off because of delays in the critical paths of your design.

The following procedure shows you how to add constraints to correct negative slack values. Timing constraints can improve your design by 10 to 20 percent.

1. Display the critical path in a filtered Technology view.

- For a hierarchical critical path, either click the Critical Path icon, select HDL Analyst->Show Critical Path, or select HDL Analyst->Technology->Hierarchical Critical Path.
  - For a flat path, select HDL Analyst->Technology->Flattened Critical Path.
2. Analyze the critical path.
- Check the end points of the path. The start point can be a primary input or a flip-flop. The end point can be a primary output or a flip-flop.
  - Examine the instances. Use the commands described in [Expanding Pin and Net Logic, on page 318](#) and [Expanding and Viewing Connections, on page 321](#). For more information on filtering schematics, see [Filtering Schematics, on page 315](#).
3. Determine whether there is a timing exception, like a false or multicycle path. If this is the cause of the negative slack, set the appropriate timing constraint.
- If there are fewer start points, pick a start point to add the constraint. If there are fewer end points, add the constraint to an end point.
4. If your design does not meet timing by 20 percent or more, you may need to make structural changes. You could do this by doing either of the following:
- Enabling options like pipelining ([Pipelining, on page 384](#)), retiming ([Retiming, on page 388](#)), or resource sharing ([Sharing Resources, on page 406](#)).
  - Modifying the source code.
5. Rerun synthesis and check your results.

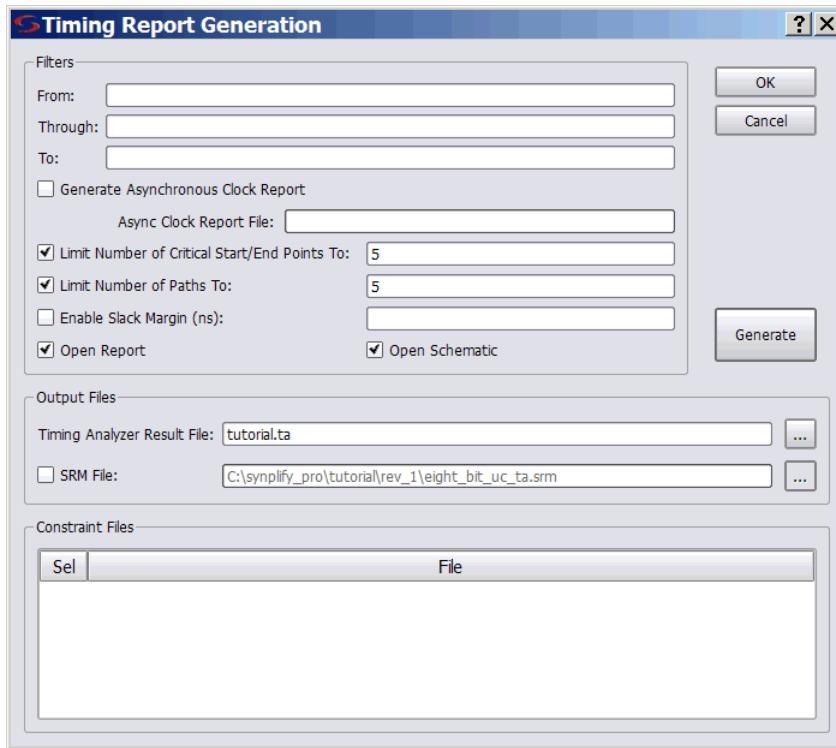
# Generating Custom Timing Reports with STA

The log file generated after synthesis includes a timing report and default timing information. Use the stand-alone timing analyst (STA) when you need to generate a customized timing report (`ta`) for the following situations:

- You need more details about a specific path.
- You want results for paths other than the top five timing paths (log file default).
- You want to modify constraints and analyze, without resynthesizing. See [Using Analysis Design Constraints, on page 347](#) for details.

The following procedure shows you how to generate a custom report:

1. Select Analysis->Timing Analyst or click on the Timing Analyst icon().
2. Fill in the parameters.
  - You can type in the from/to or through points, or you can cut and paste or drag and drop valid objects from the Technology view (not the RTL view) into the fields. See [Timing Report Generation Parameters, on page 361](#) in the *Command Reference Manual* for details on timing analysis parameters and how they can be filtered.
  - Set options for clock reports as needed.
  - Specify a name for the output timing report (`ta`).



3. Click Generate to run the report.

The software generates a custom report file called *projectName.ta*, located in the implementation directory (the directory you specified for synthesis results). The software also generates a corresponding output netlist file, with an *srm* extension.

4. Analyze results.

- View the report (Open Report) in the Text Editor. The following figure is a sample report showing analysis results based on maximum delay for the worst paths.

```
##### START OF TIMING REPORT #####
# Timing Report written on Mon Jun 05 11:05:33 2006
#
Top view:          sw
Requested Frequency: 10.0 MHz
Wire load mode:    top
Paths requested:   8
from:              i:svmult2o[11]
Constraint File(s): None

Worst From-To Path Information
*****
Path information for path number 1:
  Requested Period:      100.000
  - Setup time:           0.245
  = Required time:        99.755
  - Propagation time:    2.215
  = Slack :              97.540
Number of logic level(s): 12
Starting point:          svmult2o[11] / regout
Ending point:             o_o[12] / datain
The start point is clocked by clk_i [rising] on pin clk
The end point is clocked by clk_i [rising] on pin clk

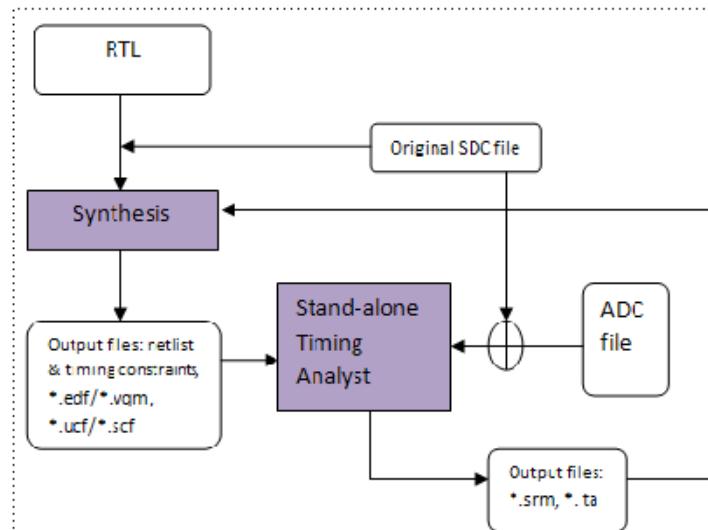
Instance / Net          Pin       Pin      Arrival    No. of
Name       Type     Name     Dir     Delay   Time     Fan Out(s)
-----|-----|-----|-----|-----|-----|-----|-----|
svmult2o[11]  stratixii_lcell_ff  regout  Out    0.094  0.094  -
svmult2o[11]  Net                  -       -      0.311  -        2
svmult1_carry_9  stratixii_lcell_comb  dataaf In    -     ...  0.405  -
```

- View the netlist (View Critical Path) in a Technology view. This Technology view, labeled Timing View in the title bar, shows only the paths you specified in the Timing Analyst dialog box. Note that the Timing Analyst and Show Critical Path commands (and equivalent icons and shortcuts) are disabled whenever the Timing View is active.

# Using Analysis Design Constraints

Besides generating custom timing reports (see [Generating Custom Timing Reports with STA, on page 344](#)), you can also use the Stand-alone Timing Analyst to create constraints in an adc file. You can use these constraints to experiment with different timing values, or to add or modify timing constraints.

The advantage to using analysis design constraints (ADC) is that you do not have to resynthesize the whole design. This reduces debugging time because you can get a quick estimate, or try out different values. The Standalone Timing Analyst (STA) puts these constraints in an Analysis Design Constraints file (adc). The process for using this file is summarized in the following flow diagram:



See the following for details:

- [Scenarios for Using Analysis Design Constraints, on page 348](#)
- [Creating an ADC File, on page 349](#)
- [Using Object Names Correctly in the adc File, on page 353](#)

## Scenarios for Using Analysis Design Constraints

The following describe situations where you can effectively use adc constraints to debug, explore options or modify constraints. For details about creating these constraints, see [Creating an ADC File, on page 349](#).

- What-if analysis of design performance

If your design meets the target frequency, you can use adc constraints to analyze higher target frequencies, or analyze performance of a module in a different design/technology/target device.

- Constraints on enable registers

Similarly, you can apply syn\_reference\_clock on enable registers to analyze if the enables have a regular pattern like clock, or if they operate on a frequency other than clock. For example:

```
FDC  create_clock {clk} -name {clk} -freq 100 -clockgroup  
      clk_grp_0
```

```
ADC  define_attribute {n:en} syn_reference_clock {clk2}  
      create_clock {clk2} -name {clk2} -freq 50 -clockgroup  
      clk_grp_1
```

---

- Adding additional timing exceptions

When you analyze the results of the first synthesis run, you often find functional or clock-to-clock timing exceptions, and you can handle these with adc constraints. For example:

- Applying false paths on synchronization circuitry
- Adding false paths between clocks belonging to different clock groups

You must add these constraints to see more critical paths in the design. The adc constraints let you add these constraints on the fly, and helps you debug designs faster.

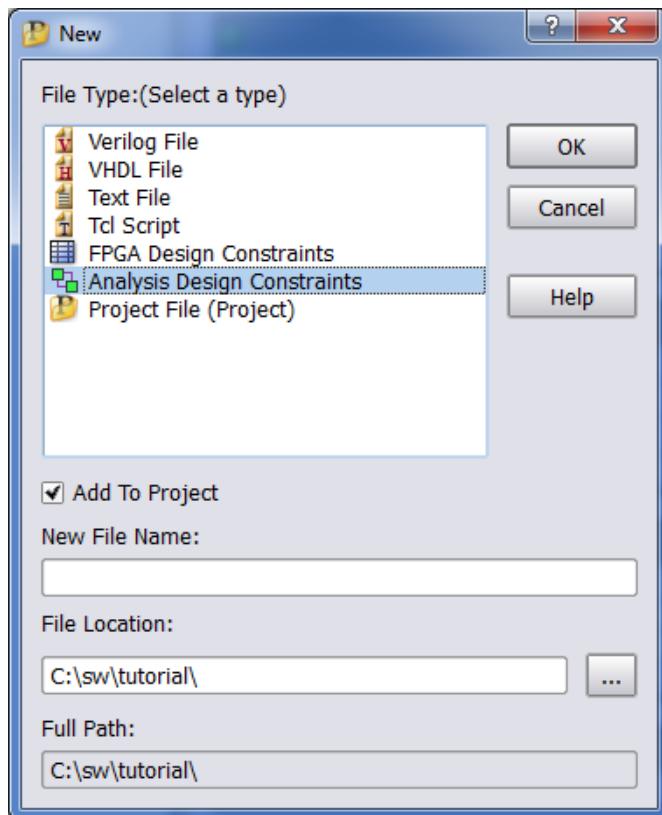
- Modifying timing exceptions that were previously applied

For example you might want to set a multicycle path constraint for a path that was defined as a false path in the constraint file or vice versa. To modify the timing exception, you must first ignore or reset the timing exception that was set in the constraint file, as described in [Using Analysis Design Constraints, on page 347](#), step 3.

## Creating an ADC File

The following procedure explains how to create an adc file.

1. Select File->New.
2. Do the following in the dialog box that opens:
  - Select Analysis Constraint File.



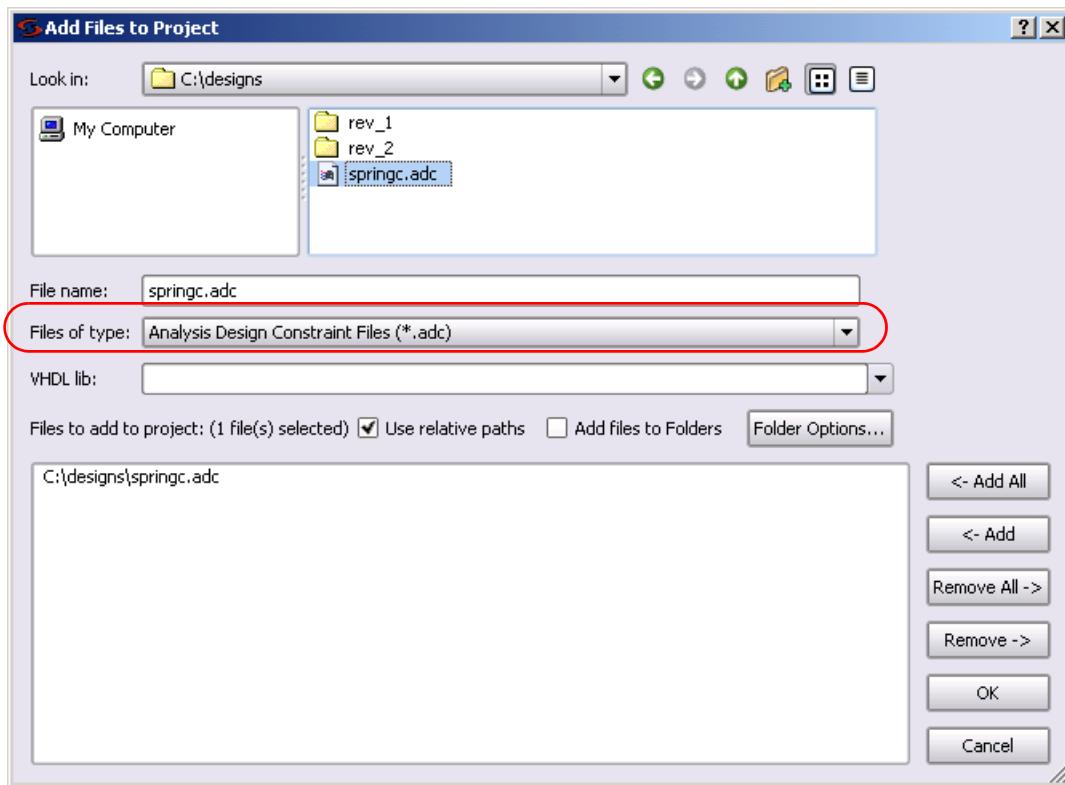
- Type a name and location for the file. The tool automatically assigns the adc extension to the filename.
- Enable Add to Project, and click OK. This opens the text editor where you can specify the new constraints.

3. Type in the constraints you want and save the file. Remember the following when you enter the constraints:

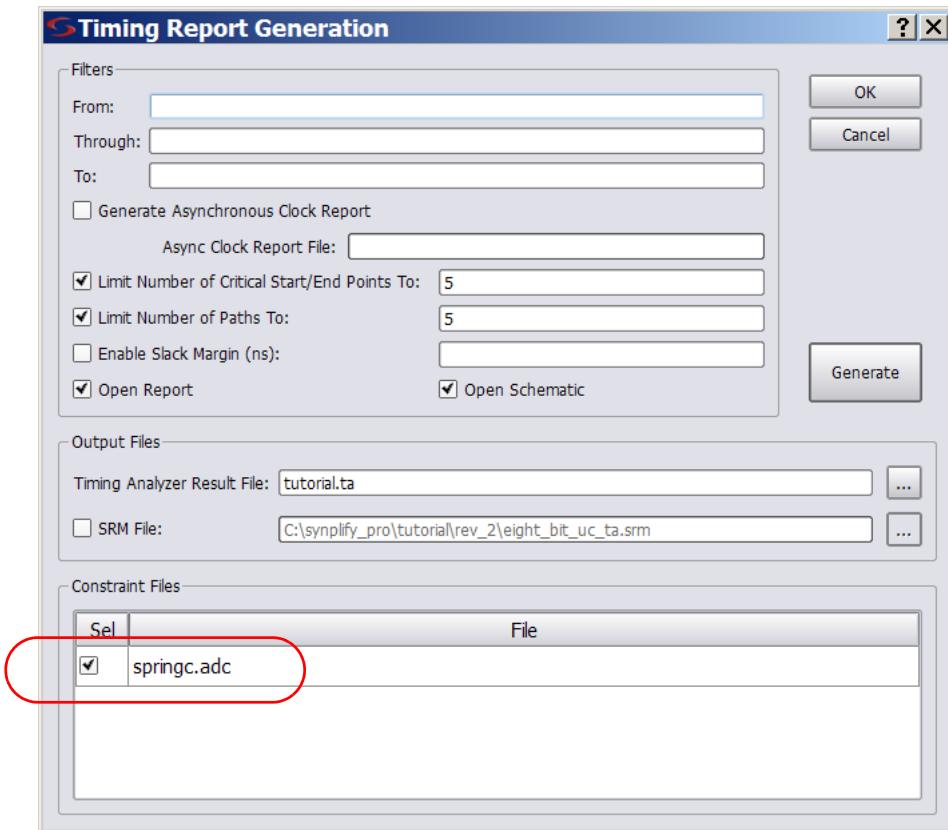
- Keep in mind that the original fdc file has already been applied to the design. Any timing exception constraints in this file must not conflict with constraints that are already in effect. For example, if there is a conflict when multiple timing exceptions (false path, path delay, and multicycle timing constraints) are applied to the same path, the tool uses this order to resolve conflicts: false path, multicycle path, max delay. See [Conflict Resolution for Timing Exceptions, on page 187](#) for details about how the tool prioritizes timing exceptions.
- The object names must be mapped object names, so use names from the Technology view, not names from the RTL view. Unlike the constraint file (RTL view), the adc constraints apply to the mapped database because the database is not remapped with this flow. For more information, see [Using Object Names Correctly in the adc File, on page 353](#).
- If you want to modify an existing constraint for a timing exception, you must first reset the original fdc constraint, and then apply the new constraint. In the following example the multicycle path constraint was changed to 3:

Original FDC	set_multicycle_path -to [get_cells{a_reg*}] 2
ADC	reset_path -to {get_cells{a_reg*}} set_multicycle_path -to [get_cells{a_reg*}] 3

- When you are done, save and close the file. This adds the file to your project.



- You can create multiple adc files for different purposes. For example, you might want to keep timing exception constraints, I/O constraints, and clock constraints in separate files. If you have an existing adc file, use the Add File command to add this file to your project. Select Analysis Design Constraint Files (\*.adc) as the file type.
4. Run timing analysis.
- Select Analysis->Timing Analyst or click the Timing Analyst icon (). The Timing Analyst window will look like the example below, with pointers to the srm file, the original fdc and the new adc files you created.



- If you have multiple adc files, enable the ones you want.
- If you have a previous run and want to save that report, type a new name for the output ta file. If you do not specify a name, the tool overwrites the previous report.
- Fill in other parameters as appropriate, and click Generate.

The tool runs static timing analysis in the same implementation directory as the original implementation. The tool applies the adc constraints on top of the fdc constraints. Therefore, adc constraints affect timing results only if there are no conflicts with fdc constraints.

The tool generates a timing report called \*\_adc.ta and an \*\_adc.srm file by default. It does not change any synthesis outputs, like the output netlist or timing constraints for place and route.

5. Analyze the results in the timing report and \*\_adc.srm file.
6. If you need to resynthesize after analysis, add the adc constraints as an fdc file to the project and rerun synthesis.

## Using Object Names Correctly in the adc File

Constraints and collections applied in the constraint file reference the RTL-level database. Synthesis optimizations such as retiming and replication can change object names during mapping because objects may be merged.

The standalone timing analyst does not map objects. It just reads the gate-level object names from the post-mapping database; this is reflected in the Technology view. Therefore, you must define objects either explicitly or with collections from the Technology view when you enter constraints into the adc file. Do not use RTL names when you create these constraints (see [Creating an ADC File, on page 349](#) for details of that process).

### Example

Assume that register en\_reg is replicated during mapping to reduce fanout. Further, registers en\_reg and en\_reg\_rep2 connect to register dataout[31:0]. In this case, if you define the following false path constraint in the adc file, then the standalone timing analyzer does not automatically treat paths from the replicated register en\_reg\_rep2 as false paths.

```
set_false_path -from {{i:en_reg}} -to {{i:dataout[31:0]}}
```

Unlike constraints in the fdc file, you must specify this replicated register explicitly or as a collection. Only then are all paths properly treated as false paths. So in this example, you must define the following constraints in the adc file:

```
set_false_path -from {{i:en_reg}} -to {{i:dataout[31:0]}}  
set_false_path -from {{i:en_reg_rep2}}  
-to {{i:dataout[31:0]}}
```

or

```
define_scope_collection en_regs {find -seq {i:en_reg*}  
-filter (@name == en_reg || @name == en_reg_rep2)}  
set_false_path -from {{$en_regs}} -to {{i:dataout[31:0]}}
```

## Using Auto Constraints

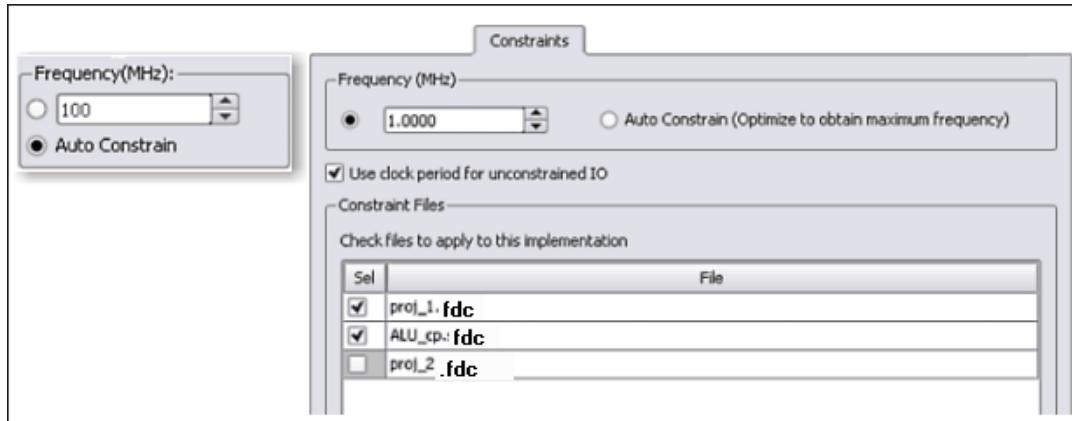
Auto constraining lets you synthesize with automatic constraints as a first step to get an idea of what you can achieve. Automatic constraints generate the fastest design implementation, so they force the timing engine to work harder. Based on the results from auto-constraining, you can refine the constraints manually later. For an explanation of how auto constraints work, see [Results of Auto Constraints, on page 356](#).

1. To automatically constrain your design, first do the following:
  - Set your device to a technology that supports auto-constraining. With supported technologies, the Auto Constrain button under Frequency in the Project view is available.



- Do not define any clocks. If you define clocks using the SCOPE window or a constraint file, or set the frequency in the Project view, the software uses the user-defined `create_clock` constraints instead of auto constraints.

- Make sure any multi-cycle or false path constraints are specified on registers.
2. Enable the Auto Constrain button on the left side of the Project view. Alternatively, select Project->Implementation Options->Constraints, and enable the Auto Constrain option there.



3. If you want to auto constrain I/O paths, select Project->Implementation Options->Constraints and enable Use Clock Period for Unconstrained IO.

If you do not enable this option, the software only auto constrains flop-to-flop paths. Even when the software auto constrains the I/O paths, it does not generate these constraints for forward-annotation.

4. Synthesize the design.

The software puts each clock in a separate clock group and adjusts the timing of each clock individually. At different points during synthesis it adjusts the clock period of each clock to be a target percentage of the current clock period, usually 15% - 25%.

After the clocks, the timing engine constrains I/O paths by setting the default combinational path delay for each I/O path to be one clock period.

The software writes out the generated constraints in a file called `AutoConstraint_designName.sdc` in the run directory. It also forward-announces these constraints to the place-and-route tools.

5. Check the results in `AutoConstraint_designName.sdc` and the log file. To open the constraint file as a text file, right-click on the file in the Implementation Results view and select Open as Text.

The flop-to-flop constraints use syntax like the following:

```
create_clock -name {c:leon|clk} -period 13.327 -clockgroup  
Autoconstr_clkgroup_0 -rise 0.000 -fall 6.664 -route 0.000
```

6. You can now add this generated constraint file to the project and rerun synthesis with these constraints.

## Results of Auto Constraints

This section contains information about the following:

- [Stages of the Auto Constrain Algorithm](#), on page 356
- [I/O Constraints and Timing Exceptions](#), on page 357
- [Reports and Forward-annotation](#), on page 357
- [Repeatability of Results](#), on page 358

## Stages of the Auto Constrain Algorithm

To auto constrain, do not define any clocks. When you enable the Auto Constrain option, the synthesis software goes through these stages:

1. It infers every clock in the design.
2. It puts each clock in its own clock group.
3. It invokes mapper optimizations in stages and generates the best possible synthesis results.
  - You should only use Auto Constrain early in the synthesis process to get a general idea of how fast your design runs. This option is not meant to be a substitute for declaring clocks.
4. For each clock, including the system clock, the software maintains a negative slack of between 15 and 25 percent of the requested frequency.

## I/O Constraints and Timing Exceptions

The auto constrain algorithm infers all the clocks, because none are defined. It handles the following timing situations as described below:

- I/O constraints

You can auto constrain I/O paths as well as flop-to-flop paths by selecting Project->Implementation Options->Constraints and enabling Use Clock Period for Unconstrained IO. The software does not write out these I/O constraints.

- Timing exceptions like multicycle and false paths

The auto constraint algorithm honors SCOPE multicycle and false path constraints that are specified as constraints on registers.

## Auto Constrain Limitations

The Auto Constrain feature over constrains designs with output critical paths.

## Reports and Forward-annotation

In the log file, the software reports the Requested and Estimated Frequency or Requested and Estimated Period and the negative slack for each clock it infers. The log file contains all the details.

The software also generates a constraint file in the run directory called `AutoConstraint_designName.sdc`, which contains the auto constraints generated. The following is an example of an auto constraint file:

```
#Begin clock constraint
create_clock -name {c:leon|clk} -period 13.327 -rise 0.000 -fall
6.664
#End clock constraint
```

The software forward-annotates the `create_clock` constraints, writing out the appropriate file for the place-and-route tool.

## Repeatability of Results

If you use the requested frequency resulting from the Auto constrain option as the requested frequency for a regular synthesis run, you might not get the same results as you did with auto constraints. This is because the software invokes the mapper optimizations in stages when it auto constrains. The results from a previous stage are used to drive the next stage. As the interim optimization results vary, there is no guarantee that the final results will stay the same.

## CHAPTER 9

# Inferring High-Level Objects

---

This chapter contains guidelines on how to structure your code or attach attributes so that the synthesis tools can automatically infer high-level objects like RAMs. See the following for more information:

- [Defining Black Boxes for Synthesis](#), on page 360
- [Defining State Machines for Synthesis](#), on page 370
- [Initializing RAMs](#), on page 374

# Defining Black Boxes for Synthesis

Black boxes are predefined components for which the interface is specified, but whose internal architectural statements are ignored. They are used as place holders for IP blocks, legacy designs, or a design under development.

This section discusses the following topics:

- Instantiating Black Boxes and I/Os in Verilog, on page 360
- Instantiating Black Boxes and I/Os in VHDL, on page 362
- Adding Black Box Timing Constraints, on page 365
- Adding Other Black Box Attributes, on page 368

For information about using black boxes with the Clock Conversion option, see [Working with Gated Clocks, on page 454](#).

## Instantiating Black Boxes and I/Os in Verilog

Verilog black boxes for macros and I/Os come from two sources: commonly-used or vendor-specific components that are predefined in Verilog macro libraries, or black boxes that are defined in another input source like a schematic. For information about instantiating black boxes in VHDL, see [Instantiating Black Boxes and I/Os in VHDL, on page 362](#). Additional information about black boxes can be found in [Working with Gated Clocks, on page 454](#).

The following process shows you how to instantiate both types as black boxes. Refer to the `installDirectory/examples` directory for examples of instantiations of low-level resources.

1. To instantiate a predefined Verilog module as a black box:
  - Select the library file with the macro you need from the `installDirectory/lib/technology` directory. Files are named `technology.v`. Most vendor architectures provide macro libraries that predefine the black boxes for primitives and macros.
  - Make sure the library macro file is the first file in the source file list for your project.

2. To instantiate a module that has been defined in another input source as a black box:
  - Create an empty macro that only contains ports and port directions.
  - Put the `syn_black_box` synthesis directive just before the semicolon in the module declaration.

```
module myram (out, in, addr, we) /* synthesis syn_black_box */;
    output [15:0] out;
    input [15:0] in;
    input [4:0] addr;
    input we;
endmodule
```

- Make an instance of the stub in your design.
- Compile the stub along with the module containing the instantiation of the stub.
- To simulate with a Verilog simulator, you must have a functional description of the black box. To make sure the synthesis software ignores the functional description and treats it as a black box, use the `translate_off` and `translate_on` constructs. For example:

```
module adder8(cout, sum, a, b, cin);
// Code that you want to synthesize
/* synthesis translate_off */
// Functional description.
/* synthesis translate_on */
// Other code that you want to synthesize.
endmodule
```

3. To instantiate a vendor-specific (black box) I/O that has been defined in another input source:
  - Create an empty macro that only contains ports and port directions.
  - Put the `syn_black_box` synthesis directive just before the semicolon in the module declaration.
  - Specify the external pad pin with the `black_box_pad_pin` directive, as in this example:

```
module BBDSLHS(D,E,GIN,GOUT,PAD,Q)
    /* synthesis syn_black_box black_box_pad_pin="PAD" */

    – Make an instance of the stub in your design.
    – Compile the stub along with the module containing the instantiation
        of the stub.
```
4. Add timing constraints and attributes as needed. See [Adding Black Box Timing Constraints, on page 365](#) and [Adding Other Black Box Attributes, on page 368](#).
5. After synthesis, merge the black box netlist and the synthesis results file using the method specified by your vendor.

## Instantiating Black Boxes and I/Os in VHDL

VHDL black boxes for macros and I/Os come from two sources: commonly-used or vendor-specific components that are predefined in VHDL macro libraries, or black boxes that are defined in another input source like a schematic. For information about instantiating black boxes in VHDL, see [Instantiating Black Boxes and I/Os in Verilog, on page 360](#).

The following process shows you how to instantiate both types as black boxes. Refer to the `installDirectory/examples` directory for examples of instantiations of low-level resources.

1. To instantiate a predefined VHDL macro (for a component or an I/O),
  - Select the library file with the macro you need from the `installDirectory/lib/vendor` directory. Files are named `family.vhd`. Most vendor architectures provide macro libraries that predefine the black boxes for primitives and macros.

- Add the appropriate library and use clauses to the beginning of your design units that instantiate the macros.

```
library family;
use family.components.all;
```

2. To create a black box for a component from another input source:

- Create a component declaration for the black box.
- Declare the `syn_black_box` attribute as a boolean attribute.
- Set the attribute to true.

```
library synplify;
use synplify.attributes.all;
entity top is
    port (clk, rst, en, data: in bit; q: out bit);
end top;

architecture structural of top is
component bbox
    port(Q: out bit; D, C, CLR: in bit);
end component;

attribute syn_black_box of bbox: component is true;
...
```

- Instantiate the black box and connect the ports.

```
begin
my_bbox: bbox port map (
    Q => q,
    D => data,
    C => clk,
    CLR => rst);
```

- To simulate with a VHDL simulator, you must have the functional description of a black box. To make sure the synthesis software ignores the functional description and treats it as a black box, use the `translate_off` and `translate_on` constructs. For example:

```
architecture behave of ram4 is
begin
    synthesis translate_off
    stimulus: process (clk, a, b)
        -- Functional description
    end process;
    synthesis translate_on

    -- Other source code you WANT synthesized
```

3. To create a vendor-specific (black box) I/O for an I/O defined in another input source:

- Create a component declaration for the I/O.
- Declare the `black_box_pad_pin` attribute as a string attribute.
- Set the attribute value on the component to be the external pin name for the pad.

```
library synplify;
use synplify.attributes.all;
...

component mybuf
    port(O: out bit; I: in bit);
end component;
attribute black_box_pad_pin of mybuf: component is "I";
```

- Instantiate the pad and connect the signals.

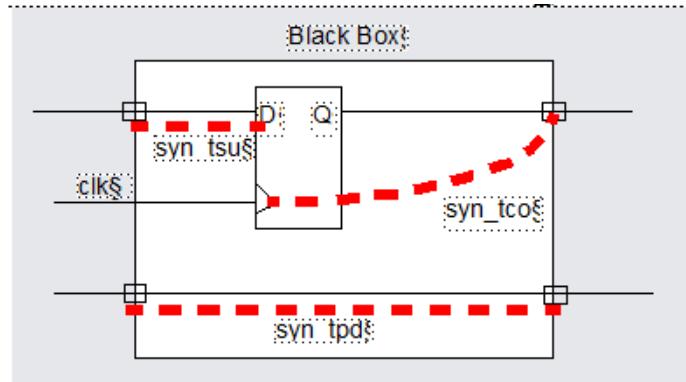
```
begin
data_pad: mybuf port map (
    O => data_core,
    I => data);
```

4. Add timing constraints and attributes. See [Adding Black Box Timing Constraints](#), on page 365, [Using Gated Clocks for Black Boxes](#), on page 476, and [Adding Other Black Box Attributes](#), on page 368.

## Adding Black Box Timing Constraints

A black box does not provide the software with any information about internal timing characteristics. You must characterize black box timing accurately, because it can critically affect the overall timing of the design. To do this, you add constraints in the source code or in the SCOPE interface.

You attach black box timing constraints to instances that have been defined as black boxes. There are three black box timing constraints, `syn_tpd`, `syn_tsu`, and `syn_tco`. There are additional attributes for black box pins and black boxes with gated clocks; see [Adding Other Black Box Attributes](#), on page 368 and [Using Gated Clocks for Black Boxes](#), on page 476.



1. Define the instance as a black box, as described in [Instantiating Black Boxes and I/Os in Verilog](#), on page 360 or [Instantiating Black Boxes and I/Os in VHDL](#), on page 362.
2. Determine the kind of constraint for the information you want to specify:

To define...	Use...
Propagation delay through the black box	<code>syn_tpd</code>
Setup delay (relative to the clock) for input pins	<code>syn_tsu</code>
Clock-to-output delay through the black box	<code>syn_tco</code>

3. In VHDL, use the following syntax for the constraints.

- Use the predefined attributes package by adding this syntax

```
library synplify;
use synplify.attributes.all;
```

In VHDL, you must use the predefined attributes package. For each directive, there are ten predeclared constraints in the attributes package, from *directive\_name1* to *directive\_name10*. If you need more constraints, declare the additional constraints using integers greater than 10. For example:

```
attribute syn_tcoll : string;
attribute syn_tcol2 : string;
```

- Define the constraints in either of these ways:

VHDL syntax	attribute <i>attributeName&lt;n&gt;</i> : "att_value"
----------------	---

---

Verilog-style notation	attribute <i>attributeName&lt;n&gt;</i> of <i>bbox_name</i> : component is "att_value"
---------------------------	---

---

The following table shows the appropriate syntax for *att\_value*. See the *Attribute Reference Manual* for complete syntax information.

<b>Attribute</b>	<b>Value Syntax</b>
<i>syn_tsu&lt;n&gt;</i>	<i>bundle -&gt; [!]clock = value</i>
<i>syn_tco&lt;n&gt;</i>	<i>[!]clock -&gt; bundle = value</i>
<i>syn_tpd&lt;n&gt;</i>	<i>bundle -&gt; bundle = value</i>

- <*n*> is a numerical suffix.
  - *bundle* is a comma-separated list of buses and scalar signals, with no intervening spaces. For example, A,B,C.
  - ! indicates (optionally) a negative edge for a clock.
  - *value* is in ns.
-

The following is an example of black box attributes, using VHDL signal notation:

```

architecture top of top is
component rcf16x4z port(
    ad0, ad1, ad2, ad3 : in std_logic;
    di0, di1, di2, di3 : in std_logic;
    wren, wpe : in std_logic;
    tri : in std_logic;
    do0, do1, do2 do3 : out std_logic;
end component

attribute syn_tpd1 of rcf16x4z : component is
    "ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is
    "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsul of rcf16x4z : component is
    "ad0,ad1,ad2,ad3 -> ck = 1.2";
attribute syn_tsu2 of rcf16x4z : component is
    "wren,wpe,do0,do1,do2,do3 -> ck = 0.0";

```

4. In Verilog, add the directives as comments, as shown in the following example. For explanations about the syntax, see the table in the previous step or the *Attribute Reference Manual*.

```

module ram32x4 (z, d, addr, we, clk)
    /* synthesis syn_black_box
    syn_tpd1="addr[3:0]->z[3:0]=8.0"
    syn_tsul="addr[3:0]->clk=2.0"
    syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule

```

5. To add black box attributes through the SCOPE interface, do the following:
  - Open the SCOPE spreadsheet and select the Attributes panel.
  - In the Object column, select the name of the black-box module or component declaration from the pull-down list. Manually prefix the black box name with **v:** to apply the constraint to the view.

- In the Attribute column, type the name of the timing attribute, followed by the numerical suffix, as shown in the following table. You cannot select timing attributes from the pull-down list.
- In the Value column, type the appropriate value syntax, as shown in the table in step 3.
- Save the constraint file, and add it to the project.

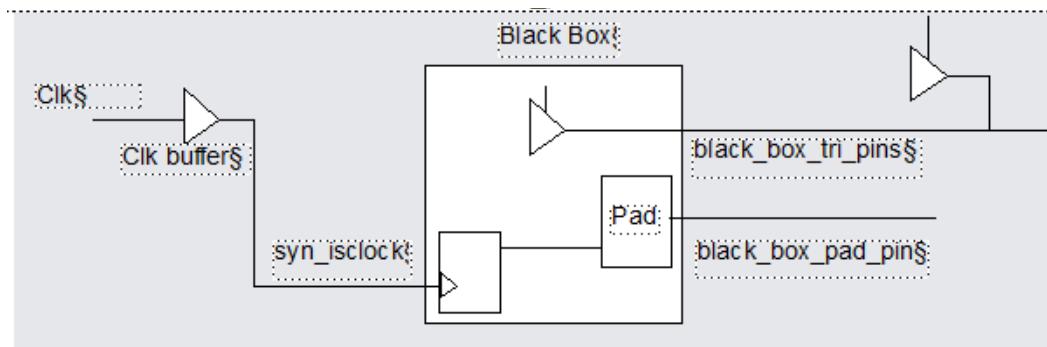
The resulting constraint file contains syntax like this:

```
define_attribute v:{blackboxModule} attribute<n> {attributeValue}
```

6. Synthesize the design and check black box timing.

## Adding Other Black Box Attributes

Besides black box timing constraints, you can also add other attributes to define pin types on the black box or define gated clocks. You cannot use the attributes for all technologies. Check the *Attribute Reference Manual* for details about which technologies are supported. For information about black boxes with gated clocks, see [Using Gated Clocks for Black Boxes, on page 476](#).



1. To specify that a clock pin on the black box has access to global clock routing resources, use `syn_isclock`.

Depending on the technology, different clock resources are inserted.

2. To specify that the software need not insert a pad for a black box pin, use `black_box_pad_pin`.

Use this for technologies that automatically insert pad buffers for the I/Os, like some GoWin technologies.

3. To define a tristate pin so that you do not get a mixed driver error when there is another tristate buffer driving the same net, use `black_box_tri_pins`.

# Defining State Machines for Synthesis

A finite state machine (FSM) is a piece of hardware that advances from state to state at a clock edge. The synthesis software recognizes and extracts the state machines from the HDL source code. For guidelines on setting up the source code, see the following:

- [Defining State Machines in Verilog](#), on page 370
- [Defining State Machines in VHDL](#), on page 371
- [Specifying FSMs with Attributes and Directives](#), on page 372

For information about the attributes used to define state machines, see [Running the FSM Compiler](#), on page 408.

## Defining State Machines in Verilog

The synthesis software recognizes and automatically extracts state machines from the Verilog source code if you follow the coding guidelines listed below. The software attaches the `syn_state_machine` attribute to each extracted FSM.

For alternative ways to define state machines, see [Defining State Machines for Synthesis](#), on page 370.

Follow these Verilog coding guidelines:

- In Verilog, model the state machine with `case`, `cased`, or `casez` statements in `always` blocks. Check the current state to advance to the next state and then set output values. Do not use `if` statements.
- Always use a default assignment as the last assignment in the `case` statement, and set the state variable to '`bx`'. This is a "don't care" statement and ensures that the software can remove unnecessary decoding and gates.
- Make sure the state machines have a synchronous or asynchronous reset to set the hardware to a valid state after power-up, or to reset the hardware when you are operating.
- Specify explicit state values for states with parameter or 'define' statements. This is an example of a parameter statement that sets the current state to `2'h2`:

```

parameter state1 = 2'h1, state2 = 2'h2;
...
current_state = state2;

```

This example shows how to set the current state value with `define statements:

```

`define state1 2'h1
`define state2 2'h2
...
current_state = `state2;

```

- Make state assignments using parameter with symbolic state names. Use parameter over `define, because `define is applied globally while parameter definitions are local. Local definitions make it easier to reuse common state names in multiple FSM designs, like RESET, IDLE, READY, READ, WRITE, ERROR, and DONE.

If you use `define to assign the names, you cannot reuse a state name because it has already been used in the global name space. To reuse the same names in this scenario, you have to use `undef and `define statements between modules to redefine the names. This method makes it difficult to probe the internal values of FSM state buses from a testbench and compare them to the state names.

## Defining State Machines in VHDL

The synthesis software recognizes and automatically extracts state machines from the VHDL source code if you follow the coding guidelines below. For alternative ways to define state machines, see [Defining State Machines in Verilog, on page 370](#) and [Specifying FSMs with Attributes and Directives, on page 372](#).

The following are VHDL guidelines for coding. The software attaches the syn\_state\_machine attribute to each extracted FSM.

- Use case statements to check the current state at the clock edge, advance to the next state, and set output values. You can also use if-then-else statements, but case statements are preferable.
- If you do not cover all possible cases explicitly, include a when others assignment as the last assignment of the case statement, and set the state vector to some valid state.

- If you create implicit state machines with multiple WAIT statements, the software does not recognize them as state machines.
- Make sure the state machines have a synchronous or asynchronous reset to set the hardware to a valid state after power-up, or to reset the hardware when you are operating.
- To choose an encoding style, attach the `syn_encoding` attribute to the enumerated type. The software automatically encodes your state machine with the style you specified.

## Specifying FSMs with Attributes and Directives

If your design has state machines, the software can extract them automatically with the FSM Compiler, or you can manually attach attributes to state registers to define them as state machines. See [Optimizing State Machines, on page 407](#) for information about automatic FSM extraction, and [Defining State Machines for Synthesis, on page 370](#) for other ways to specify FSMs.

The following steps show you how to manually attach attributes to define FSMs for extraction.

1. To determine how state machines are extracted, set attributes in the source code as shown in the following table:

To...	Attribute
Specify a state machine for extraction and optimization	<code>syn_state_machine=1</code>
Prevent state machines from being extracted and optimized	<code>syn_state_machine=0</code>
Prevent the state machine from being optimized away	<code>syn_preserve=1</code>

For information about how to add attributes, see [Specifying Attributes and Directives, on page 89](#).

2. To determine the encoding style for the state machine, set the `syn_encoding` attribute in the source code or in the SCOPE window. For VHDL users there are alternative methods, described in the next step.

The FSM Compiler honors this setting. The different values for this attribute are briefly described here:

Situation: If...	syn_encoding Value	Explanation
Area is important	sequential	One of the smallest encoding styles.
Speed is important	onehot	Usually the fastest style and suited to most FPGA styles.
There are <5 states	sequential	Default encoding.
A large output decoder follows the FSM	sequential   gray	Could be faster than onehot, even though the value must be decoded to determine the state. For sequential, more than one bit can change at a time; for gray, only one bit changes at a time, but more than one bit can be hot.
There are a large number of flip-flops	onehot	Fastest style, because each state variable has one bit set, and only one bit of the state register changes at a time.

3. If you are using VHDL, you have two choices for defining encoding:

- Use syn\_encoding as described above, and enable the FSM compiler.
- Use syn\_enum\_encoding to define the states (sequential, onehot, gray, and safe) and disable the FSM compiler. If you do not disable the FSM compiler, the syn\_enum\_encoding values are not implemented. This is because the FSM compiler, a mapper operation, overrides syn\_enum\_encoding, which is a compiler directive.

Use the syn\_enum\_encoding method for user-defined FSM encoding. For example:

```
attribute syn_enum_encoding of state_type : type is "001 010 101";
```

# Initializing RAMs

You can specify startup values for RAMs and pass them on to the place-and-route tools. See the following topics for ways to set the initial values:

- [Initializing RAMs in Verilog](#), on page 374
- [Initializing RAMs in VHDL](#), on page 375
- [Initializing RAMs with \\$readmemb and \\$readmemh](#), on page 378

## Initializing RAMs in Verilog

In Verilog, you specify startup values using initial statements, which are procedural assign statements guaranteed by the language to be executed by the simulator at the start of simulation. This means that any assignment to a variable within the body of the initial statement is treated as if the variable was initialized with the corresponding LHS value. You can initialize memories using the built-in load memory system tasks \$readmemb (binary) and \$readmemh (hex).

The following procedure is the recommended method for specifying initial values:

1. Create a data file with an initial value for every address in the memory array. This file can be a binary file or a hex file. See [Initialization Data File](#), on page 195 in the *Reference Manual* for details of the formats for these files.
2. Do the following in the Verilog file to define the module:
  - Include the appropriate task enable statement, \$readmemb or \$readmemh, in the initial statement for the module:

```
$readmemb ("fileName", memoryName [, startAddress [, stopAddress]]);
```

```
$readmemh ("fileName", memoryName [, startAddress [, stopAddress]]);
```

Use \$readmemb for a binary file and use \$readmemh for a hex file. For descriptions of the syntax, see [Initial Values in Verilog](#), on page 79 in the *Reference Manual*.

- Make sure the array declaration matches the order in the initial value data file you specified. As the file is read, each number encountered is assigned to a successive word element of the memory. The software starts with the left-hand address in the memory declaration, and loads consecutive words until the memory is full or the data file has been completely read. The loading order is the order in the declaration. For example, with the following memory definition, the first line in the data file corresponds to address 0:

```
reg [7:0] mem_up [0:63]
```

With this next definition, the first line in the data file applies to address 63:

```
reg [7:0] mem_down [63:0]
```

3. To forward-annotate initial values, use the \$readmemb or \$readmemh statements, as described in [Initializing RAMs with \\$readmemb and \\$readmemh, on page 378](#).

See [Example 1: RAM Initialization](#), on page 192 in the *Reference Manual* for an example of a Verilog single-port RAM.

## Initializing RAMs in VHDL

There are two ways to initialize the RAM in the VHDL code: with signal declarations or with variable declarations.

### Initializing VHDL Rams with Signal Declarations

The following example shows a single-port RAM that is initialized with signal initialization statements. For alternative methods, see [Initializing VHDL Rams with Variable Declarations, on page 377](#).

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

```
entity w_r2048x28 is
port (
    clk : in std_logic;
    adr : in std_logic_vector(10 downto 0);
    di : in std_logic_vector(26 downto 0);
    we : in std_logic;
    dout : out std_logic_vector(26 downto 0));
end;

architecture arch of w_r2048x28 is

-- Signal Declaration --

type MEM is array(0 to 2047) of std_logic_vector (26 downto 0);
signal memory : MEM := (
"1111111111111100000000000000",
,"11111001101101010011110001",
,"111001111000111100101100111",
,"110010110011101110011110001",
,"10100111100011111100110111",
,"100000000000000111111111111",
,"010110000111001111100110111",
,"001101001100011110011110001",
,"000110000111001100101100111",
,"000001100100011010011110001",
,"0000000000000001000000000000",
,"000001100100010101100001110",
,"000110000111000011010011000",
,"001101001100010001100001110",
,"01011000011100000011001000",
,"01111111111110000000000000",
,"101001111000110000011001000",
,"110010110011100001100001110",
,"111001111000110011010011000",
,"111110011011100101100001110",
,"111111111111101111111111111",
,"111110011011101010011110001",
,"111001111000111100101100111",
,"110010110011101110011110001",
,"10100111100011111100110111",
,"100000000000000111111111111",
,others => (others => '0'));

begin
process(clk)
```

```

begin
    if rising_edge(clk) then
        if (we = '1') then
            memory(conv_integer(adr)) <= di;
        end if;
        dout <= memory(conv_integer(adr));
    end if;
end process;

end arch;

```

## Initializing VHDL Rams with Variable Declarations

The following example shows a RAM that is initialized with variable declarations. For alternative methods, see [Initializing VHDL Rams with Signal Declarations, on page 375](#) and [Initializing RAMs with \\$readmemb and \\$readmemh, on page 378](#).

```

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

entity one is
generic (data_width      : integer := 6;
          address_width :integer  := 3
        );
port ( data_a      :in std_logic_vector(data_width-1 downto 0);
       raddr1      :in unsigned(address_width-2 downto 0);
       waddr1      :in unsigned(address_width-1 downto 0);
       we1         :in std_logic;
       clk         :in std_logic;
       out1        :out std_logic_vector(data_width-1 downto 0) );
end;

architecture rtl of one is
type mem_array is array(0 to 2**address_width -1) of
std_logic_vector(data_width-1 downto 0);
begin

WRITEL_RAM : process (clk)
variable mem : mem_array := (1 => "111101", others => (1=>'1',
others => '0'));
begin
    if rising_edge(clk) then
        out1 <= mem(to_integer(raddr1));
        if (we1 = '1') then

```

```
        mem(to_integer(waddr1)) := data_a;
    end if;
end if;
end process WRITE1_RAM;
end rtl;
```

## Initializing RAMs with \$readmemb and \$readmemh

1. Create a data file with an initial value for every address in the memory array. This file can be a binary file or a hex file. See [Initialization Data File](#), on page 195 in the *Reference Manual* for details.
2. Include one of the task enable statements, \$readmemb or \$readmemh, in the initial statement for the module:

```
$readmemb ("fileName", memoryName [, startAddress [, stopAddress]]) ;
$readmemh ("fileName", memoryName [, startAddress [, stopAddress]]) ;
```

Use \$readmemb for a binary file and \$readmemh for a hex file. For details about the syntax, see [Initial Values in Verilog, on page 79](#) in the *Reference Manual*.

## CHAPTER 10

# Specifying Design-Level Optimizations

---

This chapter covers techniques for optimizing your design using built-in tools or attributes. For vendor-specific optimizations, see [Chapter 13, Optimizing for GoWin Designs](#). It describes the following:

- [Tips for Optimization](#), on page 380
- [Pipelining](#), on page 384
- [Retiming](#), on page 388
- [Preserving Objects from Being Optimized Away](#), on page 396
- [Optimizing Fanout](#), on page 402
- [Sharing Resources](#), on page 406
- [Inserting I/Os](#), on page 406
- [Optimizing State Machines](#), on page 407
- [Inserting Probes](#), on page 413

# Tips for Optimization

The software automatically makes efficient trade-offs to achieve the best results. However, you can optimize your results by using the appropriate control parameters. This section describes general design guidelines for optimization. The topics have been categorized as follows:

- [General Optimization Tips](#), on page 380
- [Optimizing for Area](#), on page 381
- [Optimizing for Timing](#), on page 382

## General Optimization Tips

This section contains general optimization tips that are not directly area or timing-related. For area optimization tips, see [Optimizing for Area, on page 381](#). For timing optimization, see [Optimizing for Timing, on page 382](#).

- In your source code, remove any unnecessary priority structures in timing-critical designs. For example, use CASE statements instead of nested IF-THEN-ELSE statements for priority-independent logic.
- If your design includes safe state machines, use the syn\_encoding attribute with a value of safe. This ensures that the synthesized state machines never lock in an illegal state.
- For FSMs coded in VHDL using enumerated types, use the same encoding style (syn\_enum\_encoding attribute value) on both the state machine enumerated type and the state signal. This ensures that there are no discrepancies in the type of encoding to negatively affect the final circuit.
- Make sure that the source code supports inferencing or instantiation by using architecture-specific resources like memory blocks.
- Some designs benefit from hierarchical optimization techniques. To enable hierarchical optimization on your design, set the syn\_hier attribute to firm.
- For accurate results with timing-driven synthesis, explicitly define clock frequencies with a constraint, instead of using a global clock frequency.

## Optimizing for Area

This section contains information on optimizing to reduce area. Optimizing for area often means larger delays, and you will have to weigh your performance needs against your area needs to determine what works best for your design. For tips on optimizing for performance, see [Optimizing for Timing, on page 382](#). General optimization tips are in [General Optimization Tips, on page 380](#).

- Increase the fanout limit when you set the implementation options. A higher limit means less replicated logic and fewer buffers inserted during synthesis, and a consequently smaller area. In addition, as P&R tools typically buffer high fanout nets, there is no need for excessive buffering during synthesis. See [Setting Fanout Limits, on page 402](#) for more information.
- Enable the Resource Sharing option when you set implementation options. With this option checked, the software shares hardware resources like adders, multipliers, and counters wherever possible, and minimizes area. This is a global setting, but you can also specify resource sharing on an individual basis for lower-level modules. See [Sharing Resources, on page 406](#) for details.
- For designs with large FSMs, use the gray or sequential encoding styles, because they typically use the least area. For details, see [Specifying FSMs with Attributes and Directives, on page 372](#).
- If you are mapping into a CPLD and do not meet area requirements, set the default encoding style for FSMs to sequential instead of onehot. For details, see [Specifying FSMs with Attributes and Directives, on page 372](#).
- For small CPLD designs (less than 20K gates), you might improve area by using the `syn_hier` attribute with a value of flatten. When specified, the software optimizes across hierarchical boundaries and creates smaller designs.

## Optimizing for Timing

This section contains information on optimizing to meet timing requirements. Optimizing for timing is often at the expense of area, and you will have to balance the two to determine what works best for your design. For tips on optimizing for area, see [Optimizing for Area, on page 381](#). General optimization tips are in [General Optimization Tips, on page 380](#).

- Use realistic design constraints, about 10 to 15 percent of the real goal. Over-constraining your design can be counter-productive because you can get poor implementations. Typically, you set timing constraints like clock frequency, clock-to-clock delay paths, I/O delays, register I/O delays and other miscellaneous path delays. Use clock, false path, and multi-cycle path constraints to make the constraints realistic.
- Enable the Retiming option. This optimization moves registers into I/O buffers if this is permitted by the technology and the design. However, it may add extra registers when clouds of logic are balanced across more than one register-to-register timing path. Extra registers are only added in parallel within the timing path and only if no extra latency is added by the additional registers. For example, if registers are moved across a 2x1 multiplexer, the tool adds two new registers to accommodate the select and data paths.

You can set this option globally or on specific registers. When it is enabled, it automatically enables pipelining as well. See [Retiming, on page 388](#) for details.

- Enable the Pipelining option. With this optimization enabled, the tool moves existing registers into a ROM or multiplier. Unlike retiming, it does not add any new logic. Pipelining reduces routing and delay and the extra instance delay of the external register by moving it into the ROM or multiplier and making it a built-in register.
- Select a balanced fanout constraint. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated logic. See [Setting Fanout Limits, on page 402](#) for information about setting limits and using the `syn_maxfan` attribute. You can use this in conjunction with the `syn_replicate` attribute that controls register duplication and buffering.

- Control register duplication and buffering criteria with the `syn_replicate` attribute. The tool automatically replicates registers during optimization, and you can use this attribute globally or locally on a specific register to turn off register duplication. See [Controlling Buffering and Replication, on page 404](#) for a description. Use `syn_replicate` in conjunction with the `syn_maxfan` attribute that controls fanout.
- If the critical path goes through arithmetic components, try disabling Resource Sharing. You can get faster times at the expense of increased area, but use this technique carefully. Adding too many resources can cause longer delays and defeat your purpose.
- If the P&R and synthesis tools report different critical paths, use a timing constraint with the `-route` option. With this option, the software adds route delay to its calculations when trying to meet the clock frequency goal. Use realistic values for the constraints.
- For FSMs, use the onehot encoding style, because it is often the fastest implementation. If a large output decoder follows an FSM, gray or sequential encoding could be faster.
- For designs with black boxes, characterize the timing models accurately, using the `syn_tpd`, `syn_tco`, and `syn_tso` directives.
- If you see warnings about feedback muxes being created for signals when you compile your source code, make sure to assign set/resets for the signals. This improves performance by eliminating the extra mux delay on the input of the register.
- Make sure that you pass your timing constraints to the place-and-route tools, so that they can use the constraints to optimize timing.

# Pipelining

Pipelining is the process of splitting logic into stages so that the first stage can begin processing new inputs while the last stage is finishing the previous inputs. This ensures better throughput and faster circuit performance. If you are using selected technologies which use pipelining, you can also use the related technique of retiming to improve performance. See [Retiming, on page 388](#) for details.

For pipelining, the software splits the logic by moving registers into the multiplier or ROM:

This section discusses the following pipelining topics:

- [Prerequisites for Pipelining, on page 384](#)
- [Pipelining the Design, on page 385](#)

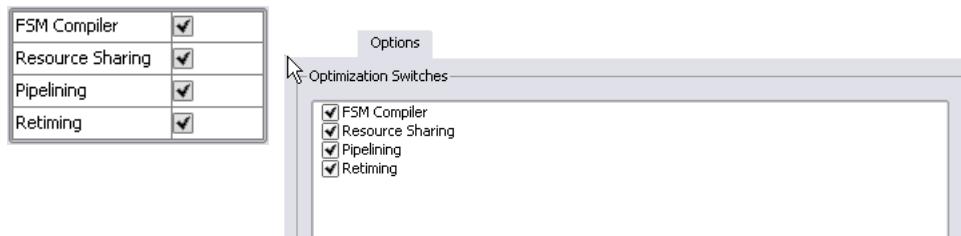
## Prerequisites for Pipelining

ROMs to be pipelined must be at least 512 words. Anything below this limit is too small.

## Pipelining the Design

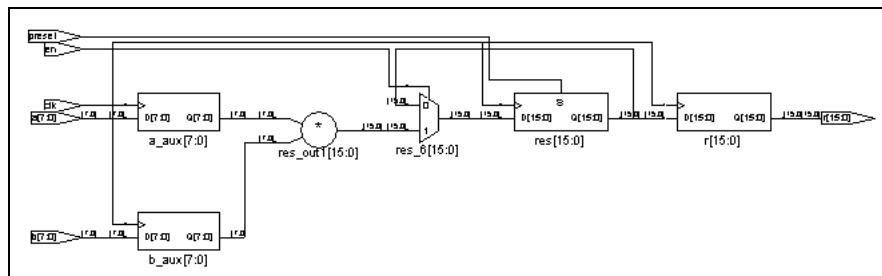
The following procedure shows you techniques for pipelining.

1. Make sure the design meets the criteria described in [Prerequisites for Pipelining, on page 384](#).
2. To enable pipelining for the whole design, check the Pipelining check box from the button panel in the Project window, or with the Project->Implementation Options command (Options tab). The option is only available in the appropriate technologies.

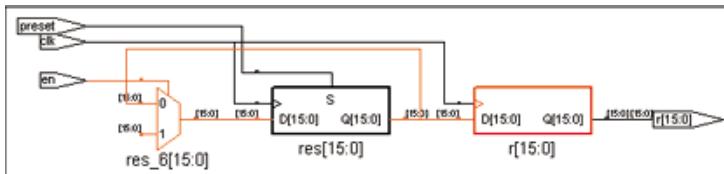


Use this approach as a first pass to get a feel for which modules you can pipeline. If you know exactly which registers you want to pipeline, add the attribute to the registers in the source code or interactively using the SCOPE interface.

3. To check whether individual registers are suitable for pipelining, do the following:
  - Open the RTL view of the design.
  - Select the register and press F12 to filter the schematic view.



- In the new schematic view, select the output and type e (or select Expand from the popup menu. Check that the register is suitable for pipelining.



- To enable pipelining on selected registers, use either of the following techniques:

- Check the Pipelining checkbox and attach the syn\_pipeline attribute with a value of 0 or false to any registers you do not want the software to move. This attribute specifies that the register cannot be moved for pipelining.
- Do not check the Pipelining checkbox. Attach the syn\_pipeline attribute with a value of 1 or true to any registers you want the software to consider for retiming. This attribute marks the register as one that can be moved during retiming, but does not necessarily force it to be moved during retiming.

The following are examples of the attribute:

SCOPE Interface:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Descr
1	<input checked="" type="checkbox"/>	register	res[15:0]	syn_pipeline	1	boolean	Controls pipelining c

Verilog Example:

```
reg [`lefta:0] a_aux;
reg [`leftb:0] b_aux;
reg [`lefta+`leftb+1:0] res /* synthesis syn_pipeline=1 */;
reg [`lefta+`leftb+1:0] res1;
```

VHDL Example:

```
architecture beh of onereg is
  signal temp1, temp2, temp3,
    std_logic_vector(31 downto 0);
  attribute syn_pipeline : boolean;
  attribute syn_pipeline of temp1 : signal is true;
  attribute syn_pipeline of temp2 : signal is true;
  attribute syn_pipeline of temp3 : signal is true;
```

5. Click Run.

The software looks for registers where all the flip-flops of the same row have the same clock, no control signal, or the same unique control signal, and pushes them inside the module. It attaches the `syn_pipeline` attribute to all these registers. If there already is a `syn_pipeline` attribute on a register, the software implements it.

6. Check the log file (\*.srr). You can use the Find command for occurrences of the word pipelining to find out which modules got pipelined.

The log file entries look like this:

```
@N: | Pipelining module res_out1
@N: | res_i is level 1 of the pipelined module res_out1
@N: | r is level 2 of the pipelined module res_out1
```

# Retiming

Retiming improves the timing performance of sequential circuits without modifying the source code. It automatically moves registers (register balancing) across combinational gates or LUTs to improve timing while maintaining the original behavior as seen from the primary inputs and outputs of the design. Retiming moves registers across gates or LUTs, but does not change the number of registers in a cycle or path from a primary input to a primary output. However, it can change the total number of registers in a design.

The retiming algorithm retimes only edge-triggered registers. It does not retime level-sensitive latches. Note that registers associated with RAMS, DSPs, and the mapping for generated clocks may be moved, regardless of the Retiming option setting. The Retiming option is not available if it does not apply to the family you are using.

These sections contain details about using retiming.

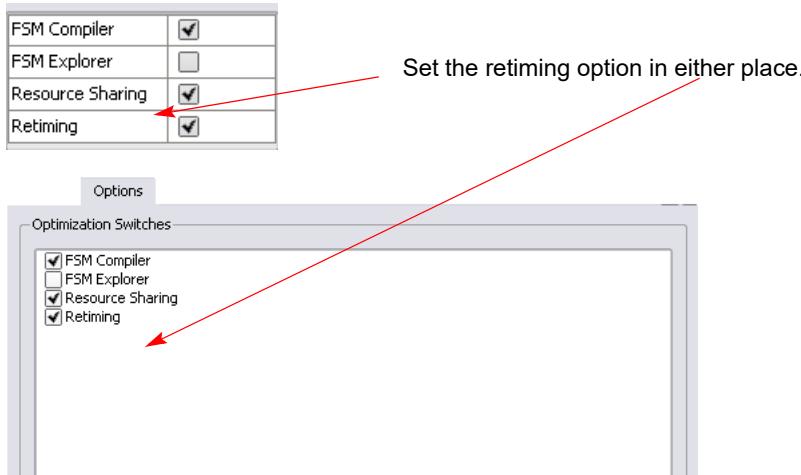
- [Controlling Retiming](#), on page 388
- [Retiming Example](#), on page 391
- [Retiming Report](#), on page 392
- [How Retiming Works](#), on page 392

## Controlling Retiming

The following procedure shows you how to use retiming.

1. To enable retiming for the whole design, check the Retiming check box.

You can set the Retiming option from the button panel in the Project window, or with the Project->Implementation Options command (Options tab). The option is only available in certain technologies.



Retiming works globally on the design, and moves edge-triggered registers as needed to balance timing.

2. To enable retiming on selected registers, use either of the following techniques:
  - Check the Retiming checkbox and attach the `syn_allow_retimimg` attribute with a value of 0 or false to any registers you do not want the software to move. This attribute specifies that the register cannot be moved for retiming. Refer to [How Retiming Works, on page 392](#) for a list of the components the retiming algorithm will move.
  - Do not check the Retiming checkbox. Attach the `syn_allow_retimimg` attribute with a value of 1 or true to any registers you want the software to consider for retiming. You can do this in the SCOPE interface or in the source code. This attribute marks the register as one that can be moved during retiming, but does not necessarily force it to be moved during retiming. If you apply the attribute to an FSM, RAM or SRL that is decomposed into flip-flops and logic, the software applies the attribute to all the resulting flip-flops

Retiming is a superset of pipelining; therefore adding `syn_allow_retimimg=1` on any registers implies that `syn_pipeline =1`.

3. You can also fine-tune retiming using attributes:

- To preserve the power-on state of flip-flops without sets or resets (FD or FDE) during retiming, set `syn_preserve=1` or `syn_allow_retimming=0` on these flip-flops.
  - To force flip-flops to be packed in I/O pads, set `syn_useioff=1` as a global attribute. This will prevent the flip-flops from being moved during retiming.
4. Set other options for the run. Retiming might affect some constraints and attributes. See [How Retiming Works, on page 392](#) for details.
  5. Click Run to start synthesis.

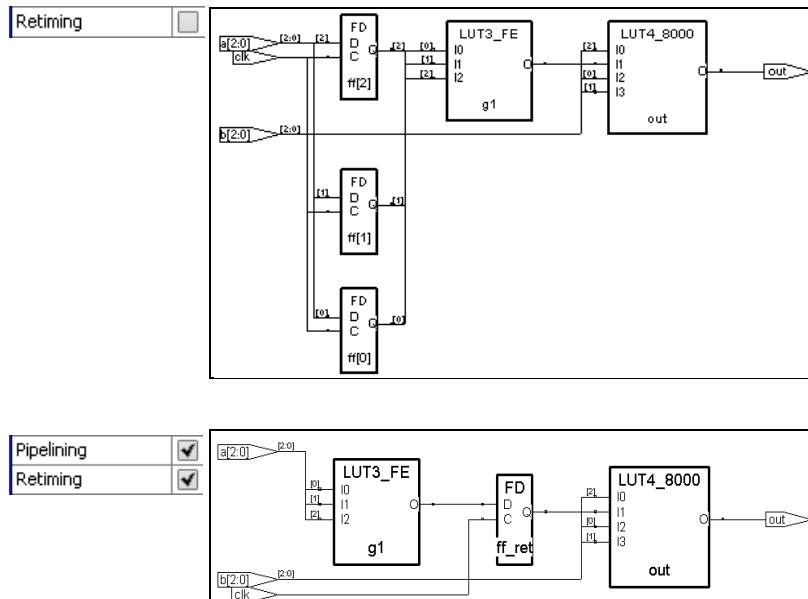
After the LUTs are mapped, the software moves registers to optimize timing. See [Retiming Example, on page 391](#) for an example. The software honors other attributes you set, like `syn_preserve`, `syn_useioff`, and `syn_ramstyle`. See [How Retiming Works, on page 392](#) for details.

Note that the tool might retime registers associated with RAMs, DSPs, and generated clocks, regardless of whether the Retiming option is on or off.

The log file includes a retiming report that you can analyze to understand the retiming changes. It contains a list of all the registers added or removed because of retiming. Retimed registers have a \_ret suffix added to their names. See [Retiming Report, on page 392](#) for more information about the report.

## Retiming Example

The following example shows a design with retiming disabled and enabled.



## Retiming Report

The retiming report is part of the log file, and includes the following:

- The number of registers added, removed, or untouched by retiming.
- Names of the original registers that were moved by retiming and which no longer exist in the Technology view.
- Names of the registers created as a result of retiming, and which did not exist in the RTL view. The added registers have a \_ret suffix.

## How Retiming Works

This section describes how retiming works when it moves sequential components (flip-flops). Registers associated with RAMs, DSPs, and the mapping for fixing generated clocks might be moved, whether Retiming is enabled or not. Here are some implications and results of retiming:

- Flip-flops with no control signals (resets, presets, and clock enables) are moved. Flip-flops with minimal control logic can also be retimed. Multiple flip-flops with reset, set or enable signals that need to be retimed together are only retimed if they have exactly the same control logic.
- The software does not retime the following combinational sequential elements: flip-flops with both set and reset, flip-flops with attributes like syn\_preserve, flip-flops packed in I/O pads, level-sensitive latches, registers that are instantiated in the code, SRLs, and RAMs. If a RAM with combinational logic has syn\_ramstyle set to registers, the registers can be retimed into the combinational logic.
- Retimed flip-flops are only moved through combinational logic. The software does not move flip-flops across the following objects: black boxes, sequential components, tristates, I/O pads, instantiated components, carry and cascade chains, and keepbufs.
- You might not be able to crossprobe retimed registers between the RTL and the Technology view, because there may not be a one-to-one correspondence between the registers in these two views after retiming. A single register in the RTL view might now correspond to multiple registers in the Technology view.
- Retiming affects or is affected by, these attributes and constraints:

Attribute/Constraint	Effect
False path constraint	Does not retime flip-flops with different false path constraints. Retimed registers affect timing constraints.
Multicycle constraint	Does not retime flip-flops with different multicycle constraints. Retimed registers affect timing constraints.
Register constraint	Does not maintain <code>set_reg_input_delay</code> and <code>set_reg_output_delay</code> constraints. Retimed registers affect timing constraints.
from/to timing exceptions	If you set a timing constraint using a from/to specification on a register, it is not retimed. The exception is when using a <code>max_delay</code> constraint. In this case, retiming is performed but the constraint is not forward annotated. (The <code>max_delay</code> value would no longer be valid.)
<code>syn_hier=macro</code>	Does not retime registers in a macro with this attribute.
<code>syn_keep</code>	Does not retime across <code>keepbufs</code> generated because of this attribute.
<code>syn_hier=macro</code>	Does not retime registers in a macro with this attribute.
<code>syn_pipeline</code>	Automatically enabled if retiming is enabled.
<code>syn_preserve</code>	Does not retime flip-flops with this attribute set.
<code>syn_probe</code>	Does not retime net drivers with this attribute. If the net driver is a LUT or gate, no flip-flops are retimed across it.
<code>syn_reference_clock</code>	On a critical path, does not retime registers with different <code>syn_reference_clock</code> values together, because the path effectively has two different clock domains.
<code>syn_useioff</code>	Does not override attribute-specified packing of registers in I/O pads. If the attribute value is false, the registers can be retimed. If the attribute is not specified, the timing engine determines whether the register is packed into the I/O block.
<code>syn_allow_retimng</code>	Registers are not retimed if the value is 0.

- Retiming does not change the simulation behavior (as observed from primary inputs and outputs) of your design. However if you are monitoring (probing) values on individual registers inside the design, you might need to modify your test bench if the probe registers are retimed.

- If retiming is enabled, registers connected to unconstrained I/O pins are not retimed by default. If you want to revert back to how retiming I/O paths was previously implemented, you can:
  - Globally turn on the Use clock period for unconstrained IO switch from the Constraints tab of the Implementation Options panel.
  - Add constraints to all input/output ports.
  - Separately constrain each I/O pin as required.

# Preserving Objects from Being Optimized Away

Synthesis can collapse or remove nets during optimization. If you want to retain a net for simulation, probing, or for a different synthesis implementation, you must specify this with an attribute. Similarly, the software removes duplicate registers or instances with unused output. If you want to preserve this logic for simulation or analysis, you must use an attribute. The following table lists the attributes to use in each situation. For details about the attributes and their syntax, see the *Attributes Reference Manual*.

To Preserve...	Use...	Result
Nets	<code>syn_keep</code> on wire or reg (Verilog), or signal (VHDL).	Keeps net for simulation, a different synthesis implementation, or for passing to the place-and-route tool.
Nets for probing	<code>syn_probe</code> on wire or reg (Verilog), or signal (VHDL)	Preserves internal net for probing.
Shared registers	<code>syn_keep</code> on input wire or signal of shared registers	Preserves duplicate driver cells, prevents sharing. See <a href="#">Using syn_keep for Preservation or Replication, on page 397</a> for details on the effects of applying <code>syn_keep</code> to different objects.
Sequential components	<code>syn_preserve</code> on reg or module (Verilog), signal or architecture (VHDL)	Preserves logic of constant-driven registers, keeps registers for simulation, prevents sharing
FSMs	<code>syn_preserve</code> on reg or module (Verilog), signal (VHDL)	Prevents the output port or internal signal that holds the value of the state register from being optimized
Instantiated components	<code>syn_noprune</code> on module or component (Verilog), architecture or instance (VHDL)	Keeps instance for analysis, preserves instances with unused outputs

See the following for more information:

- [Using syn\\_keep for Preservation or Replication](#), on page 397
- [Controlling Hierarchy Flattening](#), on page 400
- [Preserving Hierarchy](#), on page 401

## Using syn\_keep for Preservation or Replication

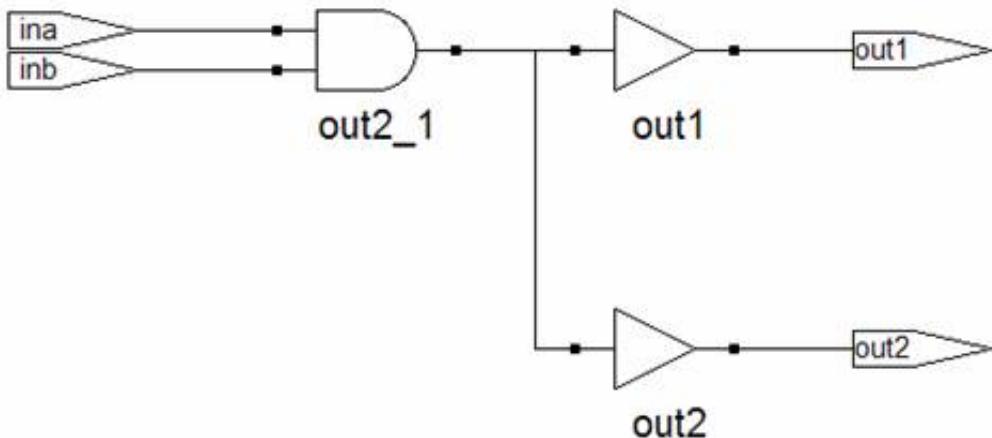
By default the tool considers replicated logic redundant, and optimizes it away. If you want to maintain the redundant logic, use syn\_keep to preserve the logic that would otherwise be optimized away.

The following Verilog code specifies a replicated AND gate:

```
module redundant1(ina,inb,out1);
  input ina,inb;
  output out1,out2;
  wire out1;
  wire out2;

  assign out1 = ina & inb;
  assign out2 = ina & inb;;
endmodule
```

The compiler implements the AND function by replicating the outputs out1 and out2, but optimizes away the second AND gate because it is redundant.



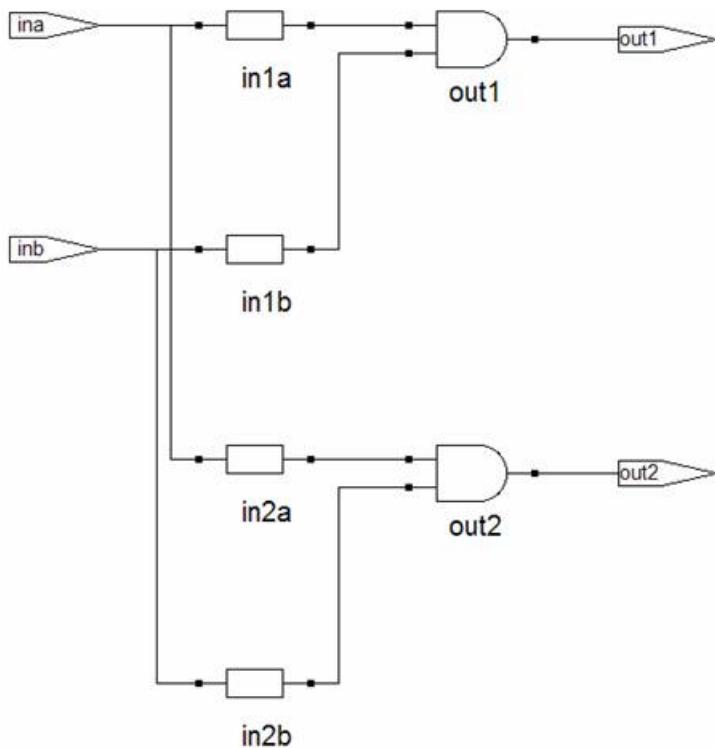
To replicate the AND gate in the previous example, apply `syn_keep` to the input wires, as shown below:

```
module redundant1d(ina,inb,out1,out2);
  input ina,inb;
  output out1,out2;
  wire out1;
  wire out2;

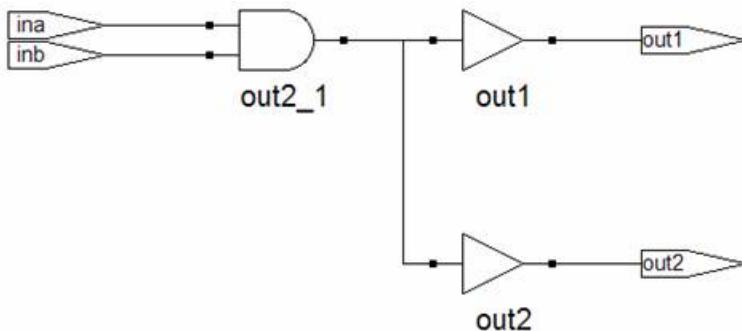
  wire in1a /*synthesis syn_keep = 1*/;
  wire in1b /*synthesis syn_keep = 1*/;
  wire in2a /*synthesis syn_keep = 1*/;
  wire in2b /*synthesis syn_keep = 1 */;

  assign in1a = ina ;
  assign in1b = inb ;
  assign in2a = ina;
  assign in2b = inb;
  assign out1 = in1a & in1b;
  assign out2 = in2a & in2b;
endmodule
```

Setting `syn_keep` on the input wires ensures that the second AND gate is preserved:



You must set `syn_keep` on the input wires of an instance if you want to preserve the logic, as in the replication of this AND gate. If you set it on the outputs, the instance is not replicated, because `syn_keep` preserves the nets but not the function driving the net. If you set `syn_keep` on the outputs in the example, you get only one AND gate, as shown in the next figure.



## Controlling Hierarchy Flattening

Optimization flattens hierarchy. To control the flattening, use the `syn_hier` attribute as described here. You can also use the attribute to prevent flattening, as described in [Preserving Hierarchy, on page 401](#).

1. Attach the `syn_hier` attribute with the value you want to the module or architecture you want to preserve.

To...	Value...
Flatten all levels below, but not the current level	<code>flatten</code>
Remove the current level of hierarchy without affecting the lower levels	<code>remove</code>
Remove the current level of hierarchy and the lower levels	<code>flatten, remove</code>
Flatten the current level (if needed for optimization)	<code>soft</code>

You can also add the attribute in SCOPE instead of the HDL code. If you use SCOPE to enter the attribute, make sure to use the `v:` syntax. For details, see [syn\\_hier, on page 57](#) in the *Attribute Reference Manual*.

The software flattens the design as directed. If there is a lower-level `syn_hier` attribute, it takes precedence over a higher-level one.

2. If you want to flatten the entire design, use the `syn_netlist_hierarchy` attribute set to `false`, instead of the `syn_hier` attribute.

This flattens the entire netlist and does not preserve any hierarchical boundaries. See [syn\\_netlist\\_hierarchy, on page 88](#) in the *Reference Manual* for the syntax.

## Preserving Hierarchy

The synthesis process includes cross-boundary optimizations that can flatten hierarchy. To override these optimizations, use the `syn_hier` attribute as described here. You can also use this attribute to direct the flattening process as described in [Controlling Hierarchy Flattening, on page 400](#).

1. Attach the `syn_hier` attribute to the module or architecture you want to preserve. You can also add the attribute in SCOPE. If you use SCOPE to enter the attribute, make sure to use the `v:` syntax.
2. Set the attribute value:

To...	Value...
Preserve the interface but allow cell packing across the boundary	firm
Preserve the interface with no exceptions	hard
Preserve the interface and contents with no exceptions	macro
Flatten lower levels but preserve the interface of the specified design unit	flatten, firm

The software flattens the design as directed. If there is a lower-level `syn_hier` attribute, it takes precedence over a higher-level one.

# Optimizing Fanout

You can optimize your results with attributes and directives, some of which are specific to the technology you are using. Similarly, you can specify objects or hierarchy that you want to preserve during synthesis. For a complete list of all the directives and attributes, see the *Attribute Reference Manual*. This section describes the following:

- [Setting Fanout Limits](#), on page 402
- [Controlling Buffering and Replication](#), on page 404

## Setting Fanout Limits

Optimization affects net fanout. If your design has critical nets with high fanout, you can set fanout limits. You can only do this for certain technologies. For details specific to individual technologies, see the *Reference Manual*.

1. To set a global fanout limit for the whole design, do either of the following:
  - Select Project-> Implementation Options->Device and type a value for the Fanout Guide option.
  - Apply the `syn_maxfan` attribute to the top-level view or module.

The value sets the number of fanouts for a given driver, and affects all the nets in the design. The defaults vary, depending on the technology. Select a balanced fanout value. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated or buffered logic. Both extremes affect routing and design performance. The right value depends on your design. The same value of 32 might result in fanouts of 11 or 12 and large delays on the critical path in one design or in excessive replication in another design.

The software uses the value as a soft limit, or a guide. It traverses the inverters and buffers to identify the fanout, and tries to ensure that all fanouts are under the limit by replicating or buffering where needed (see [Controlling Buffering and Replication](#), on page 404 for details). However, the synthesis tool does not respect the fanout limit absolutely; it ignores the limit if the limit imposes constraints that interfere with optimization.

2. To override the global fanout guideline and set a soft fanout limit at a lower level, set the `syn_maxfan` attribute on modules, views, or non-primitive instances.

These limits override the more global limits for that object. However, these limits still function as soft limits, and are replicated or buffered, as described in [Controlling Buffering and Replication, on page 404](#).

Attribute specified on	Effect
Module or view	Soft limit for the module; overrides the global setting.
Non-primitive instance	Soft limit; overrides global and module settings
Clock nets or asynchronous control nets	Soft limit.

3. To set a hard or absolute limit, set the `syn_maxfan` attribute on a port, net, register, or primitive instance.

Fanouts that exceed the hard limit are buffered or replicated, as described in [Controlling Buffering and Replication, on page 404](#).

4. To preserve net drivers from being optimized, attach the `syn_keep` or `syn_preserve` attributes.

For example, the software does not traverse a `syn_keep` buffer (inserted as a result of the attribute), and does not optimize it. However, the software can optimize implicit buffers created as a result of other operations; for example, it does not respect an implicit buffer created as a result of `syn_direct_enable`.

5. Check the results of buffering and replication in the following:

- The log file (click **View Log**). The log file reports the number of buffered and replicated objects and the number of segments created for the net.
- The HDL Analyst views. The software might not follow DRC rules when buffering or replicating objects, or when obeying hard fanout limits.

## Controlling Buffering and Replication

To honor fanout limits (see [Setting Fanout Limits, on page 402](#)) and reduce fanout, the software either replicates components or adds buffers. The tool uses buffering to reduce fanout on input ports, and uses replication to reduce fanout on nets driven by registers or combinational logic. The software first tries replication, replicating the net driver and splitting the net into segments. This increases the number of register bits in the design. When replication is not possible, the software buffers the signals. Buffering is more expensive in terms of intrinsic delay and resource consumption. The following table summarizes the behavior.

Replicates When...	Creates Buffers When...
<code>syn_maxfan</code> is set on a register output	<code>syn_maxfan</code> is set on input ports.
<code>syn_replicate</code> is 1	<code>syn_replicate</code> is 0. The <code>syn_replicate</code> attribute is only used to turn off the replication.
	<code>syn_maxfan</code> is set on a port/net that is driven by a port or I/O pad
	The net driver has a <code>syn_keep</code> or <code>syn_preserve</code> attribute
	The net driver is not a primitive gate or register

You can control whether high fanout nets are buffered or replicated, using the techniques described here:

- To use buffering instead of replication, set `syn_replicate` with a value of 0 globally, or on modules or registers. The `syn_replicate` attribute prevents replication, so that the software uses buffering to satisfy the fanout limit. For example, you can prevent replication between clock boundaries for a register that is clocked by `clk1` but whose fanin cone is driven by `clk2`, even though `clk2` is an unrelated clock in another clock group.

- Inverters merged with fanout loads increase fanout on the driver during placement and routing. A distinction is made between a keep buffer created as the result of the `syn_keep` attribute being applied by the user (explicit keep buffer) and a keep buffer that exists as the result of another attribute (implicit keep buffer). For example, the `syn_direct_enable` attribute inserts a keep buffer. When a `syn_maxfan` attribute is applied to the output of an explicit keep buffer, the signal is buffered (the keep buffer is not traversed so that the driver is not replicated). When the `syn_maxfan` attribute is applied to the output of an implicit keep buffer, the keep buffer is traversed and the driver is replicated.
- Turn off buffering and replication entirely, by setting `syn_maxfan` to a very high number, like 1000.

## Sharing Resources

One of the ways to optimize area is to use resource sharing in the compiler. With resource sharing, the software uses the same arithmetic operators for mutually exclusive statements; for example, with the branches of a case statement. Conversely, you can improve timing by disabling resource sharing, but at the expense of increased area.

Compiler resource sharing is on by default. You can set it globally and then override the global setting on individual modules

To disable resource sharing globally for the whole design:

- Leave the default setting to improve area; disable the option to improve timing.
- Select Project->Implementation Options->Options, disable Resource Sharing. Alternatively, disable the Resource Sharing button on the left side of the Project view.

## Inserting I/Os

You can control I/O insertion globally, or on a port-by-port basis.

To control the insertion of I/O pads at the top level of the design, use the Disable I/O Insertion option as follows:

- Select Project->Implementation Options and click the Device panel.
- Enable the option (checkbox on) if you want to do a preliminary run and check the area taken up by logic blocks, before synthesizing the entire design.

Do this if you want to check the area your blocks of logic take up, before you synthesize an entire FPGA. If you disable automatic I/O insertion, you do not get *any* I/O pads in your design, unless you manually instantiate them.

- Leave the Disable I/O Insertion checkbox empty (disabled) if you want to automatically insert I/O pads for all the inputs, outputs and bidirectionals.

When this option is set, the software inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist. Once inserted, you can override the I/O pad inserted by directly instantiating another I/O pad.

- For the most control, enable the option and then manually instantiate the I/O pads for specific pins, as needed.

Enable this attribute to preserve user-instantiated pads, insert pads on unconnected ports, insert bi-directional pads on bi-directional ports instead of converting them to input ports, or insert output pads on unconnected outputs.

If you do not set the `syn_force_pads` attribute, the synthesis design optimizes any unconnected I/O buffers away.

## Optimizing State Machines

You can optimize state machines with the symbolic FSM Compiler tool.

The Symbolic FSM Compiler is an advanced state machine optimizer, it automatically recognizes state machines in your design and optimizes them. Unlike other synthesis tools that treat state machines as regular logic, 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.

For more information, see the following:

- [Deciding when to Optimize State Machines](#), on page 408
- [Running the FSM Compiler](#), on page 408

## Deciding when to Optimize State Machines

The FSM Compiler is an automatic tool for encoding state machines, but you can also specify FSMs manually with attributes. For more information about using attributes, see [Specifying FSMs with Attributes and Directives, on page 372](#).

Here are the main reasons to use the FSM Compiler:

- To generate better results for your state machines

The software uses optimization techniques that are specifically tuned for FSMs, like reachability analysis for example. The FSM Compiler also lets you convert an encoded state machine to another encoding style (to improve speed and area utilization) without changing the source. For example, you can use a onehot style to improve results.

- To debug the state machines

State machine description errors result in unreachable states, so if you have errors, you will have fewer states. You can check whether your source code describes your state machines correctly. You can also use the FSM Viewer to see a high-level bubble diagram and crossprobe from there. For information about the FSM Viewer, see [Using the FSM Viewer \(Standard\), on page 329](#).

## Running the FSM Compiler

You can run the FSM Compiler tool on the whole design or on individual FSMs. See the following:

- [Running the FSM Compiler on the Whole Design, on page 408](#)
- [Running the FSM Compiler on Individual FSMs, on page 410](#)

### Running the FSM Compiler on the Whole Design

1. Enable the compiler by checking the Symbolic FSM Compiler box in one of these places:
  - The main panel on the left side of the project window
  - The Options tab of the dialog box that comes up when you click the Add Implementation/New Impl or Implementation Options buttons

2. To set a specific encoding style for a state machine, define the style with the syn\_encoding attribute, as described in [Specifying FSMs with Attributes and Directives, on page 372](#).

If you do not specify a style, the FSM Compiler picks an encoding style based on the number of states.

3. Click Run to run synthesis.

The software automatically recognizes and extracts the state machines in your design, and instantiates a state machine primitive in the netlist for each FSM it extracts. It then optimizes all the state machines in the design, using techniques like reachability analysis, next state logic optimization, state machine re-encoding and proprietary optimization algorithms. Unless you specified an encoding style, the tool automatically selects the encoding style. If you did specify a style, the tool uses that style.

In the log file, the FSM Compiler writes a report that includes a description of each state machine extracted and the set of reachable states for each state machine.

4. Select View->View Log File and check the log file for descriptions of the state machines and the set of reachable states for each one. You see text like the following:

```
Extracted state machine for register cur_state
State machine has 7 reachable states with original encodings of:
 0000001
 0000010
 0000100
 0001000
 0010000
 0100000
 1000000
....
original code -> new code
 0000001 -> 0000001
 0000010 -> 0000010
 0000100 -> 0000100
 0001000 -> 0001000
 0010000 -> 0010000
 0100000 -> 0100000
 1000000 -> 1000000
```

5. Check the state machine implementation in the RTL and Technology views and in the FSM viewer.
  - In the RTL view you see the FSM primitive with one output for each state.
  - In the Technology view, you see a level of hierarchy that contains the FSM, with the registers and logic that implement the final encoding.
  - In the FSM viewer you see a bubble diagram and mapping information. For information about the FSM viewer, see [Using the FSM Viewer \(Standard\), on page 329](#).
  - In the statemachine.info text file, you see the state transition information.

## Running the FSM Compiler on Individual FSMs

If you have state machines that you do not want automatically optimized by the FSM Compiler, you can use one of these techniques, depending on the number of FSMs to be optimized. You might want to exclude state machines from automatic optimization because you want them implemented with a specific encoding or because you do not want them extracted as state machines. The following procedure shows you how to work with both cases.

1. If you have just a few state machines you do not want to optimize, do the following:
  - Enable the FSM Compiler by checking the box in the button panel of the Project window.
  - If you do not want to optimize the state machine, add the syn\_state\_machine directive to the registers in the Verilog or VHDL code. Set the value to 0. When synthesized, these registers are not extracted as state machines.

```
Verilog reg [3:0] curstate /* synthesis syn_state_machine=0 */ ;
```

```
VHDL signal curstate : state_type;
attribute syn_state_machine : boolean;
attribute syn_state_machine of curstate : signal is
false;v
```

---

- If you want to specify a particular encoding style for a state machine, use the `syn_encoding` attribute, as described in [Specifying FSMs with Attributes and Directives, on page 372](#). When synthesized, these registers have the specified encoding style.
- Run synthesis.

The software automatically recognizes and extracts all the state machines, except the ones you marked. It optimizes the FSMs it extracted from the design, honoring the `syn_encoding` attribute. It writes out a log file that contains a description of each state machine extracted, and the set of reachable states for each FSM.

2. If you have many state machines you do not want optimized, do this:

- Disable the compiler by disabling the Symbolic FSM Compiler box in one of these places: the main panel on the left side of the project window or the Options tab of the dialog box that comes up when you click the Add Implementation or Implementation Options buttons. This disables the compiler from optimizing any state machine in the design. You can now selectively turn on the FSM compiler for individual FSMs.
- For state machines you want the FSM Compiler to optimize automatically, add the `syn_state_machine` directive to the individual state registers in the VHDL or Verilog code. Set the value to 1. When synthesized, the FSM Compiler extracts these registers with the default encoding styles according to the number of states.

---

```
Verilog reg [3:0] curstate /* synthesis syn_state_machine=1 */ ;  
VHDL signal curstate : state_type;  
attribute syn_state_machine : boolean;  
attribute syn_state_machine of curstate : signal is true;
```

---

- For state machines with specific encoding styles, set the encoding style with the `syn_encoding` attribute, as described in [Specifying FSMs with Attributes and Directives, on page 372](#). When synthesized, these registers have the specified encoding style.

- Run synthesis.

The software automatically recognizes and extracts only the state machines you marked. It automatically assigns encoding styles to the state machines with the `syn_state_machine` attribute, and honors the encoding styles set with the `syn_encoding` attribute. It writes out a log file that contains a description of each state machine extracted, and the set of reachable states for each state machine.

3. Check the state machine implementation in the RTL and Technology views and in the FSM viewer. For information about the FSM viewer, see [Using the FSM Viewer \(Standard\), on page 329](#).

# Inserting Probes

Probes are extra wires that you insert into the design for debugging. When you insert a probe, the signal is represented as an output port at the top level. You can specify probes in the source code or by interactively attaching an attribute.

## Specifying Probes in the Source Code

To specify probes in the source code, you must add the `syn_probe` attribute to the net. You can also add probes interactively, using the procedure described in [Adding Probe Attributes Interactively, on page 415](#).

1. Open the source code file.
2. For Verilog source code, attach the `syn_probe` attribute as a comment on any internal signal declaration:

```
module alu(out, opcode, a, b, sel);
    output [7:0] out;
    input [2:0] opcode;
    input [7:0] a, b;
    input sel;
    reg [7:0] alu_tmp /* synthesis syn_probe=1 */;
    reg [7:0] out;
//Other code
```

The value 1 indicates that probe insertion is turned on. For detailed information about Verilog attributes and examples of the files, see the *Attribute Reference Manual*.

To define probes for part of a bus, specify where you want to attach the probes; for example, if you specify `reg [1:0]` in the previous code, the software only inserts two probes.

3. For VHDL source code, add the `syn_probe` attribute as follows:

```
architecture rtl of alu is
    signal alu_tmp : std_logic_vector(7 downto 0) ;
    attribute syn_probe : boolean;
    attribute syn_probe of alu_tmp : signal is true;
    --other code;
```

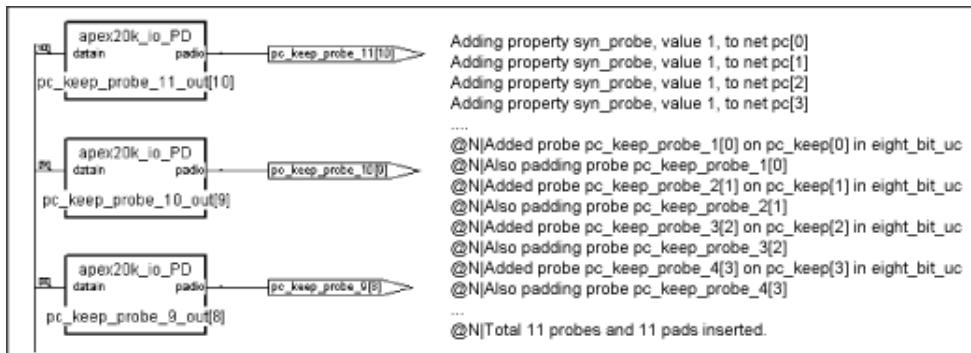
For detailed information about VHDL attributes and sample files, see the *Attribute Reference Manual*.

4. Run synthesis.

The software looks for nets with the `syn_probe` attribute and creates probes and I/O pads for them.

5. Check the probes in the log file (\*.srr) and the Technology view.

This figure shows some probes and probe entries in the log file.



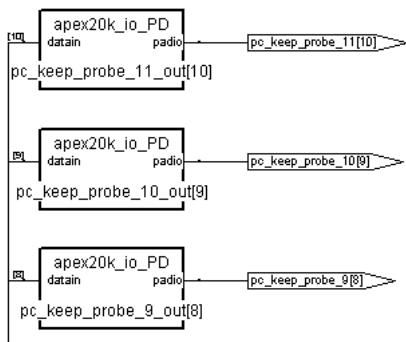
## Adding Probe Attributes Interactively

The following procedure shows you how to insert probes by adding the `syn_probe` attribute through the SCOPE interface. Alternatively, you can add the attribute in the source code, as described in [Specifying Probes in the Source Code, on page 413](#).

1. Open the SCOPE window and click Attributes.
2. Push down as necessary in an RTL view, and select the net for which you want to insert a probe point.

Do not insert probes for output or bidirectional signals. If you do, you see warning messages in the log file.
3. Do the following to add the attribute:
  - Drag the net into a SCOPE cell.
  - Add the prefix `n:` to the net name in the SCOPE window. If you are adding a probe to a lower-level module, the name is created by concatenating the names of the hierarchical instances.
  - If you want to attach probes to part but not all of a bus, make the change in the Object column. For example, if you enter `n:UC_ALU.longq[4:0]` instead of `n:UC_ALU.longq[8:0]`, the software only inserts probes where specified.
  - Select `syn_probe` in the Attribute column, and type `1` in the Value column.
  - Add the constraint file to the project list.
4. Rerun synthesis.
5. Open a Technology view and check the probe wires that have been inserted. You can use the Ports tab of the Find form to locate the probes.

The software adds I/O pads for the probes. The following figure shows some of the pads in the Technology view and the log file entries.



Adding property `syn_probe`, value 1, to net `pc[0]`  
Adding property `syn_probe`, value 1, to net `pc[1]`  
Adding property `syn_probe`, value 1, to net `pc[2]`  
Adding property `syn_probe`, value 1, to net `pc[3]`  
....  
@N|Added probe `pc_keep_probe_1[0]` on `pc_keep[0]` in  
eight\_bit\_uc  
@N|Also padding probe `pc_keep_probe_1[0]`  
@N|Added probe `pc_keep_probe_2[1]` on `pc_keep[1]` in  
eight\_bit\_uc  
@N|Also padding probe `pc_keep_probe_2[1]`  
@N|Added probe `pc_keep_probe_3[2]` on `pc_keep[2]` in  
eight\_bit\_uc

## CHAPTER 11

# Working with Compile Points

---

The following sections describe compile points and how to use them in logic synthesis iterative flows:

- [Compile Point Basics](#), on page 418
- [Compile Point Synthesis Basics](#), on page 426
- [Synthesizing Compile Points](#), on page 436
- [Resynthesizing Incrementally](#), on page 449

# Compile Point Basics

Compile points are RTL partitions of the design that you define before synthesizing the design. Compile points can be defined manually, or the tool can generate them automatically. The software treats each compile point as a block, and can synthesize, optimize, place, and route the compile points independently. Compile points can be nested.

See the following topics for some details about compile points:

- [Advantages of Compile Point Design](#), next
- [Nested Compile Points](#), on page 420
- [Compile Point Types](#), on page 422

## Advantages of Compile Point Design

Designing with compile points makes it more efficient to work with the increasingly larger designs of today and the corresponding team approach to design. They offer several advantages, which are described here:

- [Compile Points and Design Flows](#), next
- [Runtime Savings](#), on page 419
- [Design Preservation](#), on page 419

## Compile Points and Design Flows

Compile points improve the efficacy of both top-down and bottom-up design flows:

- In a traditional bottom-up design flow, compile points make it possible to easily divide up the design effort between designers or design teams. The compile points can be worked on separately and individually. The compile point synthesis flow eliminates the need to maintain the complex error-prone scripts for stitching, modeling, and ordering required by the traditional bottom-up design flow.

- From a top-down design flow perspective, compile points make it easier to work on the top-level design. You can mark compile points that are still being developed as black boxes, and synthesize the top level with what you have. You can also customize the compile point type settings for individual compile points to take advantage of cross-boundary optimizations.

You can also synthesize incrementally, because the tool does not resynthesize compile points that are unchanged when you resynthesize the design. This saves runtime and also preserves parts of the design that are done while the rest of the design is completed.

See [Compile Point Synthesis, on page 432](#) for a description of the synthesis process with compile points.

## Runtime Savings

Compile points are the required foundation for incremental synthesis, which translates directly to runtime savings.

Incremental synthesis uses compile points to determine which portions of the design to resynthesize, only resynthesizing the compile points that have been modified. See [Resynthesizing Compile Points Incrementally, on page 449](#).

## Design Preservation

Using compile points addresses the need to maintain the overall stability of a design while portions of the design evolve. When you use compile points to partition the design, you can isolate one part from another. This lets you preserve some compile points, and only resynthesize those that need to be rerun. These scenarios describe some design situations where compile points can be used to isolate parts of the design and run incremental synthesis:

- During the initial design phase, design modules are still being designed. Use compile points to preserve unchanged design modules and evaluate the effects of modifications to parts of the design that are still changing.
- During design integration, use compile points to preserve the main design modules and only allow the glue logic to be remapped.
- If your design contains IP, synthesize the IP, and use compile points to preserve them while you run incremental synthesis on the rest of the design.

- In the final stages of the design, use compile points to preserve design modules that do not need to be updated while you work through minor RTL changes in some other part of the design.

## Manual Compile Points

Manual compile points provide control. You can specify boundary constraints for each compile point individually. You can separate completed parts of the design from parts that are still being designed, or fine-tune the compile points to take advantage of as many cross-boundary optimizations as possible. For example, you can ensure that a critical path does not cross a compile point boundary, thus ensuring synthesis results with optimal performance.

### Guidelines for Using Manual Compile Points

The table lists some guidelines:

#### Use Manual Compile Points...

---

When you know the design in detail.

Create manual compile points to get better QoR. Good candidates for manual compile points include the following:

- Completed modules with registered interfaces, where you want to preserve the design
  - Modules created to include an entire critical path, so as to get the best performance.
  - Modules that are less likely to be affected by cross boundary optimizations like constant propagation and register absorption.
- 

When you do not want further optimizations to a completed compile point.

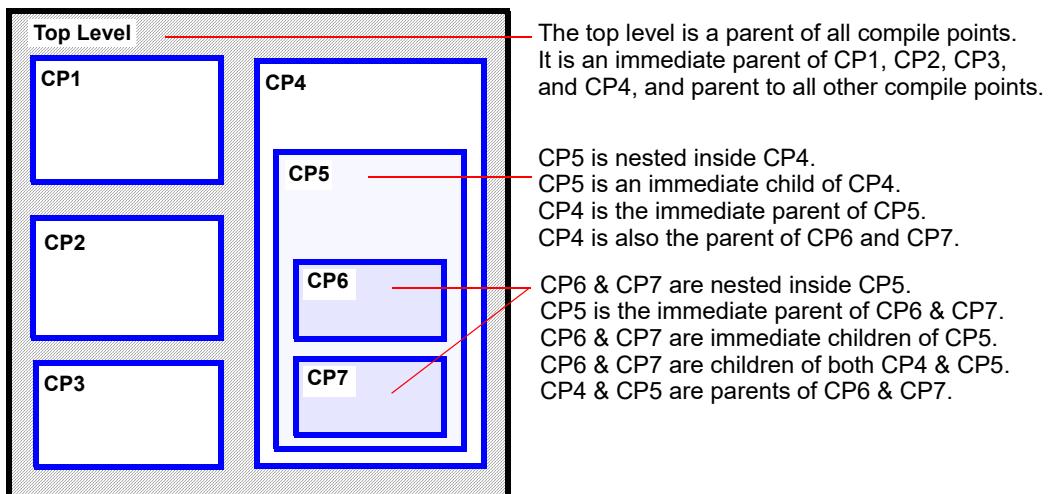
When you want more control to determine cross-boundary optimizations on an individual basis.

---

## Nested Compile Points

A design can have any number of compile points, and compile points can be nested inside other compile points. In the following figure, compile point CP6 is nested inside compile point CP5, which is nested inside compile point CP4.

To simplify things, the term *child* is used to refer to a compile point that is contained inside another compile point; the term *parent* is used to refer to a container compile point that contains a child. These terms are not used in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points. In the figure above, both CP5 and CP6 are children of CP4; both CP4 and CP5 are parents of CP6; CP5 is an immediate child of CP4 and an immediate parent of CP6.



## Compile Point Types

Compile point designs do not have as good QoR as designs without them because the boundaries limit optimizations. Cross-boundary optimizations typically improve area and timing, at the expense of runtime. The compile point type determines whether boundary optimizations are allowed. For manual compile points, you define the type. See [Defining the Compile Point Type, on page 442](#) for details.

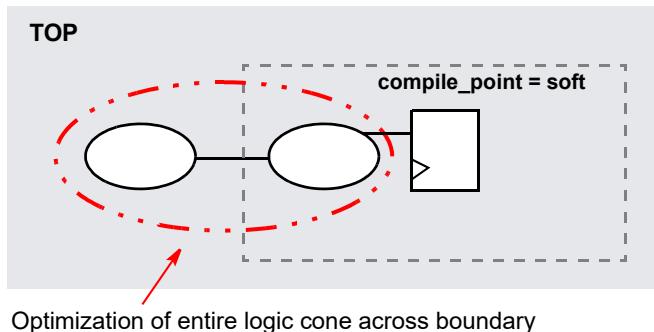
These are descriptions of the soft, hard, locked, and black\_box compile types:

- Soft

Compile point boundaries can be reoptimized during top-level mapping. Timing optimizations like sizing, buffering, and DRC logic optimizations can modify boundary instances of the compile point and combine them with functions from the next higher level of the design. The compile point interface can also be modified. Multiple instances are unqualified. Any optimization changes can propagate both ways: into the compile point and from the compile point to its parent.

Using soft mode usually yields the best quality of results, because the software can utilize boundary optimizations. On the other hand, soft compile points can take a longer time to run than the same design with hard or locked compile points. Unless they are at the leaf level, soft compile points are not processed in parallel. Upper levels that contain soft compile points cannot be processed until the lower level has been mapped, with the top level processed last.

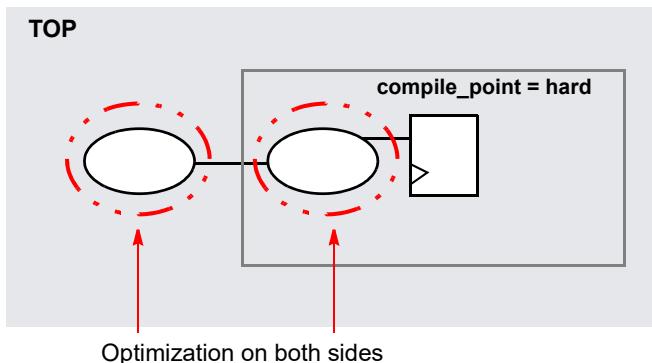
The following figure shows the soft compile point with a dotted boundary to show that logic can be moved in or out of the compile point.



- Hard

For hard compile points, the compile point boundary can be reoptimized during top-level mapping and instances on both sides of the boundary can be modified by timing and DRC optimizations using top-level constraints. However, the boundary is not modified. Any changes can propagate in either direction while the compile point boundary (port/interface) remains unchanged. Multiple instances are uniquified. For performance improvements, constant propagation and removal of unused logic optimizations are performed across hard compile points.

In the following figure, the solid boundary on the hard compile point indicates that no logic can be moved in or out of the compile point.



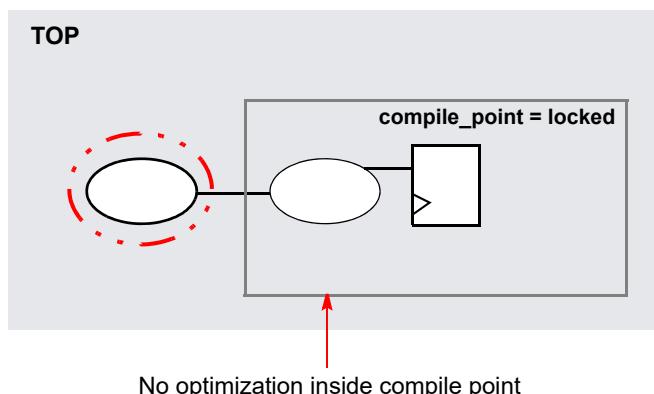
The hard compile point type allows for optimizations on both sides of the boundary without changing the boundary. There is a trade-off in quality of results to keep the boundaries. Using hard also allows for hierarchical equivalence checking for the compile point module.

- Locked

This is the default compile point type. With a locked compile point, the tool does not make any interface changes or reoptimize the compile point during top-level mapping. An interface logic model (ILM) of the compile point is created (see [Interface Logic Models, on page 428](#)) and included for the top-level mapping. The ILM remains unchanged during top-level mapping.

The locked value indicates that all instances of the same compile point are identical and unaffected by top-level constraints or critical paths. As a result, multiple instances of the compile point module remain identical even though the compile point is unqualified. The Technology view (srm file) shows unique names for the multiple instances, but in the final Verilog netlist (vma file) the original module names for the multiple instances are restored.

Timing optimization can only modify instances outside the compile point. Although the compile point is used to time the top-level netlist, changes do not propagate into or out of a locked compile point. The following figure shows a solid boundary for the locked compile point to indicate that no logic is moved in or out of the compile point during top-level mapping.



This mode has the largest trade-off in terms of QoR, because there are no boundary optimizations. So, it is very important to provide accurate constraints for locked compile points. The following table lists some advantages and limitations with the locked compile point:

Advantages	Limitations
Consumes smallest amount of memory. Used for large designs because of this memory advantage.	Interface timing
Provides most runtime advantage compared to other compile point types.	Constant propagation
Allows for hierarchical place and route with multiple output netlists for each compile point and the top-level output netlist.	GSR hookup
Allows for hierarchical simulation.	IO pads, like IBUFs and OBUFs, should not be instantiated within compile points

## Compile Point Type Summary

The following table summarizes how the tool handles different compile points during synthesis:

Features	Compile Point Type		
	Soft	Hard	Locked
Boundary optimizations	Yes	Limited	No
Uniquification of multiple instance modules	Yes	Yes	Limited
Compile point interface (port definitions)	Modified	Not modified	Not modified
Hierarchical simulation	No	no	Yes
Hierarchical equivalence checking	No	Yes	Yes
Interface Logic Model (created/used)	No	No	Yes

# Compile Point Synthesis Basics

This section describes the compile point constraint files and timing models, and describes the steps the tool goes through to synthesize compile points. See the following for details:

- [Compile Point Constraint Files](#), on page 426
- [Interface Logic Models](#), on page 428
- [Interface Timing for Compile Points](#), on page 429
- [Compile Point Synthesis](#), on page 432
- [Incremental Compile Point Synthesis](#), on page 434
- [Forward-annotation of Compile Point Timing Constraints](#), on page 435

For step-by-step information about how to use compile points, see [Synthesizing Compile Points](#), on page 436.

## Compile Point Constraint Files

A compile point design can contain two levels of constraint files, as described below:

- The constraint file at the top level

This is a required file, and contains constraints that apply to the entire design. This file also contains the definitions of the compile points in the design. The `define_compile_point` command is automatically written to the top-level constraint file for each compile point you define.

The following figure shows that this design has one locked compile point, `pgm_cntr`. It uses the following syntax to define the compile point:

```
define_compile_point {v:work.pgm_cntr} -type {locked}
```

```
13
14 ##### BEGIN Collections - (Populated from tab in SCOPE, do not edit)
15 ##### END Collections
16
17 ##### BEGIN Clocks - (Populated from tab in SCOPE, do not edit)
18 create_clock {p:clock} -period {10}
19
20 ##### END Clocks
21
22 ##### BEGIN "Generated Clocks" - (Populated from tab in SCOPE, do not edit)
23 ##### END "Generated Clocks"
24
25 ##### BEGIN Inputs/Outputs - (Populated from tab in SCOPE, do not edit)
26 ##### END Inputs/Outputs
27
28 ##### BEGIN "Delay Paths" - (Populated from tab in SCOPE, do not edit)
29 ##### END "Delay Paths"
30
31 ##### BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
32 ##### END Attributes
33
34 ##### BEGIN "I/O Standards" - (Populated from tab in SCOPE, do not edit)
35 ##### END "I/O Standards"
36
37 ##### BEGIN "Compile Points" - (Populated from tab in SCOPE, do not edit)
38 define_compile_point {v:work.prgm_cntr} -type {locked}
39 ##### END "Compile Points"
40
```

- Constraint files at the compile point level

These constraint files are optional, and are used for better control over manual compile points.

The compile point constraints are specific to the compile point and only apply within it. If your design has manual compile points, you can define corresponding compile point constraint files for them. See [Setting Constraints at the Compile Point Level, on page 444](#) for a step-by-step procedure.

When compile point constraints are defined, the tool uses them to synthesize the compile point, not automatic interface timing. Note that depending on the compile point type, the tool might further optimize the compile points during top-down synthesis of the top level to improve timing performance and overall design results, but the compile point itself is synthesized with the defined compile point constraints.

The first command in a compile point constraint file is `define_current_design`, and it specifies the compile point module for the contained constraints. This command sets the context for the constraint file. The remainder of the file is similar to the top-level constraint file. For example:

```
define_current_design {work.pgrm_cntr}
```

```
4 # by Synplify Pro, Synplify FPGA 9.0 Scope Editor
5 # Block-Level Constraint File
6 #
7 define_current_design {prgm_cntr}
8 #
9
10 #
11 # Collections
12 #
13 #
14 #
15 # Clocks
16 #
17 create_clock -enable    {clock}    -clockgroup default_clkgroup_0
18 #
19 #
20 # Clock to Clock
21 #
22 #
23 #
24 # Inputs/Outputs
25 #
26 define_input_delay          -default
27 define_output_delay -disable -default
28 define_input_delay -disable   {resetn}
```

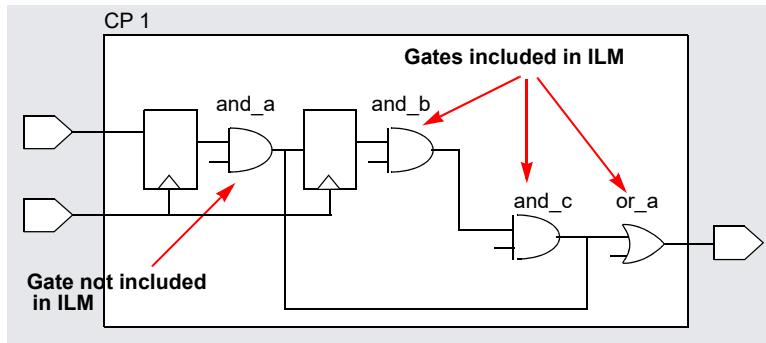
If your design has some compile points with their own constraint files and others without them, the tool uses the defined compile point constraints when it synthesizes those compile points. For the other compile points without defined constraints, it uses automatic interface timing, as described in [Interface Timing for Compile Points, on page 429](#).

## Interface Logic Models

The interface logic model (ILM) of a locked or hard compile point is a timing model that contains only the interface logic necessary for accurate timing. An ILM is a partial gate-level netlist that represents the original design accurately while requiring less memory during mapping. Using ILMs improves the runtime for static timing analysis without compromising timing accuracy.

The tool does not do any timing optimizations on an ILM. The interface logic is preserved with no modifications. All logic required to recreate timing at the top level is included in the ILM. ILM logic includes any paths from an input/inout port to an internal register, an internal register to an output/inout port, and an input/inout port to an output/inout port.

The tool removes internal register-to-register paths, as shown in this example. In this design, `and_a` is not included in the ILM because the timing path that goes through `and_a` is an internal register-to-register path.

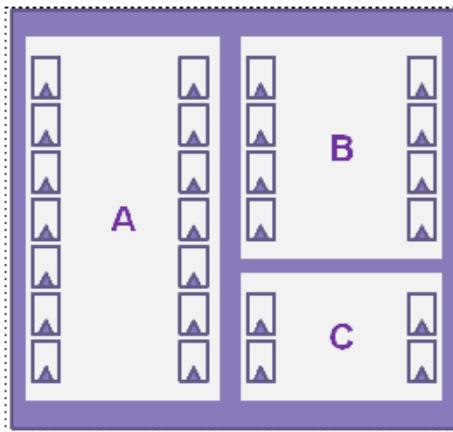


## Interface Timing for Compile Points

By default, the synthesis tool automatically infers timing constraints for all compile points from the top-level constraints. However, if a compile point has its own constraint file, the tool applies those compile point-specific constraints to synthesize the compile point.

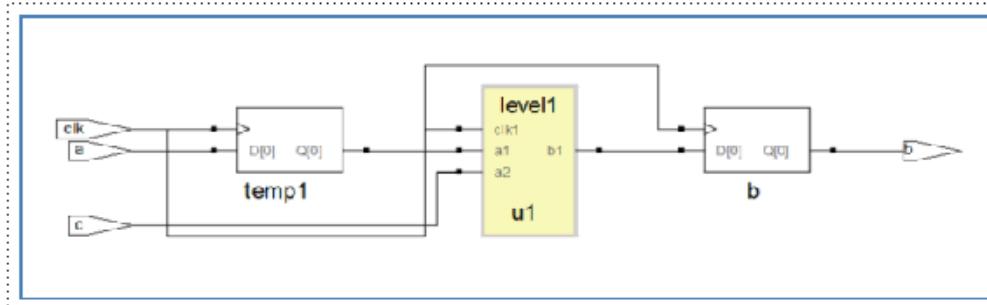
- For automatic interface timing, the tool derives constraints from the top level and uses them to synthesize the compile point. The top level is synthesized at the same time as the other compile points.
- When there are compile point constraint files, the tool first synthesizes the compile point using the constraints in the compile point constraints file and then synthesizes the top level using the top-level constraints.

When it synthesizes a compile point, the tool considers all other compile points as black boxes and only uses their interface timing information. In the following figure, when the tool is synthesizing compile point A, it applies relevant timing information to the boundary registers of B and C, because it treats them as black boxes.

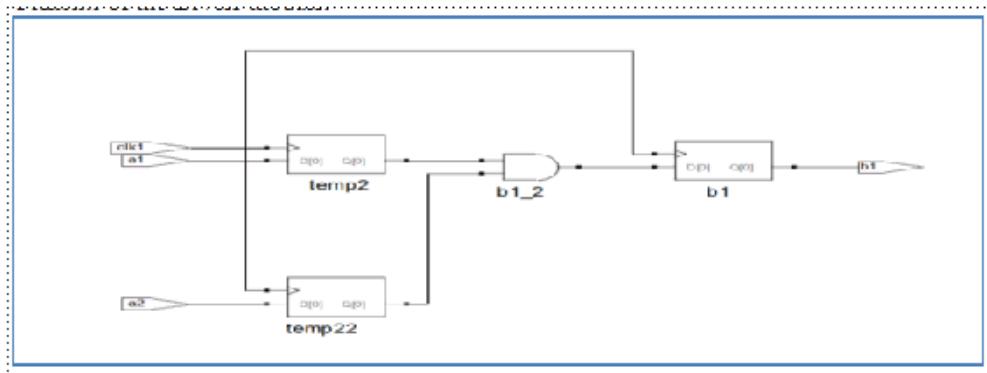


## Interface Timing Example

The design below shows how the interface timing works on compile points.



Contents of level1 Module

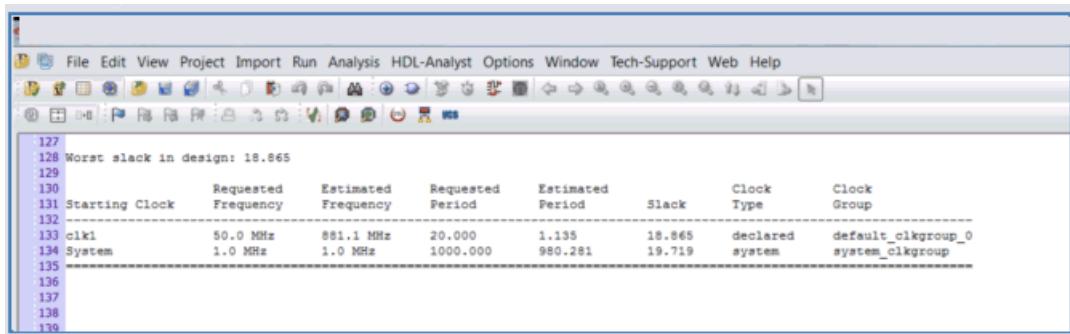


## Interface Timing Off

Interface timing is off for a compile point when you define constraints for it in a compile point constraints file. In this example, the following frequencies are defined for the level1 compile point shown above:

Clock	Period	Constraints File
Top-level clock	10 ns	Top-level constraint file
Compile point-level clock	20 ns	Compile point constraint file

When interface timing is off, the compile point log file (srr) reports the clock period for the compile point as 20 ns, which is the compile point period.



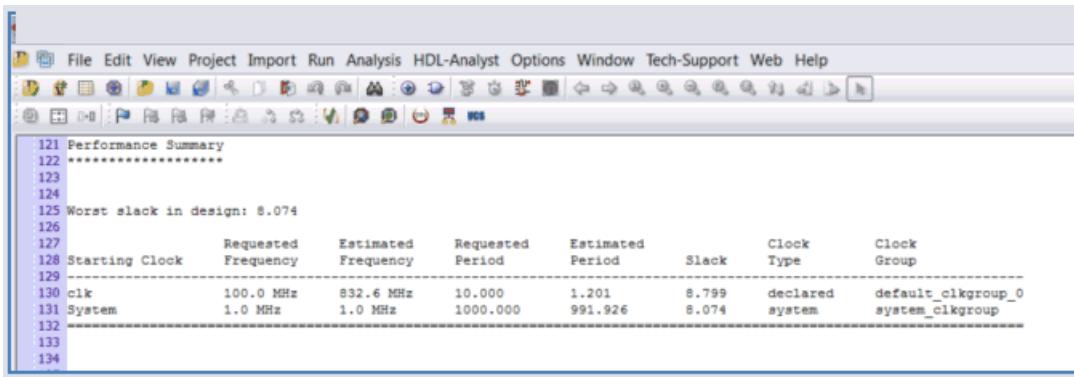
```

127
128 Worst slack in design: 18.865
129
130 Requested Estimated Requested Estimated Clock Clock
131 Starting Clock Frequency Frequency Period Period Slack Type Group
132 -----
133 clk1 50.0 MHz 881.1 MHz 20.000 1.135 18.865 declared default_clkgroup_0
134 System 1.0 MHz 1.0 MHz 1000.000 980.281 19.719 system system_clkgroup
135 -----
136
137
138
139

```

## Interface Timing On

For automatic interface timing to run on a compile point (interface timing on), there must not be a compile-point level constraints file. When interface timing is on, the compile point log file (srr) reports the clock period for the top-level design, which is 10 ns:



```

121 Performance Summary
122 ****
123
124
125 Worst slack in design: 8.074
126
127 Requested Estimated Requested Estimated Clock Clock
128 Starting Clock Frequency Frequency Period Period Slack Type Group
129 -----
130 clk 100.0 MHz 832.6 MHz 10.000 1.201 8.799 declared default_clkgroup_0
131 System 1.0 MHz 1.0 MHz 1000.000 991.926 8.074 system system_clkgroup
132 -----
133
134

```

## Compile Point Synthesis

During synthesis, the tool first synthesizes the compile points and then maps the top level. The rest of this section describes the process that the tool goes through to synthesize compile points; for step-by-step information about what you need to do to use compile points, see [Synthesizing Compile Points, on page 436](#).

## Stage 1: Bottom-up Compile Point Synthesis

The tool synthesizes compile points individually from the bottom up.

A compile point stands on its own, and is optimized separately from its parent environment (the compile point container or the top level). This means that critical paths from a higher level do not propagate downwards, and they are unaffected by them.

If you have specified compile point-level constraints, the tool uses them to synthesize the compile point; if not, it uses automatic interface timing propagated from the top level. For compile point synthesis, the tool assumes that all other compile points are black boxes, and only uses the interface information.

When defined, compile point constraints apply within the compile point. For manual compile points, it is recommended that you set constraints on locked compile points, but setting constraints is optional for soft and hard compile points.

By default, synthesis stops if the tool encounters an error while synthesizing a compile point. You can specify that the tool ignore the error and continue synthesizing other compile points.

## Stage 2: Top-Level Synthesis

Once all the compile points have been synthesized, the tool synthesizes the entire design from the top down, using the model information generated for each compile point and constraints defined in the top-level constraints file. You do not need to duplicate compile point constraints at a higher level, because the tool takes the compile point timing models into account when it synthesizes a higher level. Note that if you run standalone timing analysis on a compile point, the timing report reflects the top-level constraints and not the compile point constraints, although the tool used compile point level constraints to synthesize the compile point.

The software writes out a single output netlist and one constraint file for the entire design. See [Forward-annotation of Compile Point Timing Constraints, on page 435](#) for a description of the constraints that are forward-annotated.

## Incremental Compile Point Synthesis

The tool treats compile points as blocks for incremental synthesis. On subsequent synthesis runs, the tool runs incrementally and only resynthesizes those compile points that have changed, and the top level. The synthesis tool automatically detects design changes and resynthesizes compile points only if necessary. For example, it does not resynthesize a compile point if you only add or change a source code comment, because this change does not really affect the design functionality.

The tool resynthesizes a compile point that has already been synthesized, in any of these cases:

- The HDL source code defining the compile point is changed in such a way that the design logic is changed.
- The constraints applied to the compile point are changed.
- Any of the options on the Device panel of the Implementation Options dialog box, except Update Compile Point Timing Data, are changed. In this case the entire design is resynthesized, including all compile points.
- You intentionally force the resynthesis of your entire design, including all compile points, with the Run -> Resynthesize All command.
- The Update Compile Point Timing Data device mapping option is enabled and at least one child of the compile point (at any level) has been remapped. The option requires that the parent compile point be resynthesized using the updated timing model of the child. This includes the possibility that the child was remapped earlier, while the option was disabled. The newly enabled option requires that the updated timing model of the child be taken into account, by resynthesizing the parent.

For each compile point, the software creates a subdirectory named for the compile point, in which it stores intermediate files that contain hierarchical interface timing and resource information that is used to synthesize the next level. Once generated, the model file is not updated unless there is an interface design change or you explicitly specify it. If you happen to delete these files, the associated compile point will be resynthesized and the files regenerated.

## Forward-annotation of Compile Point Timing Constraints

In addition to a top-level constraint file, each compile point can have its own constraint file. Constraints are forward-annotated to placement and routing from the top-level as well as the compile point-level files. However, not all compile point constraints are forward-annotated, as explained below. For example, constraints on top-level ports are always forward annotated, but compile point port constraints are not forward annotated.

- Top-level constraints are forward-annotated.
- Constraints applied to the interface (ports and bit ports) of the compile point are not forward-annotated.

These include `input_delays`, `output_delays`, and `clock` definitions on the ports. Such constraints are only used to map the compile point itself, not its parents. They are not used in the final timing report, and they are not forward-annotated.

- Constraints applied to instances inside the compile point are forward-annotated
- Constraints like timing exceptions and internal clocks are used to map the compile point and its parents. They are used in the final timing report, and they are forward-annotated.

# Synthesizing Compile Points

This section describes the synthesis process with manual compile points:

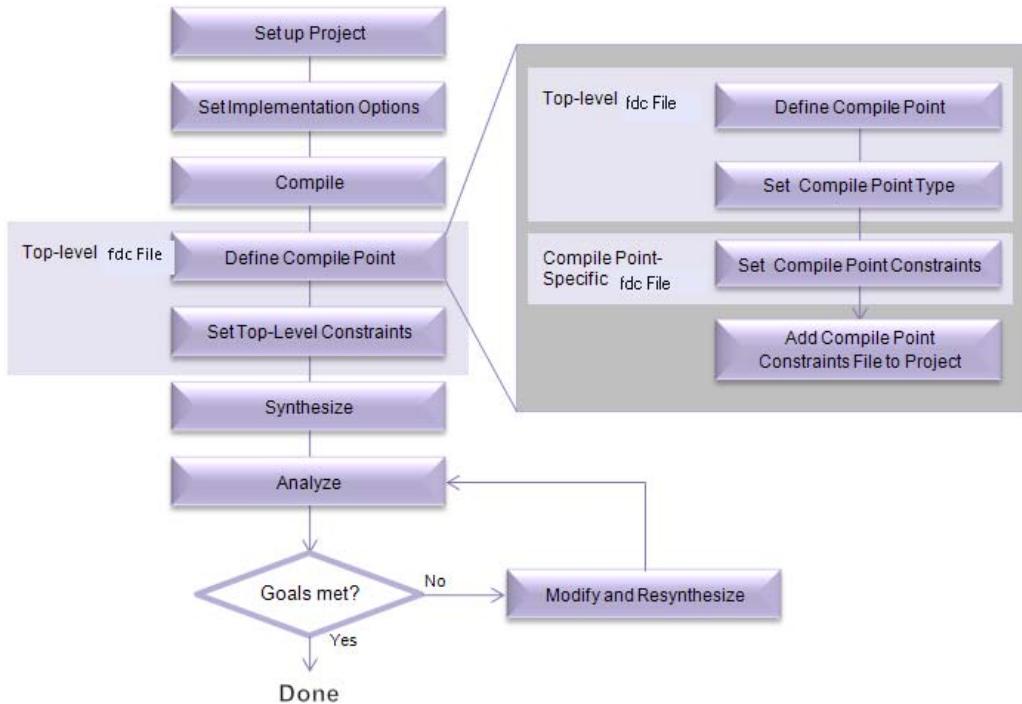
- [The Manual Compile Point Flow](#), on page 436
- [Creating a Top-Level Constraints File for Compile Points](#), on page 439
- [Defining Manual Compile Points](#), on page 441
- [Setting Constraints at the Compile Point Level](#), on page 444
- [Analyzing Compile Point Results](#), on page 446

## The Manual Compile Point Flow

Using manual compile points is most advantageous in the following situations, where you

- Have to work with a large design
- Experience long runtimes, or need to reduce synthesis runtime
- Require the maximum QoR from logic synthesis
- Can adjust design methodology to get the best results from the tools

The following figure summarizes the process for using manual compile points in your design.



This procedure describes the steps in more detail:

1. Set up the project.
  - Create the project and add RTL and IP files to the project, as usual.
  - Target a device and technology for which compile points are supported.
  - Set other options as usual.
2. Compile the design (F7) to initialize the constraints file.

3. Do the following in the top-level constraint file:

- Define compile points in the top-level constraint file. See [Creating a Top-Level Constraints File for Compile Points, on page 439](#). Note that by default, the tool automatically calculates the interface timing for all compile points.
- Set timing constraints and attributes in the top-level constraint file:

Constraint	Apply to...	Example
Clock	All clocks in the design.	<code>create_clock {p:clk} -name clk -period 100 -clockgroup cg1</code>
I/O constraints	All top-level port constraints. Register the compile point I/O boundaries to improve timing.	<code>set_input_delay {p:a} {1} -clock {clk:r}</code>
Timing exceptions	All timing exceptions that are outside the compile point module, or that might be partially in the compile point modules.	<code>set_false_path -from {i:reg1} -to {i:reg2}</code>
Attributes	All attributes that are applicable to the rest of the design, not within the compile points.	<code>define_attribute {i:statemachine_1} syn_encoding {sequential}</code>

4. Set compile point-specific constraints as needed in a separate, compile point-level constraint file.

See [Setting Constraints at the Compile Point Level, on page 444](#) for a step-by-step procedure. After setting the compile point constraints, add the compile point constraint file to the project.

5. If you do not want to interrupt synthesis for compiler errors, select Options->Configure Compile Point Process and enable the Continue on Error option.

With this option enabled, the tool black boxes any compile points that have mapper errors and continues to synthesize the rest of the design.

6. Synthesize the design.

The tool synthesizes the compile points separately and then synthesizes the top level. See [Compile Point Synthesis, on page 432](#) for details about the process.

- The first time it runs synthesis, the tool maps the entire design.
- For subsequent synthesis runs, the tool only maps compile points that were modified since the last run. It preserves unchanged compile points.

You can also run synthesis on individual compile points, without synthesizing the whole design.

7. Analyze the synthesis results using the top-level srr log file.

See [Analyzing Compile Point Results, on page 446](#) for details.

8. If you do not meet your design goals, make necessary changes to the RTL, constraints, or synthesis controls, and re-synthesize the design.

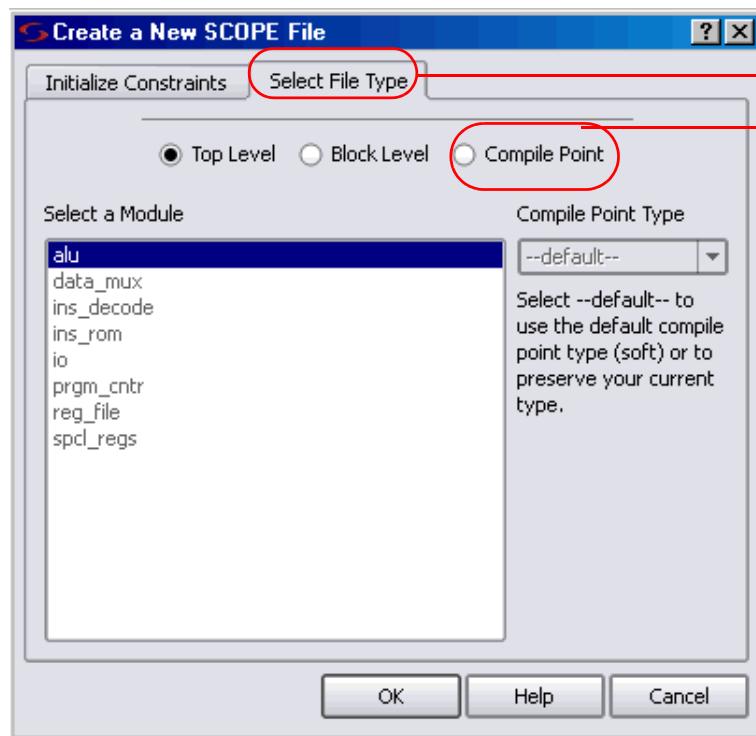
The tool runs incremental synthesis on the modified parts of the design, as described in [Incremental Compile Point Synthesis, on page 434](#). See [Resynthesizing Compile Points Incrementally, on page 449](#) for a detailed procedure.

## Creating a Top-Level Constraints File for Compile Points

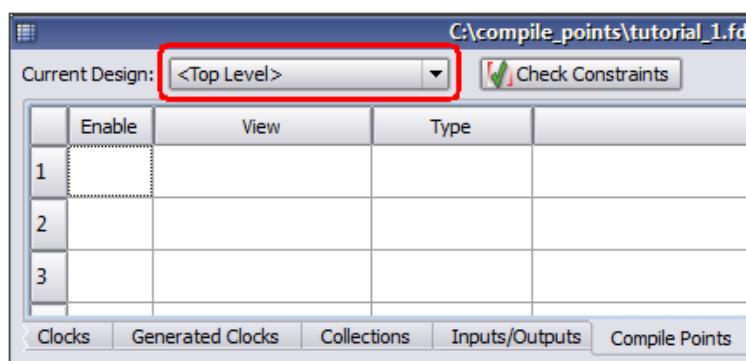
All compile points require a top-level constraints file. If you have manual compile points, define them in this file. The top-level file also contains design-level constraints. The following procedure describes how to create a top-level constraints file for a compile point design.

1. Create the top-level constraints file.
  - To define compile points in an existing top-level constraint file, open a SCOPE window by double-clicking the file in the Project view.
  - To define compile points in a new top-level constraint file, click the SCOPE icon. Click on the FPGA Constraints (SCOPE) button.

The SCOPE window opens. It includes a Current Design field, where you can specify constraints for the top-level design from the drop-down menu and define manual compile points.



Click and select



2. Set top-level constraints like input/output delays, clock frequencies or multicycle paths.

You do not have to redefine compile point constraints at the top level as the tool uses them to synthesize the compile points.

3. Define manual compile points if needed.

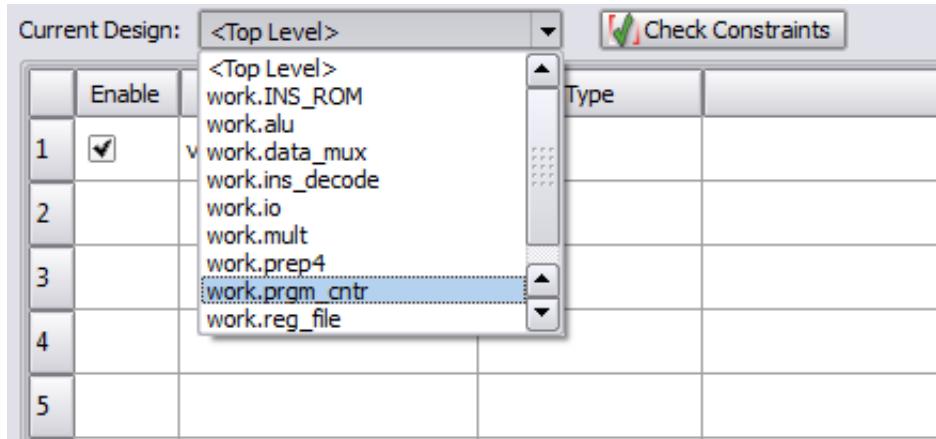
See [Defining Manual Compile Points, on page 441](#) for details.

4. Save the top-level constraints file and add it to the project.

## Defining Manual Compile Points

Compile points and constraints are both saved in a constraint file, so this step can be combined with the setting of constraints, as convenient. This procedure only describes how to define compile points. You define compile points in a top-level constraint file. You can add the compile point definitions to an existing top-level constraint file or create a new file.

1. From the Current Design field, select the module for which you want to create the compile point.



1. Click the Compile Points tab in the top-level constraints file.

See [Creating a Top-Level Constraints File for Compile Points, on page 439](#) if you need information about creating this file.

2. Set the module you want as a compile point.

Do this by either selecting a module from the drop-down list in the View column, or dragging the instance from the HDL Analyst RTL view to the

View column. The equivalent Tcl command is `define_compile_point`, as shown in this example:

```
define_compile_point {v:work.m3} -type {black_box}
```

You can get a list of all the modules from which you can select and designate compile points with the Tcl `find` command, as shown here:

```
c_print [find -hier -view * -filter (@is_verilog==1 ||@is_vhdl==1)] -file view.txt
```

3. Set the Type to locked, hard, or soft, according to your design goals. See [Defining the Compile Point Type, on page 442](#) for details.

This tags the module as a compile point. The following figure shows the `prgm_cntr` module set as a locked compile point:

	Enabled	Module	Type	Comment
1	<input checked="" type="checkbox"/>	v:work.prgm_cntr	locked	
2		v:work.prep4		
3		v:work.ins_decode		
3		v:work.prgm_cntr		
4		v:work.reg_file		
5		v:work.data_mux		
5		v:work.alu		
6		v:work.mult		
6		v:work.spcl_regs		
7		v:work.io		
7		v:work.INS_ROM		

4. Save the top-level constraint file.

You can now open the compile point constraint file and define constraints for the compile point, as needed for manual compile points. See [Setting Constraints at the Compile Point Level, on page 444](#) for details.

## Defining the Compile Point Type

The compile point type you select depends on your design goals. For descriptions of the various compile point types, see [Compile Point Types, on page 422](#). This procedure shows you how to set the compile point type in the top-level constraint file when you define the compile points:

1. When runtime is the main objective and QoR is not a primary concern, set the compile point type as follows on the SCOPE Compile Points tab:

Situation	Compile Point Type
RTL is almost ready	locked
RTL is still being built but the module interface is ready	black_box If you define black box compile points, you must update the value and rerun synthesis after you have completed the RTL for the compile point.

The following example shows the Tcl command and the equivalent version in the in the SCOPE GUI:

```
define_compile_point {v:work.user_top} -type {locked}
```

Enabled	Module	Type
1 <input checked="" type="checkbox"/>	v:work.user_top	locked <input type="button" value="▼"/>

- When runtime and QoR are both important, do the following to ensure the best performance while still saving runtime:
  - Register the I/O boundaries for the compile points.
  - As far as possible, put the entire critical path into the same compile point.
  - Set each compile point type individually, using these compile point types:

Situation	Compile Point Type
Need boundary optimizations	soft
Do not need boundary optimizations	locked

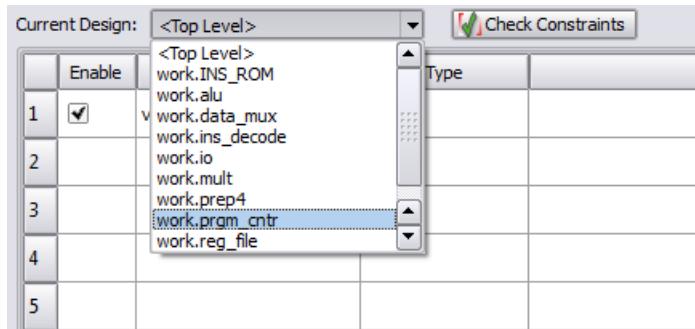
- If your goal is design preservation, set the compile point you want to preserve to locked.

## Setting Constraints at the Compile Point Level

You can specify constraints for each compile point in individual constraint files. (See [Compile Point Constraint Files, on page 426](#) for a description of the files.) It is recommended that you specify constraints for each locked manual compile point, but you do not need to set them for soft and hard compile points.

When you specify compile point constraints, the tool synthesizes the compile point using the compile point timing models instead of automatic interface timing from the top level. This procedure explains how to create a (compile point constraint file, and set constraints for the compile point:

1. In an open project, click the SCOPE icon (  ). Click on the FPGA Constraints (SCOPE) button. The New Constraints File dialog box opens.
2. From the Current Design field, select the module for which you want to create the compile point.



3. Check that you are in the right file.

A default name for the compile point file appears in the banner of the SCOPE window. Unlike the top-level constraint file, the Compile Point tab in the SCOPE UI is greyed out when the constraint file is for a compile point.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock	Comment
	Clocks	Clock to Clock	Collections	Inputs/Outputs	Registers	Delay Paths	Attributes	I/O Standards	Other			
1	<input checked="" type="checkbox"/>	clock		120	8.33333	default_clkgroup_0					<input type="checkbox"/>	
2												
3												

4. Set constraints for the compile point. In particular, do the following:
  - Define clocks for the compile point.
  - Specify I/O delay constraints for non-registered I/O paths that may be critical or near critical.
  - Set port constraints for the compile point that are needed for top-level mapping.

The tool uses the compile point constraints you define to synthesize the compile point. Compile point port constraints are not used at the parent level, because compile point ports do not exist at that level.

You can specify SCOPE attributes for the compile point as usual. See [Using Attributes with Compile Points, on page 446](#) for some exceptions.

5. Save the file and add it to the project. When prompted, click Yes to add the constraint file to the top-level design project.

Otherwise, use Save As to write a file such as, *moduleName.fdc* to the current directory. The hierarchical paths for compile point modules in the constraint file are specified at the compile point level; not the top-level design.

## Using Attributes with Compile Points

You can use attributes as usual when you set constraints for compile points. The following sections describe some caveats and exceptions:

- `syn_hier`

When you use `syn_hier` on a compile point, the only valid value is `flatten`. All other values of this attribute are ignored for compile points. The `syn_hier` attribute behaves normally for all other module boundaries that are not defined as compile points.

- `syn_allowed_resources`

Apply the `syn_allowed_resources` attribute globally or to a compile point to specify its allowed resources. When a compile point is synthesized, the resources of its siblings and parents cannot be taken into account because it stands alone as an independent synthesis unit. This attribute limits dedicated resources such as block RAMs or DSPs that the compile point can use, so that there are adequate resources available during the top-down flow.

## Analyzing Compile Point Results

The software writes all timing and area results to a single log file in the implementation directory. You can check this file and the RTL and Technology views to determine if your design has met the goals for area and performance. You can also view and isolate the critical paths, search for and highlight design objects and crossprobe between the schematics and source files.

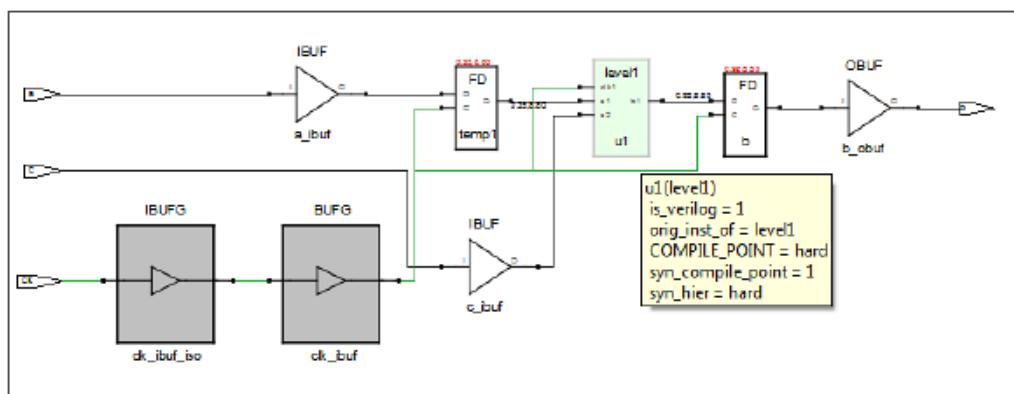
1. Check that the design meets the target frequency for the design. Use the Watch window or check the log file.
2. Open the log file and check the following:
  - Check top-level and compile point boundary timing. You can also check this visually using the RTL and Technology view schematics. If you find negative slack, check the critical path. If the critical path crosses the compile point boundary, you might need to improve the compile point constraints.
  - If the design was resynthesized, check the Summary of Compile Points section to see if compile points were preserved or remapped.

Summary of Compile Points :						
Name	Status	Reason	Start Time	End Time	Runtime	CPU Time
user_top	Renamed	Mapping options changed	Mon Mar 14 04:43:42 2011	Mon Mar 14 04:44:27 2011	0h:00m:45s	0h:00m:33s

Note that this section reports black box compile points as Not Mapped, and lists the reason as Black Box.

- Review all warnings and determine which should be addressed and which can be ignored.
  - Review the area report in the log file and determine if the cell usage is acceptable for your design.
  - Check all DRC information.
3. Check other files:
    - Check the individual compile point module log files. The tool creates a separate directory for each compile point module under the implementation directory. Check the compile point log file in this directory for synthesis information about the compile point synthesis run.
    - Check the compile point timing report. This report is located in the compile point results directory of the implementation directory for each compile point.
  4. Check the RTL and Technology view schematics for a graphic view of the design logic. Even though instantiations of compile points do not have unique names in the output netlist, they have unique names in the Technology view. This is to facilitate timing analysis and the viewing of critical paths.

**Note:** Compile point of type {hard} is easily located in the Technology view with the color green.



## 5. Fix any errors.

Remember that the mapper reports an error if synthesis at a parent level requires that interface changes be made to a locked compile point. The software does not change the compile point interface, even if changes are required to fix DRC violations.

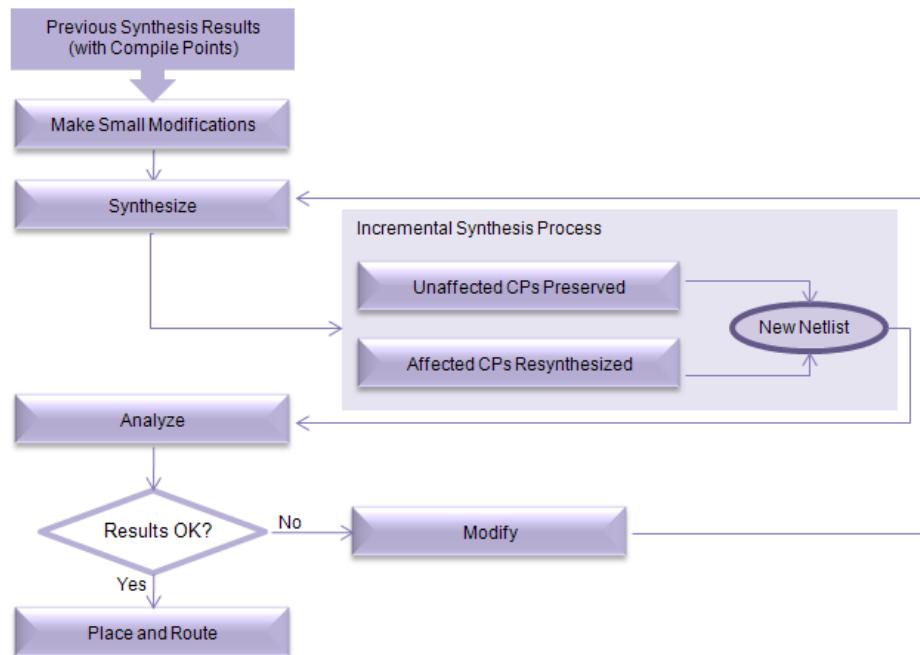
# Resynthesizing Incrementally

Incremental synthesis can significantly reduce runtime on subsequent runs. It can also help with design stabilization and preservation. The following describe the incremental synthesis process, and how compile points are used in incremental synthesis within the tool and with other tools:

- [Incremental Compile Point Synthesis](#), on page 434
- [Resynthesizing Compile Points Incrementally](#), on page 449

## Resynthesizing Compile Points Incrementally

The following figure illustrates how compile points (CP) are used in incremental synthesis.



1. To synthesize a design incrementally, make the changes you need to fix errors or improve your design.
  - Define new compile point constraints or modify existing constraints in the existing constraint file or in a new constraint file for the compile point. Save the file.
  - If necessary, reset implementation options. Click Implementation Options and modify the settings (operating conditions, optimization switches, and global frequency).

To obtain the best results, define any required constraints and set the proper implementation options for the compile point before resynthesizing.

2. Click Run to resynthesize the design.

When a design is resynthesized, compile points are not resynthesized unless source code logic, implementation options, or constraints have been modified. If there are no compile point interface changes, the software synthesizes the immediate parent using the previously generated model file for the compile point. See [Incremental Compile Point Synthesis, on page 434](#) for details.

3. Check the log file for changes.

The following figure illustrates incremental synthesis by comparing compile point summaries. After the first run, a syntax change was made in the mult module, and a logic change in the comb\_logic module. The figure shows that incremental synthesis resynthesizes comb\_logic (logic change), but does not resynthesize mult because the logic did not change even though there was a syntax change. Incremental synthesis re-uses the mapped file generated from the previous run to incrementally synthesize the top level.

Summary of Compile Points		
Name	Status	Reason
mult	Mapped	No database
comb_logic	Mapped	No database
alu	Mapped	No database
eight_bit_uc	Mapped	No database

=====

Incremental Run Log Summary		
Summary of Compile Points		
Name	Status	Reason
mult	Unchanged	-
comb_logic	Remapped	Design changed
alu	Unchanged	-
eight_bit_uc	Unchanged	-

=====

- To force the software to generate a new model file for the compile point, click Implementation Options on the Device tab and enable Update Compile Point Timing Data. Click Run.

The software regenerates the model file for each compile point when it synthesizes the compile points. The new model file is used to synthesize the parent. The option remains in effect until you disable it.

- To override incremental synthesis and force the software to resynthesize all compile points whether or not there have been changes made, use the Run->Resynthesize All command.

You might want to force resynthesis to propagate changes from a locked compile point to its environment, or resynthesize compile points one last time before tape out. When you use this option, incremental synthesis is disabled for the current run only. The Resynthesize All command does not regenerate model files for the compile points unless there are interface changes. If you enable Update Compile Point Timing Data and select Resynthesize All, you can resynthesize the entire design and regenerate the compile point model files, but synthesis will take longer than an incremental synthesis run.



## CHAPTER 12

# Clock Conversion

---

ASICs are often implemented in FPGAs for prototyping. The inherent differences between ASICs and FPGAs can make this conversion difficult. One particular source of problems can be attributed to the complex clocking circuitry of FPGAs that often includes a large number of gated and internally generated clocks. The Synopsys FPGA synthesis tools provide two features, *Gated Clock Conversion* and *Generated Clock Conversion*, to address the complex clocking schemes.

These features move the generated-clock and gated-clock logic from the clock pin of a sequential element to its enable pin. Relocating these clocks allows the sequential elements to be tied directly to the skew-free source clock and also reduces the number of clock sources in the design which frees up routing resources and expedites placement and routing. Dedicated FPGA clock trees are routed to every sequential device on the die and are designed with low skew to avoid hold-time violations. Using these global clock trees allows the programmable routing resources of the FPGA to be used primarily for logic interconnect and simplifies static timing analysis because checks for hold-time violations based on minimum delays are unnecessary.

For details, refer to the following topics:

- [Working with Gated Clocks](#), on page 454
- [Optimizing Generated Clocks](#), on page 484

# Working with Gated Clocks

This section first describes the gated-clock solution. The information is organized into the following sections:

- [Obstacles to Conversion](#), on page 456
- [Prerequisites for Gated Clock Conversion](#), on page 457
- [Defining Clocks Properly](#), on page 459
- [Synthesizing a Gated-Clock Design](#), on page 464
- [Accessing the Clock Conversion Report](#), on page 465
- [Analyzing the Clock Conversion Report](#), on page 466
- [Interpreting Gated Clock Error Messages](#), on page 469
- [Disabling Individual Gated Clock Conversions](#), on page 472
- [Using Gated Clocks for Black Boxes](#), on page 476
- [OR Gates Driving Latches](#), on page 477
- [Obstructions to Optimization](#), on page 478
- [Unsupported Constructs](#), on page 480
- [Other Potential Workarounds to Solve Clock-Conversion Issues](#), on page 480
- [Restrictions on Using Gated Clocks](#), on page 480

The clock conversion solution separates the gating from the clock inputs, and combines individual clock trees on the dedicated FPGA global clock trees. The software logically separates the gating from the clock and routes the gating to the clock enables on the sequential devices, using the programmable routing resources of the FPGA. The software separates a clock net going through an AND, NAND, OR, or NOR gate by doing one of the following:

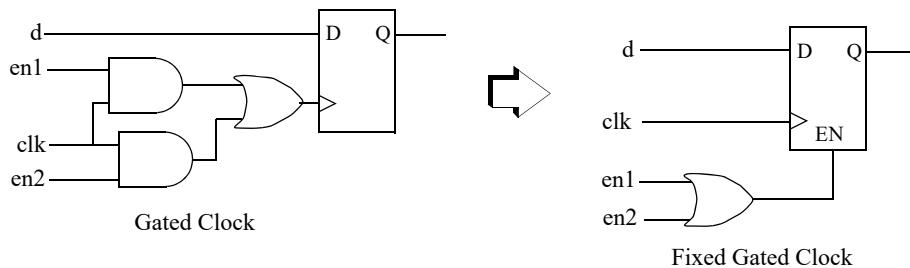
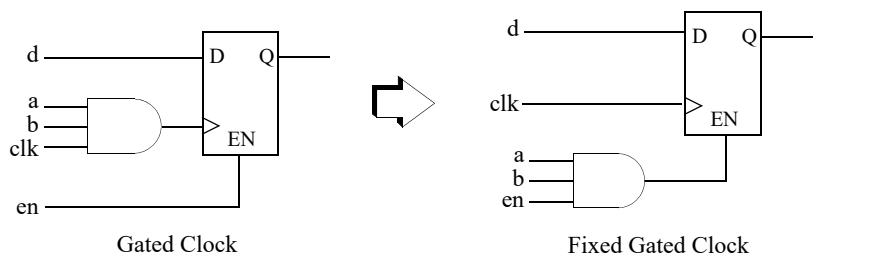
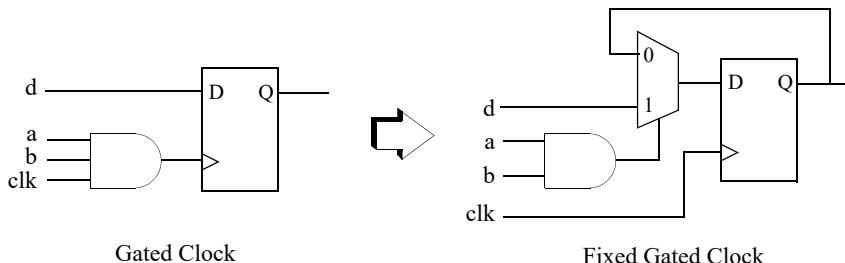
- Inserting a multiplexer in front of the input pin of the synchronous element and connecting the clock net directly to the clock pin
- Moving the gating from the clock input pin to the dedicated enable pin, when this pin is available.

The ungated or base clock is routed to the clock inputs of the sequential devices using the global FPGA clock resources. Typically, many gated clocks are derived from the same base clock, so separating the gating from the clock allows a single global clock tree to be used for all gated clocks that reference that base clock.

It is not always necessary to convert all clocks from an ASIC design – fitting the design into the FPGA device and meeting timing are the ultimate goals. However, some situations require unconverted clock structures to be modified such as:

- Clock structures with a large number of loads resulting in congestion issues
- Clock structures resulting in a large clock skew between the starting and ending clocks on a path that does not meet timing

The following figure illustrates several cases of how gated clocks are converted.



## Obstacles to Conversion

The following is a summary of issues to look for that can prevent a clock structure from being converted. More details about each condition are described later.

- Are the clocks defined properly?
- Is the Clock Conversion option enabled?
- Does the clock structure contain instantiated cells?

- Is something in the clock logic blocking optimization?
- Are there any unsupported constructs?

## Prerequisites for Gated Clock Conversion

For a gated clock to be converted successfully, the design must meet these requirements:

Condition	Description
Combinational logic only	The gated clock logic must consist only of combinational logic. A derived clock that is the output of a register is not converted.
Single base clock	Identify only one input to the combinational logic for the gated clock as a base clock. To identify a net as a clock, specify a period or frequency constraint for either the gate or the clock in the constraint file. This example defines the <code>clk</code> input as the base clock: <code>create_clock -name {clk} -freq 10.000 -clockgroup default_clkgroup</code>
Supported primitives	The sequential primitive clocked by the gated clock must be a supported object. The tools support gated-clock conversion for most sequential primitives. Black-box modules driven by gated clocks can be converted if special synthesis directives are used to define the clock and clock enable inputs to the black box. See <a href="#">Using Gated Clocks for Black Boxes, on page 476</a> .
Correct logic format	See <a href="#">Correct Logic Format, on page 457</a> for an example of the correct logic format.

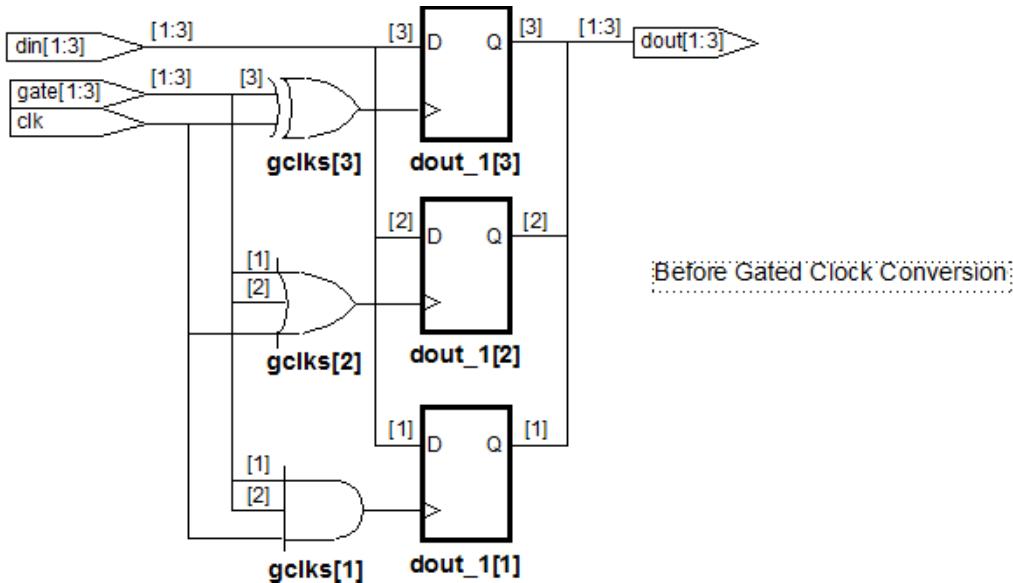
## Correct Logic Format

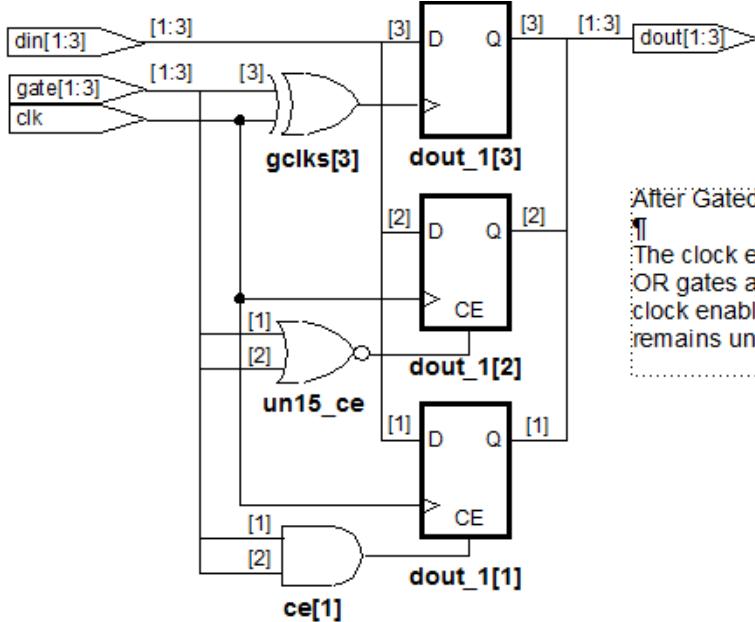
Specifically, the combinational logic for the gated clock must satisfy the following two conditions to have the correct format:

- For at least one set of gating input values, the value output for the gated clock must be constant and not change as the base clock changes.
- For at least one value of the base clock, changes in the gating input must not change the value output for the gated clock.

The correct logic format requirements are illustrated with the simple gates shown in the following figures. When the software synthesizes a design with the Gated Clocks option enabled, clock enables for the AND gate and OR gate are converted, but the exclusive-OR gate shown in the second figure is not converted. The following table explains.

AND gate gclks[1]	If either gate[1] or gate[2] is 0, then gclks[1] is 0, independent of the value of clk which satisfies the first condition. Also, if clk is 0, then gclks[1] is 0, independent of the values of gate[1] and gate[2] which satisfies the second condition. Because gclks[1] satisfies both conditions, it is successfully converted to the clock-enable format.
OR gate gclks[2]	If either gate[1] or gate[2] is 1, then gclks[2] is 1 independent of the value of clk which satisfies the first condition. Also, if clk is 1, then gclks[2] is 1 independent of the value of gate[1] or gate[2] which satisfies the second condition. Because gclks[2] satisfies both conditions, it is successfully converted to the clock-enable format.
Exclusive-OR gate gclks[3]	Irrespective of the value of gate[3], gclks[3] continues to toggle. The exclusive-OR function causes gclks[3] to fail both conditions which prevents gclks[3] from being converted.





After Gated Clock Conversion:

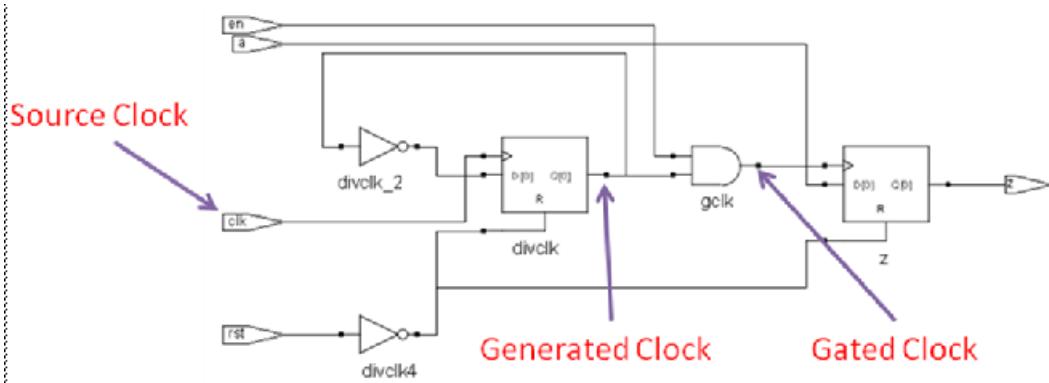
The clock enables for the AND and OR gates are converted, but the clock enable for the exclusive OR remains unchanged. §

## Defining Clocks Properly

The most common problems preventing clock conversion from occurring automatically are related to improperly defined clocks. When defining gated and generated clocks:

- Gated clocks should be defined at the nodes to be connected to the clock pins of the sequential cells.
- Use `create_generated_clock` constraints to define the relationship between generated clocks and their sources.
- If a clock circuit contains both a clock generator and a gating element, a `create_generated_clock` constraint is needed. Without this constraint, the tool does not know the relationship between the gated signal and the source clock.

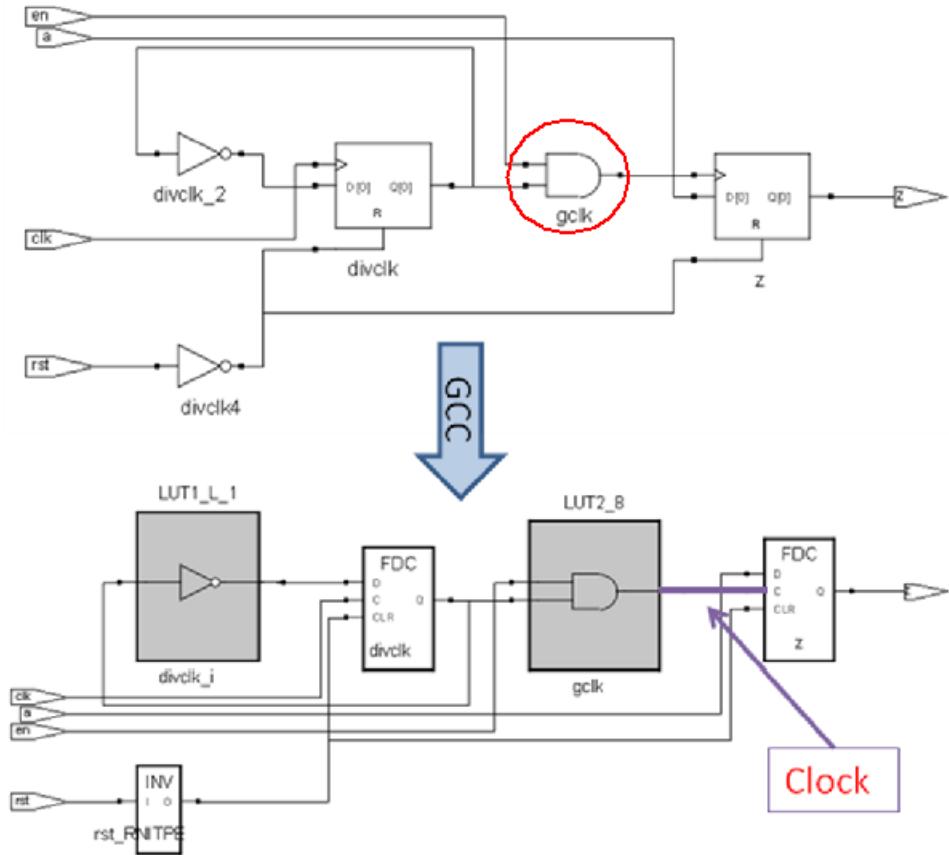
## Defining Clocks Example



Using the above example, the first constraint applied is:

```
create_clock -name clk [get_ports clk] -period 10
```

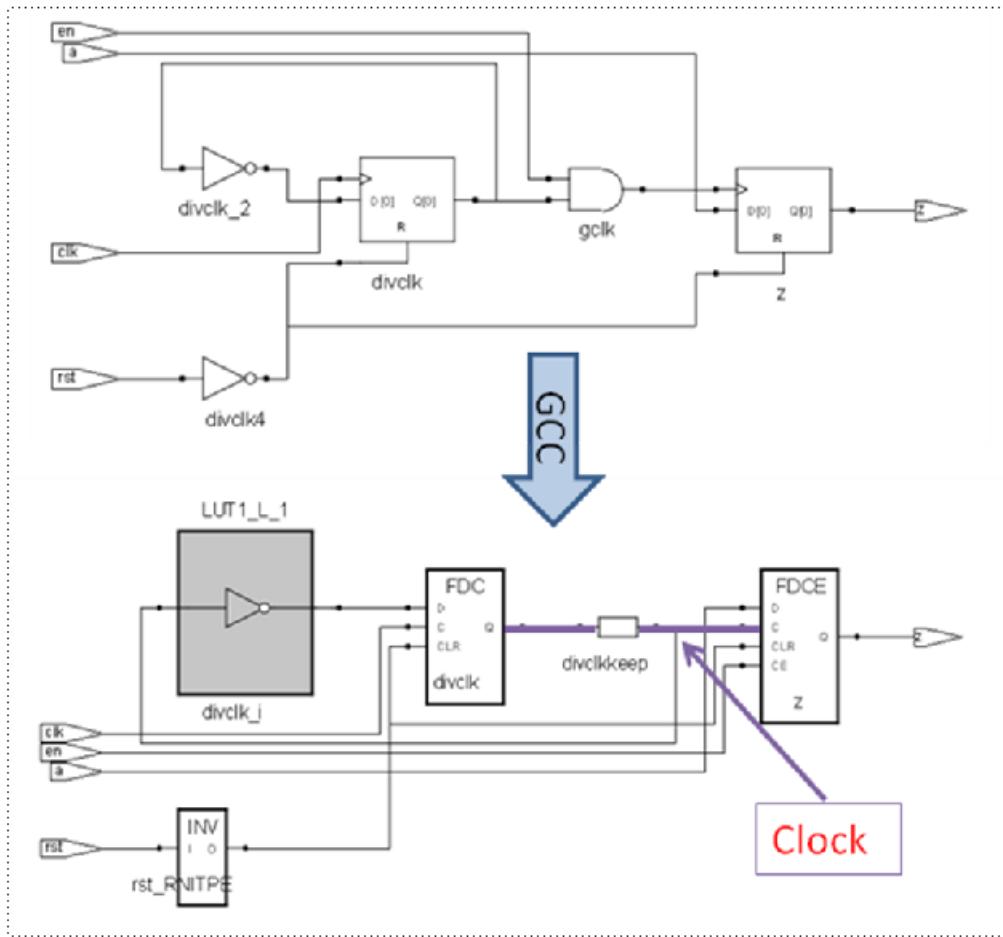
In the above constraint, `get_ports` identifies `clk` as a port. With only the source clock defined, the tool is unable to determine which input of the AND gate is the clock and which input is the enable. Accordingly, no conversion is performed as shown in the following schematic diagram.



Continuing with the example, a second constraint is added:

```
create_clock -name clk [get_ports clk] -period 10
create_clock -name divclk [get_nets {n:divclk}] -period 20
```

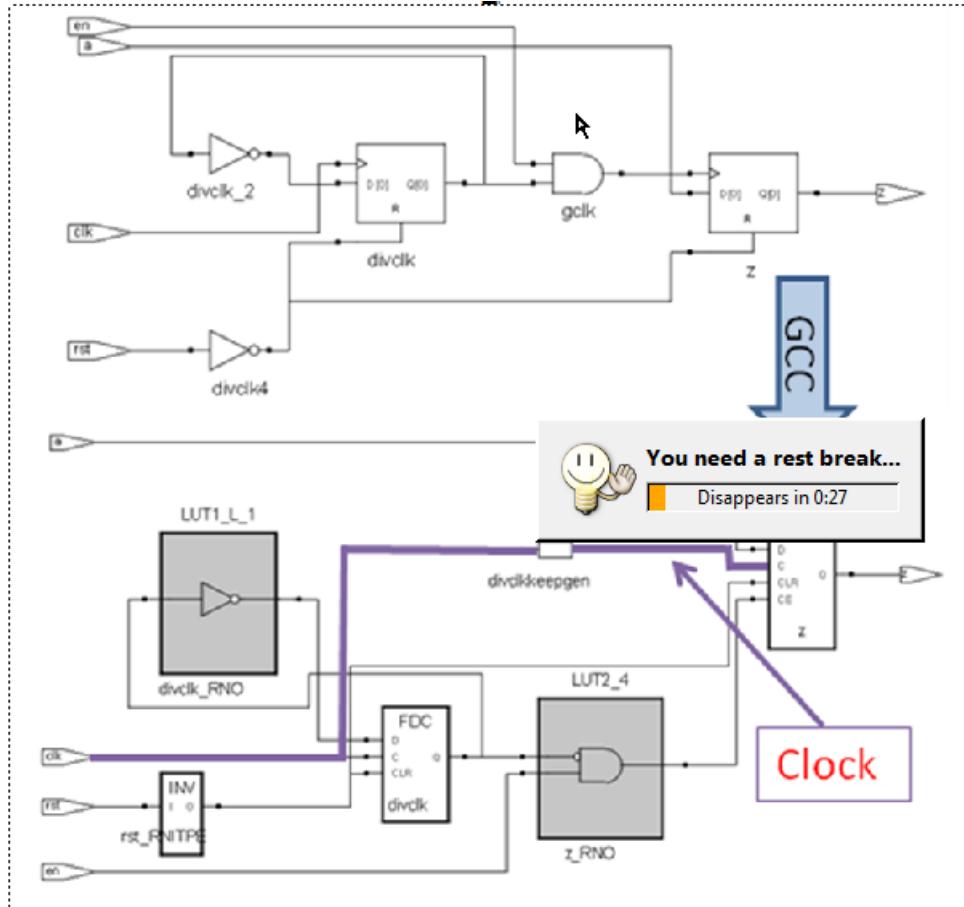
In the second constraint, `get_nets` identifies `divclk` as a net. With independent clock constraints defining the source and generated clocks, the tool now knows which input of the AND gate is the clock, but still does not know the relationship between the source and generated clocks. In this case, the clock gate and data register 'Z' are converted to an enable flip-flop (FDCE) which is connected to the generated clock as shown in the next schematic.



For the final example, the second clock constraint is replaced with a generated clock constraint.

```
create_clock -name clk [get_ports clk] -period 10
create_generated_clock -name divclk [get_nets {n:divclk}]
-source [get_ports clk] -divide_by 2
```

The tool knows which input of the AND gate is the clock and the relationship between the source and generated clocks. For this case, the entire clock circuit is converted to an enable on the 'Z' register, and the 'Z' register clock pin is connected directly to the source clock as shown in the final schematic.



## Internal Clocks

Applying internal clock constraints to output pins of inferred combinational logic (for example, an AND gate) is not supported. Apply clock constraints only to nets driven by logical output pins.

## Only One Clock per Clock Pin Fan-in Cone

Every fan-in cone of the clock pin of a sequential device should have one defined clock. As long as one clock is defined, the clock can be traced forward through unate logic to the clock pins.

When no clock is defined:

- The clock source is derived by tracing the clock pin back to the first non-trivial (multiple input) gate
- Because the tool does not know which input of a multiple-input cell to trace, a clock is inferred at the output of that cell
- Since inferred clocks were not defined by the user explicitly or implicitly, no conversion occurs

When more than one clock is defined, the tool does not know which clock to convert, and no conversion occurs.

## Synthesizing a Gated-Clock Design

To synthesize a gated-clock design:

1. Make sure that the gated clocks have the correct logic format and satisfy the prerequisites for conversion. See [Prerequisites for Gated Clock Conversion, on page 457](#) for details.
2. If the gated clock drives a black box, specify the clock and the associated clock enable signal with using `syn_force_seq_prim`, `syn_isclock`, and `syn_gatedclk_en` directives. See [Using Gated Clocks for Black Boxes, on page 476](#) for details.
3. Make sure that the clock net has a constraint specified in the constraint file for the current implementation. If you do not specify an explicit constraint on the clock net or if you set a global frequency constraint, enabling clock conversion as described in the next step will not have any effect.
4. Enable the Clock Conversion option.
  - Select Project->Implementation Options.
  - On the GCC tab, click on the Clock Conversion check box.

5. Synthesize the design. For gated clocks, the option converts qualified flip-flops, counters, latches, synchronous memories, and instantiated technology primitives. The software logically separates the gating from the clock and routes the gating to the clock enables on the sequential devices, using the programmable routing resources of the FPGA. The ungated base clock is routed to the clock inputs of the sequential devices using the global clock resources. Because many gated clocks are normally derived from the same base clock, separating the gating from the clock allows a single global clock tree to be used for all gated clocks that reference the same base clock.

See [Restrictions on Using Gated Clocks, on page 480](#) for additional information.

6. Check the results in the START OF CLOCK OPTIMIZATION REPORT section of the log file. See [Analyzing the Clock Conversion Report, on page 466](#) for an example of this report.

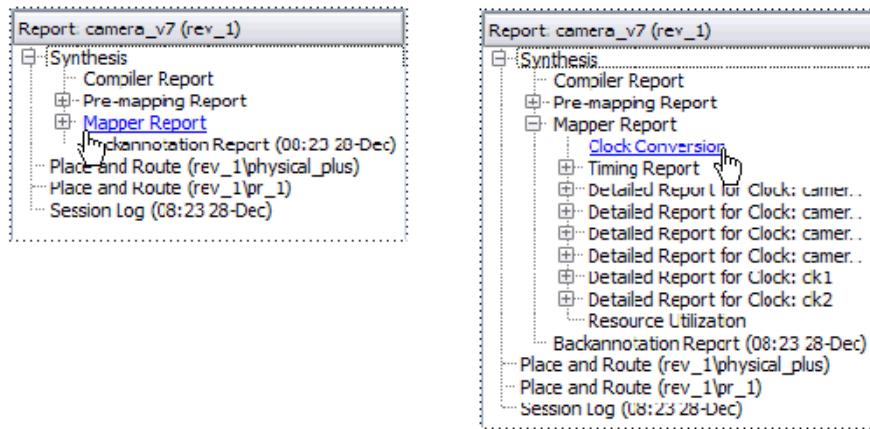
## Accessing the Clock Conversion Report

The clock conversion report can be accessed from the log file or from the Project Status tab.

### Log File Access

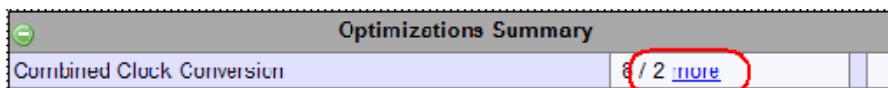
To access the clock conversion report directly in the log file:

1. Open the log file by clicking the View Log button.
2. In the browser, expand the Mapper Report entry by clicking the adjacent '+' sign and click on the Clock Conversion link to go directly to the clock conversion report. Note that the browser is only available when the View log file in HTML option is enabled.



## Project Status Tab Access

To access the clock conversion report directly from the Project Status tab, click the more link in the Optimizations Summary section at the bottom of the display.



## Analyzing the Clock Conversion Report

When clock conversion is enabled, a typical report resembles the following:

### START OF CLOCK OPTIMIZATION REPORT #####					3-line Summary
8 non-gated/non-generated clock tree(s) driving 184 clock pin(s) of sequential element(s)					
2 gated/generated clock tree(s) driving 16 clock pin(s) of sequential elements					
10 instances converted, 16 sequential instances remain driven by gated/generated clocks					
Clock Tree ID	Driving Element	Driven Element Type	Fanout	Sample Instances	
<b>Non-gated/Non-generated Clock Tree Table</b>					
<a href="#">ClockID0000</a>	<a href="#">wr_rlk_i</a>	port	215	<a href="#">camera_wishbone_if_dcam_addr_out[31]</a> <a href="#">control_synchronization_beyne_active_hi_sync</a> <a href="#">mem_vld_...</a>	
<a href="#">ClockID0004</a>	<a href="#">hi_pg_inn=2</a>	RREG	160		
<a href="#">ClockID0005</a>	<a href="#">ictrgds_inst</a>	IBUF	1		
<a href="#">ClockID0006</a>	<a href="#">ictrgq_inst3</a>	SBUF			
<a href="#">ClockID0007</a>	<a href="#">ictrgq_inst1</a>	SBUF			
<a href="#">ClockID0008</a>	<a href="#">clk1</a>	port			
<a href="#">ClockID0009</a>	<a href="#">clk2</a>	port	1	<a href="#">soc_file_u1_ff_out[7]</a>	
<a href="#">ClockID0010</a>	<a href="#">clk3</a>	port	1	<a href="#">clk_top_inst.clk_tree2.cl_ff_clk_u1.clk_out</a> <a href="#">clk_top_inst.req_ff_u4_ff_out</a>	
Clock Tree ID	Driving Element			Gated/generated Clock Tree Table	Sample Instances
<a href="#">ClockID0001</a>	<a href="#">clk_top_inst.clk_tree2.cl.com</a>			<a href="#">clk_top_inst.req_file_u2_ff_out[0]</a>	
<a href="#">ClockID0002</a>	<a href="#">clk_top_inst.clk_tree3.cl.q</a>	----		<a href="#">clk_top_inst.req_file_u3_ff_out[1]</a>	
##### END OF CLOCK OPTIMIZATION REPORT #####					

## 3-line Summary

The 3-line summary provides a quick snapshot of the design.

- The first line reports the number of *clean* clock trees (8 in the above report) and the number of driven clock pins (434).
- The next line reports the number of gated/generated clock trees (2 in the above report) and the number of driven clock pins (16).
- The last line reports the number of instances converted (10 in the above report) and the number of remaining instances driven by gated/generated clocks (16).

The following `find` command can be used to group the converted instances:

```
find -hier -inst * -filter @syn_gcConverted==1
```

## Non-gated/non-generated Clocks Table

The non-gated/non-generated clocks section provides details of the *clean* clocks in the design and also includes any clock trees that never included gating logic. This section reports the following:

- Clock Tree ID – created for all instances in the clock tree that lead to the sample instance. Clicking on an ID opens a filtered technology view of the source clock.
- Driving Element – identifies the source of the driving clock.
- Drive Element Type – identifies the type of drive element such as a port.
- Fanout – lists the clock fanout.
- Sample Instance – an instance within the clock tree.

The following find command can be used to identify all instances in the clock path:

- `find -hier -inst * -filter @syn_sample_clock_path==CKIDxxxx`

If a clock is defined within the cone-of-logic for the gated clock, it is reported in this table.

## Gated/generated Clocks Table

The gated/generated clocks section lists a sample failure within each clock tree in the design with an explanation of why the conversion failed (see the following section [Interpreting Gated Clock Error Messages, on page 469](#)). The gated/generated clocks section reports the following:

- Clock Tree ID – created for all instances in the clock tree that lead to the failed sample instance. Clicking on an ID opens a filtered technology view of the source clock.
- Driving Element – identifies the source of the driving clock.
- Drive Element Type – identifies the type of drive element such as a LUT.
- Fanout – lists the clock fanout.
- Sample Instance – an instance within the clock tree.

In the table, failures with common gating-control signals are only reported once. The software traverses back from the clock pin of the sample instance until it reaches the driving element at the point of the failure.

## Interpreting Gated Clock Error Messages

The following table describes the gated clock conversion error messages reported in the Gated/Generated Clocks section of the clock conversion report.

Error Message	Explanation
Asynchronous set/reset mismatch prevents generated clock conversion	Unshared asynchronous signals have been detected between a FF-derived clock circuit and its sequential load.
Can't determine input clock driver	Self-explanatory. Specify the driver.
Clock conversion disabled	Gated clocks have been detected, but clock conversion is not enabled.
Clock propagation blocked by fixed hierarchy	Clock property has been blocked by a fixed hierarchy which prevents clock conversion from propagating upstream.
Clock propagation blocked by hard hierarchy	Clock property has been blocked by a hard hierarchy which prevents clock conversion from propagating upstream.
Clock propagation blocked by locked hierarchy	Clock property has been blocked by a locked hierarchy which prevents clock conversion from propagating upstream.
Clock propagation blocked by syn_keep	Clock property has been blocked by a syn_keep property which prevents clock conversion from propagating upstream.
Clock source is constant	Self-explanatory.
Combinational loop in clock network	Self-explanatory. Fix the combination loop.
FF-derived clock conversion disabled	FF-derived clocks have been detected, but clock conversion is not enabled.
Gating structure creates improper gating logic	Self-explanatory. Analyze and fix logic.
Illegal instance on clock path	Self-explanatory. Analyze and fix logic.
Inferred clock from port	Self-explanatory notification.
Input clock depends on output	Self-explanatory.

Error Message	Explanation
Latch gated by OR originally on clock tree	Self-explanatory.
Multiple clock inputs on sequential instance	Self-explanatory. Analyze and fix logic.
Multiple clocks on generating sequential element	Self-explanatory. Analyze and fix logic.
Multiple clocks on instance	Multiple clocks found feeding an instance of a clock tree.
Need declared clock or clock from port to derive clock from ff	Self-explanatory. Specify the clock.
No clocks found on inputs	No clocks found feeding an instance of a clock tree.
No generated or derived clock directive on output of sequential instance	Self-explanatory. Specify the clock.
No hierarchical driver	Self-explanatory. Specify the driver.
Signal from port	Self-explanatory.
Unconverted clock gate	Self-explanatory.
Unable to determine clock driver on net	Self-explanatory. Specify the clock driver.
Unable to determine clock input on sequential instance	Self-explanatory. Analyze and fix logic.
Unable to follow clock across hierarchy	Self-explanatory. Analyze and fix logic.
Unable to use latch as gated clock generator	Self-explanatory. Analyze and fix logic.

## Checking the Log file

Many clock constraint problems can be identified by reviewing the Clock Summary table in the pre-mapping report section and the Performance Summary section of the log file shown below:

- Clock conversion is attempted on declared, generated, and derived clocks, and is never performed on inferred clocks or on the system clock.
- Declared, generated, and derived clocks can still have issues such as unsupported structures in their clock logic or incorrect constraints.
- Inferred clocks and the system clock are created when no clocks are defined in the fan-in logic of the clock pin of a sequential cell.

Performance Summary			
Starting Clock	Slack	Clock Type	Clock Group
camera_open_source CLKOUT0_derived_clock_CLKIN1	0.077.047	derived	Inferred_clkgroup_1
camera_open_source CLKOUT1 derived clock CLKIN1	997.915	derived	Inferred_clkgroup_1
camera_open_source camera_clock_pad_i	998.078	inferred	Inferred_clkgroup_1
camera_open_source wb_clk_i	995.467	inferred	Inferred_clkgroup_0
clk1	8.447	declared	default_clkgroup
clk2	8.300	declared	default_clkgroup
clk3	NA	declared	default_clkgroup
an3_r	NA	declared	default_clkgroup
System	NA	system	system_clkgroup

Following compilation, the software issues warnings about inferred clocks or the system clock, that include the clock name and one of the registers driven by that clock to help the user locate the issue in the HDL Analyst. The following are examples of these warnings:

- @W: MT529 : Found inferred clock *clkName* which controls *numClkPins* sequential elements including *instanceName*. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.
- @W: MT531 : Found signal identified as System clock which controls *numClkPins* sequential elements including *instanceName*. Using this clock, which has no specified timing constraint, can prevent conversion of gated or generated clocks and can adversely impact design performance.

## Disabling Individual Gated Clock Conversions

By default, enabling gated-clock conversion applies globally to a design. For critical paths, individual clocked elements can be expressly excluded from the gated-clock conversion by adding a `syn_keep` directive directly to the input clock net. The `syn_keep` attribute can be applied in the source code or in a constraint file. The following example shows a `syn_keep` attribute applied to the `gclk` net in a source file.

```
module and_gate (clk, en, a, z);
  input clk, en, a;
  output reg z;
  wire gclk /* synthesis syn_keep = 1 */;
  assign gclk = en & clk;

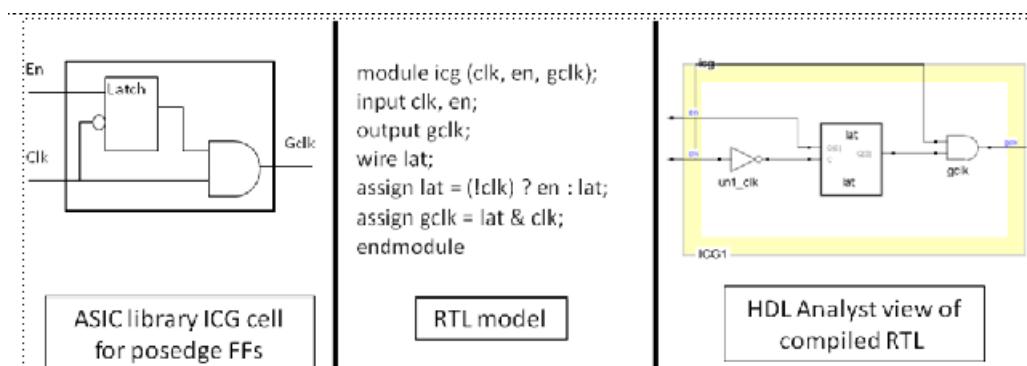
  always @(posedge gclk) z <= a;
endmodule
```

The corresponding entry in a constraint file is:

```
define_attribute {n:gclk} {syn_keep} {1}
```

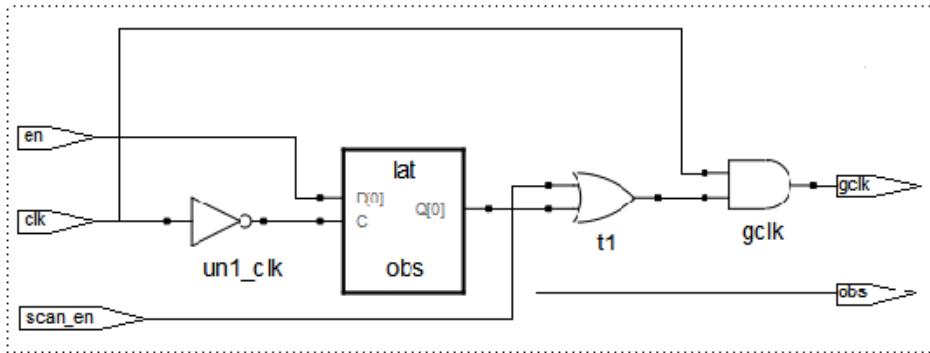
## Integrated Clock Gating Cells

The Synopsys synthesis tools do not read in ASIC cell libraries. To support conversion of Integrated Clock Gating (IGC) cells, these cells must be modeled in RTL.



If there are ASIC-only pins on an ICG cell, such as a scan-enable input or an observation output, the pins must be removed to allow the clock conversion to occur.

The following example shows an ICG cell with ASIC-only pins and the corresponding Verilog source code.

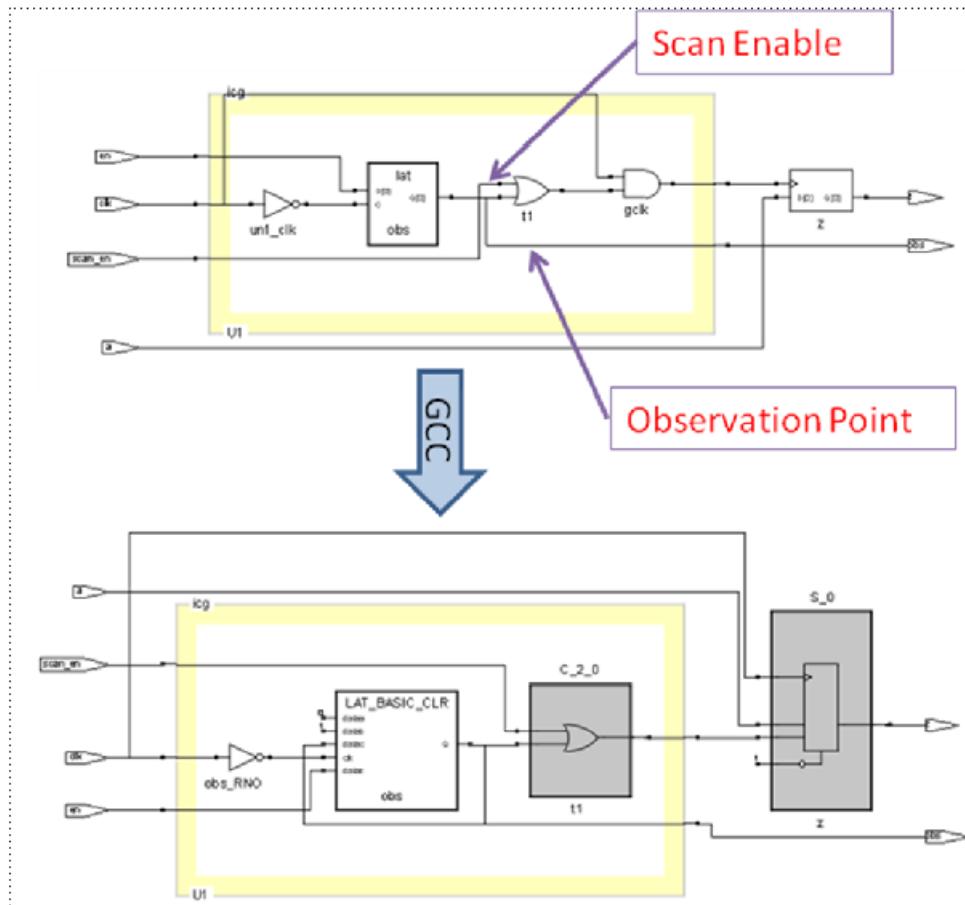


```

module icg (clk, en, scan_en, obs, gclk);
  input clk, en, scan_en;
  output obs, gclk;
  wire t0, t1;
  assign t0 = (!clk) ? en : t0;
  assign obs = t0;
  assign t1 = t0 | scan_en;
  assign gclk = t1 & clk;
endmodule

```

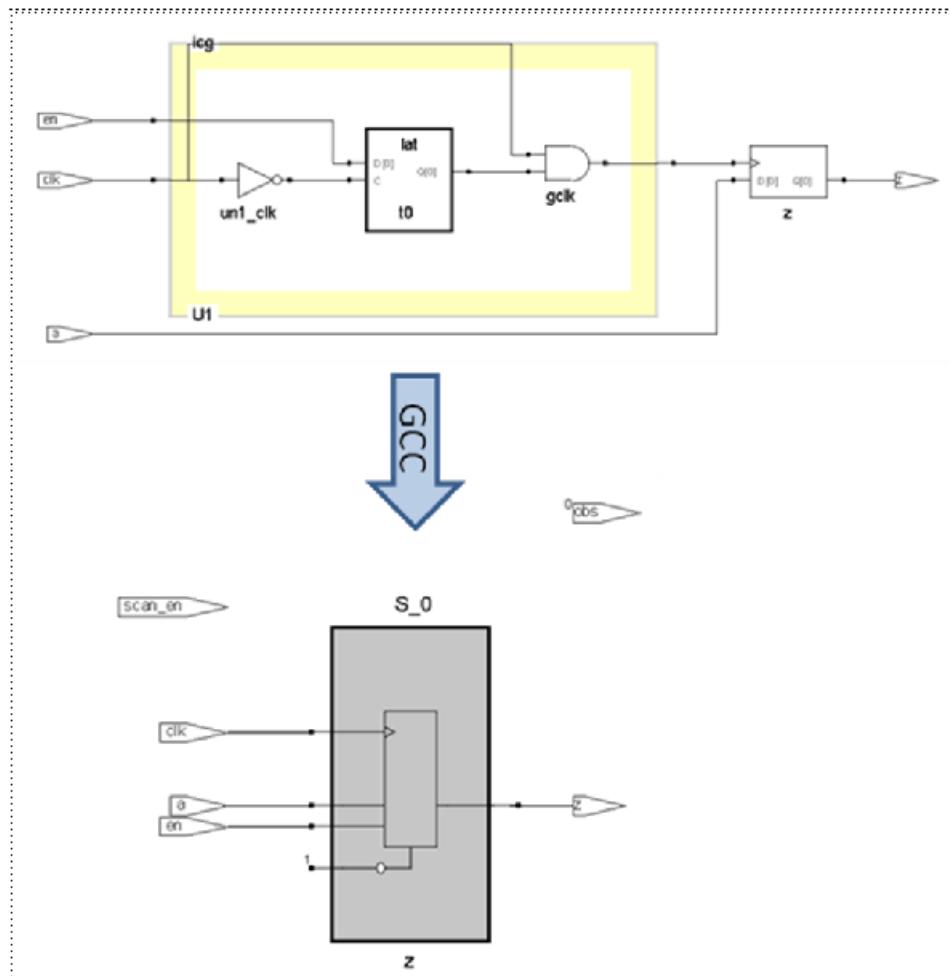
The presence of the scan control and observation point signals forces the tool to preserve intermediate signals in the clock-gating circuit that prevents it from being optimized. In this case, the data register is connected directly to the source clock, but the latch and OR gate for the ICG model remain as part of the enable logic as shown in the following schematic.



The following code simplifies the RTL model of the complex IGC cell and allows the desired gated-clock conversion to occur.

```
module icg (clk, en, scan_en, obs, gclk);
  input clk, en, scan_en;
  output obs, gclk;
  wire t0, t1;
  assign t0 = (!clk) ? en : t0;
  assign obs = 1'b0 /* t0 */;
  assign t1 = t0 | 1'b0 /* scan_en */;
  assign gclk = t1 & clk;
endmodule
```

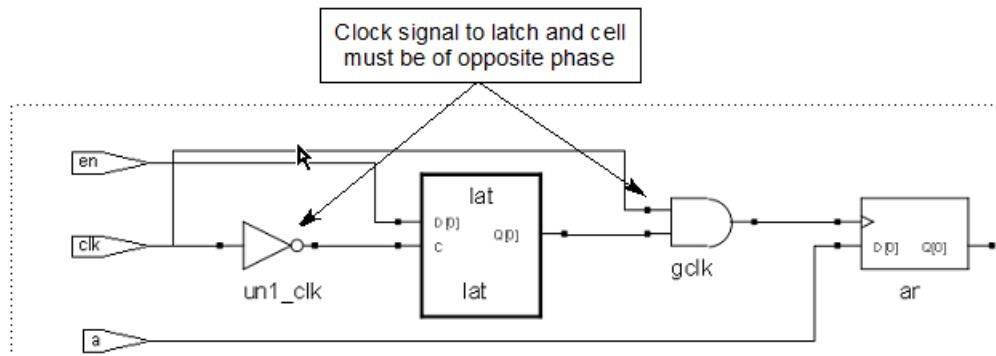
In the modified code, the `scan_en` input, which is generally not used in an FPGA implementation, is replaced with a constant '0'. The `obs` observation point output, which normally is not required in an FPGA implementation, is also tied to a constant '0'. These changes remove the intermediate points as shown in the following schematic. Note that if no other changes are made to the design, this model still "fits" because the ports are unchanged. Also, the '0' tied to the `obs` output propagates to simplify downstream logic.



## Incorrect Phase

When there is an error in the RTL model of an ICG cell such that an incorrect phase of the clock is connected to the latch:

- Only the AND gate is converted leaving the latch driving the enable pin of the data flip-flops.
- A warning is generated stating “Active phase of latch *latchName* prevents its use as a stability latch for clock gate *clockGateName*.”



## Using Gated Clocks for Black Boxes

To convert gated clocks that drive black boxes, you must identify the clock and clock enable signal inputs to the black box using the `syn_force_seq_prim`, `syn_isclock`, and `syn_gatedclk_clock_en` directives. Refer to the *Attributes Reference Manual* for information about these directives. You assume responsibility for the functionality of the black-box content.

The following are Verilog and VHDL examples of a black box with the required directives specified.

### Verilog

```
module bbe (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim="clk" */
;
input clk
```

```
/* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

## VHDL

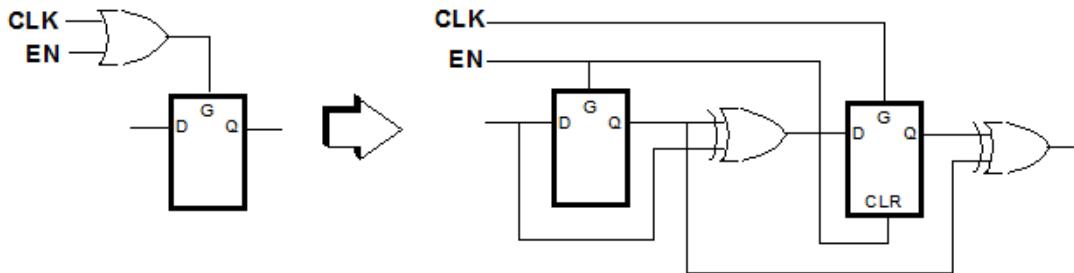
```
library synplify;
use synplify.attributes.all;

entity bbe is
    port (clk : in std_logic;
          en : in std_logic;
          data_in : in std_logic;
          data_out : out std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of clk : signal is "en";
end bbe;

architecture behave of bbe is
attribute syn_black_box : boolean;
attribute syn_force_seq_prim : string;
attribute syn_black_box of behave : architecture is true;
attribute syn_force_seq_prim of behave : architecture is "clk";
begin
end behave;
```

## OR Gates Driving Latches

To prevent problems in timing analysis when a latch is being driven by the OR of a clock and an enable, gated-clock conversion is implemented using the logic shown in the following diagram.



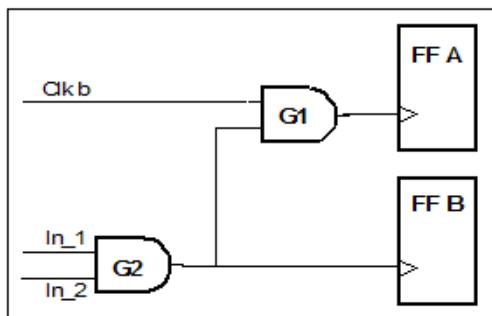
As illustrated in the above diagram, the enable (EN) and clock (CLK) feed separate latches, and XOR gates are used to condition the inputs and outputs. The conversion masks the combinational path from the timing engine and duplicates the behavior of the original circuit. Conversion only occurs if the original latch is not driving a clock network.

## Obstructions to Optimization

The following optimization obstructions in the clock structure prevent conversion:

### Inferred Clocks can Prevent Clock Conversions

Beginning with the K-2015.09-SP1 release, the presence of inferred clocks can prevent the conversion of gated clocks (as the example illustrates below) or generated clocks. As a result, rerunning a design from a previous release may perform fewer conversions than previously reported.



In the example, clk\_b is a declared clock and In\_1 and In\_2 are not declared as clocks. Previously, FF A could be a candidate for gated clock conversion. Now, this conversion cannot occur since there is also an inferred clock driving AND gate G1 (that is, the output of the AND gate G2 fed by In\_1 and In\_2 drives the clock of FF B).

## **“Keep\_buffers” in the Clock Structure**

Keep buffers are components that are added to the RTL view netlist to preserve specific nets. If a net in the clock circuitry is preserved by a keep buffer, it cannot be optimized away and can prevent conversion of the associated clock circuit. Keep buffers are the result of syn\_keep attributes and constraints placed on nets such as false path, multi-cycle path, and other clock constraints. The following Tcl find command can be used to locate the keep buffers:

```
find -hier -view keepbuf
```

## **“Hard” Hierarchy Defined on a Clock Logic Block**

A “hard” hierarchy attribute on a block containing a clock structure prevents conversion by forcing the interface of this block to remain unchanged. More specifically, if a hard hierarchy is between the clock logic and the data registers, the conversion cannot occur because the new signals for the enables cannot be added to the block interface. However, if all the clock logic is contained within the hard hierarchy, the gated clock conversion can occur. The following Tcl find command can be used to locate the hard hierarchies:

```
find -view * -filter (@syn_hier==hard)
```

## **Loads on Intermediate Nodes in the Clock Logic**

Intermediate nodes in the clock logic should not have extra loads. For conversion to occur when extra loads are present, the RTL must be modified to separate the loads that drive clock pins of data registers from the other loads.

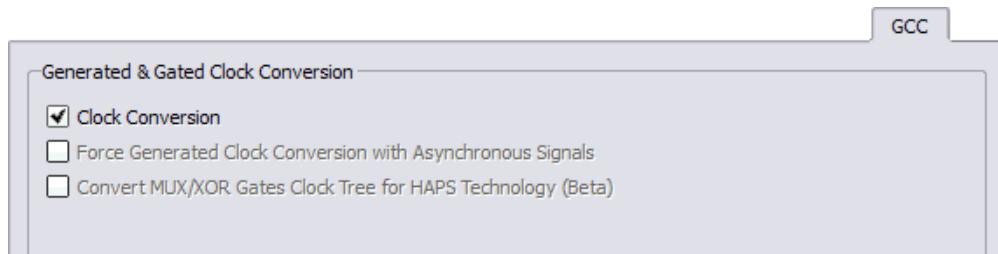
## **“Disable Sequential Optimizations” Project Option Prevents Latches from Being Removed**

For conversion to occur, the Disable Sequential Optimizations option cannot be enabled.

## Unsupported Constructs

The following are examples of constructs that prevent conversion from occurring.

- If the clock circuitry contains logic such that asynchronous set/reset signals can produce an active edge on the gated-clock line (without any change in the source clock), the circuit is not converted. Any active edges on the gated-clock output must be controlled by the source clock.
- By default, conversion only occurs if the registers in the generated-clock logic have the same asynchronous set/reset signals as the data registers that they control.



## Other Potential Workarounds to Solve Clock-Conversion Issues

If the previous sections do not help resolve your issue, there are a few more things you can try:

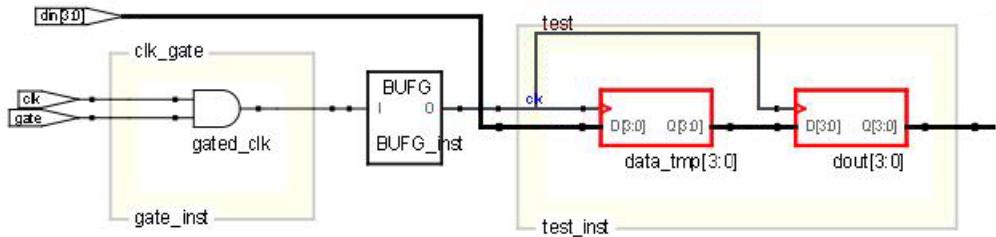
- Use a `syn_direct_enable` attribute on one of the gating signals to force the conversion.
- Recode RTL to fit one of the cases allowed for automatic conversion.

## Restrictions on Using Gated Clocks

The Clock Conversion option has the following restrictions for gated clocks:

- A `syn_keep` attribute attached to a net, which should preserve the net during optimization, is not honored by the Clock Conversion option.
- A global buffer, when instantiated on a gated clock, blocks gated-clock conversion from being performed (the buffer is viewed as the clock)

source by the downstream circuitry). In the circuit below, the clk input to test\_inst remains gated.

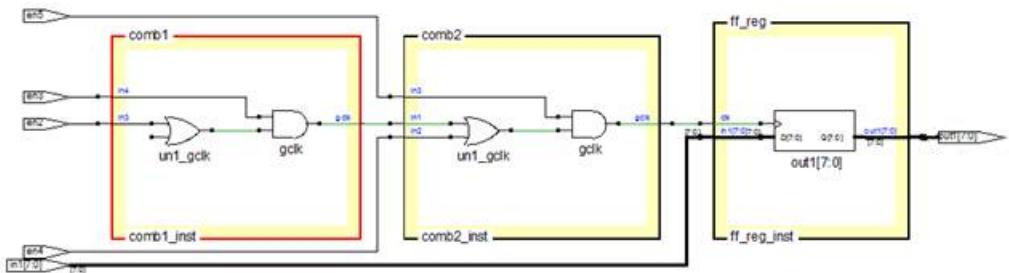


Similarly, defining a clock constraint at the output of a gated clock prevents conversion from continuing upstream.

## Issues with Gated/Generated Clock Reporting

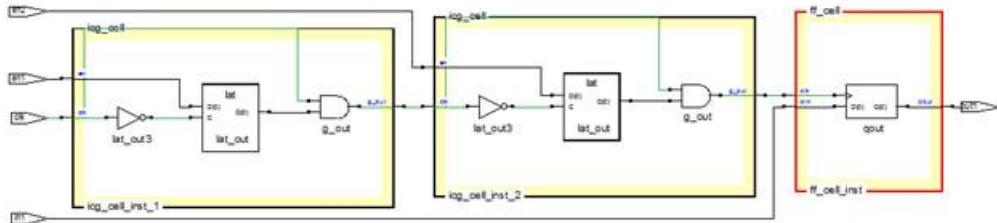
Several gated/generated clock reporting issues have been identified:

- When both inputs for XOR gates are defined as clocks, the software incorrectly reports the explanation as “No clocks found on input”. The correct explanation is “Improper clock gating structure.”
- When both inputs for MUXes are defined as clocks, the software incorrectly reports “Multiple clocks on instances”. The correct explanation is “Improper clock gating structure”.
- Gated clocks might not be converted for logic from a hierarchy that uses `syn_hier=hard/fixed`. In the following example, the `comb1` hierarchy has this attribute. Although the combinational logic within the `comb2` hierarchy is outside the scope of `syn_hier=hard`, the enable logic of register, `out1` does not get converted.

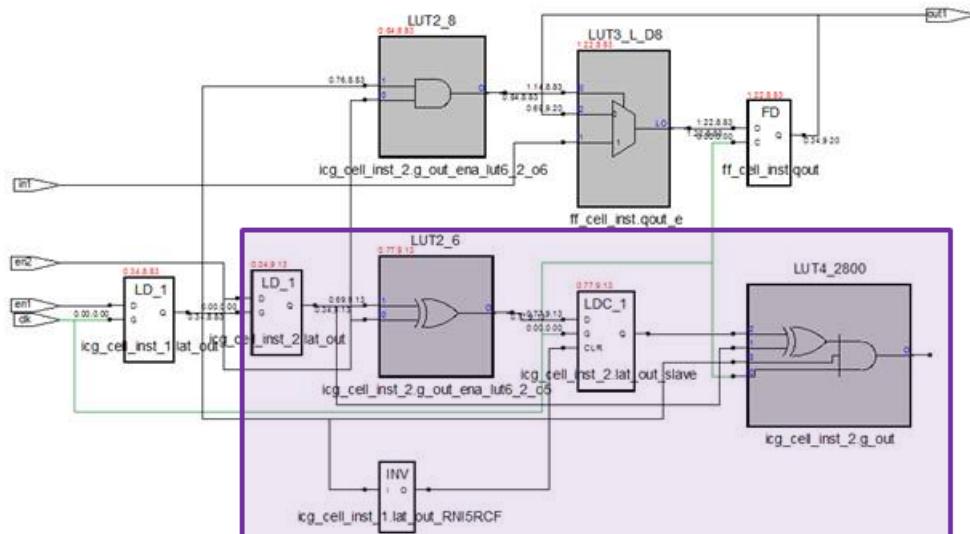


- For cascaded ICGs driving posedge flip-flops that use either compile points or the syn\_hier attribute, the tool might produce a circuit with unused logic that does not get pruned away. See the shaded logic in the following figure. Depending on how the compile point or syn\_hier boundaries are defined, the tool can report unused latches as unconverted, with the explanation “Unable to use latch as gated clock generator.”

RTL:



Post Mapping:



- The explanation “Unconverted clock gate” message indicates a gated/generated clock case that is not documented.

## Clock Conversion for Multiple Input Clocks to OR Gate Latch

If multiple clocks are defined as inputs to an OR gate driving a latch, then clock conversion does not occur and the following message is generated:  
Multiple clocks on instance. If this message occurs, make sure that only one clock is input to the OR gate driving the latch.

## Clock Conversions for the Traditional Flow

Clock conversions are handled in a traditional (non-distributed) synthesis flow as follows:

- Clock conversion runs at the pre\_map stage, on compiler primitives.
- The names of referenced objects and the numeric values in the Clock Optimization report are defined with the conditions below:
  - Driving Element/Drive Element Type references compiler objects (e.g. MUX).
  - Clock trees with  $n$  number of instances and with the same inputs will be reported  $n$  times.
- ICG latch removal is performed in conjunction with clock conversion.
  - The ICG Latch Removal Summary is listed just before the Clock Optimization Report at the pre-map stage.
- ICGs that drive latches are supported and will be converted to registers.

When the `syn_hier=hard` attribute is applied to hierarchies that “cut” a gating cone of logic, the software does not prevent clock conversion to occur.

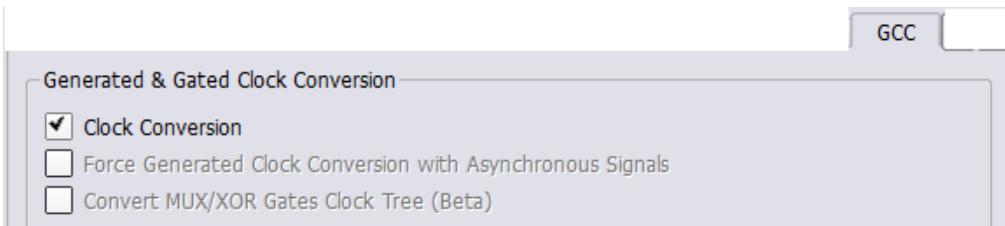
# Optimizing Generated Clocks

The Synplify Pro tool includes an option for generated clocks. When the Clock Conversion option is enabled, the generated-clock logic is replaced during synthesis with logic that uses the initial clock with an enable. With generated-clock optimization, the original circuit functionality is preserved while performance is improved by reducing clock skew. For more information, see the following topics:

- [Enabling Generated-Clock Optimization](#), on page 484
- [Conditions for Generated-Clock Optimization](#), on page 485
- [Generated-Clock Optimization Examples](#), on page 485

## Enabling Generated-Clock Optimization

Generated-clock optimization is enabled by checking Clock Conversion on the GCC tab of the Implementation Options dialog box.



The tool does not convert gated or generated clocks if the driving flip-flop has asynchronous sets or resets, unless the flip-flop or datapath latch being converted also has asynchronous controls tied to the same net as the driving flip-flop.

## Conditions for Generated-Clock Optimization

To perform generated-clock optimization, the following conditions must be met:

1. The combinational logic must be driven by flip-flops.
2. The input flip-flops cannot have an active set or reset (an active-low reset must be tied high, or an active-high set must be tied low). Similar rules apply to all the input flip-flops in the cone.
3. All input flip-flops must be driven by the same edge of the same clock.

With generated-clock optimization, you do not have to specify a primary clock.

## Generated-Clock Optimization Examples

### Example 1: Generated Clock from Combinational Logic

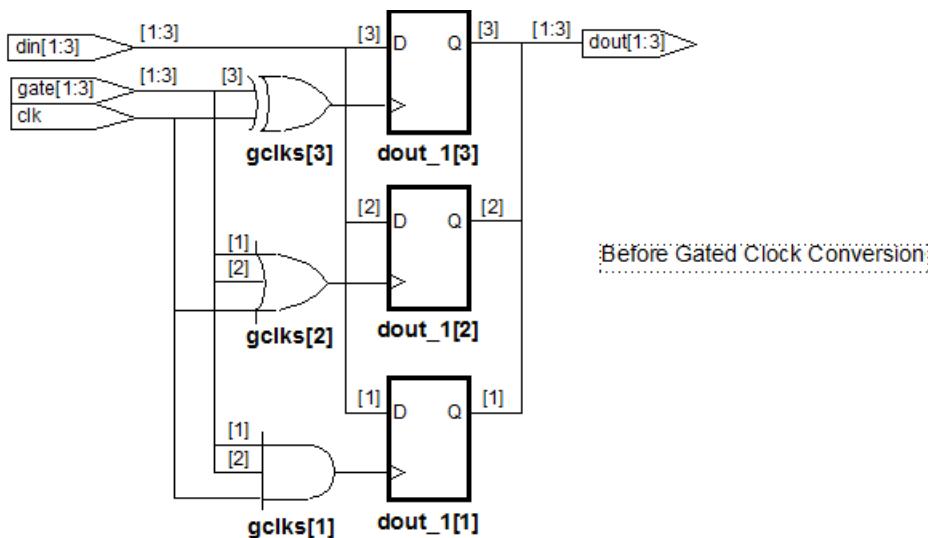
The following code segment illustrates generated-clock optimization:

```
module gen_clk(clk1,a,b,c,q);
  input clk1, a, b, c;
  output q;
  reg ao,bo,q;
  wire en;

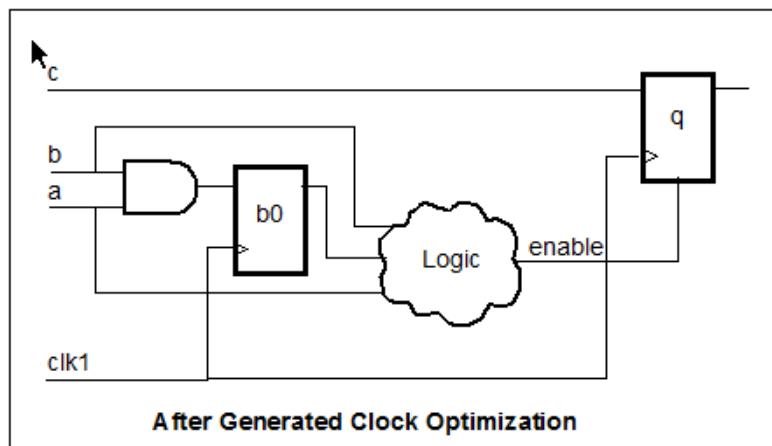
  always @(posedge clk1)
    begin
      ao <= a;
      bo <= b;
    end
  assign en = ao & bo;

  always @(posedge en)
    begin
      q <= c;
    end
endmodule
```

With generated-clock optimization disabled (Clock Conversion unchecked), the circuit in the following figure shows a flip-flop (q) driven by a generated clock that originates from the combinational logic driven by flip-flops ao and bo which, in turn, are driven by the initial clock (clk1).

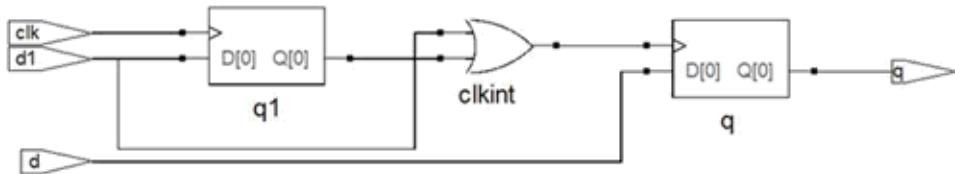


When generated-clock optimization is enabled (Clock Conversion checked), flip-flop q is replaced with an enable flip-flop. This flip-flop is clocked by the initial clock (clk1) and is enabled by combinational logic based on the a and b inputs as shown in the following figure.



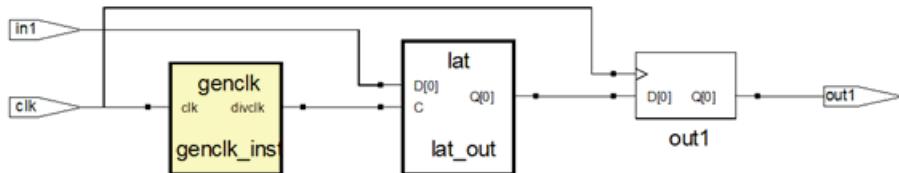
### Example 2: Generated Clock Followed by a Gate Clock

A design can have a register that is clocked by a generated clock, then followed by a gated clock. If the generated-clock constraint also includes the enable, the Clock Conversion option may optimize away the register and replace it with a constant value. When this occurs, the software converts the generated and gated-clock logic into a clock and an enable; the enable may never go active.

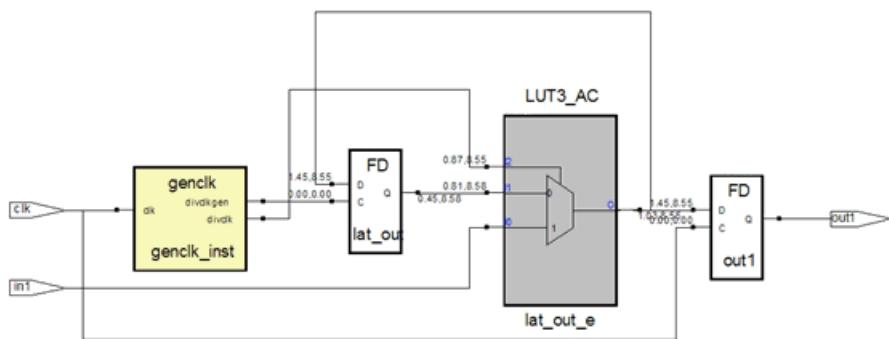


### Example 3: Datapath Latch Conversion

This example shows a latch on the datapath that is clocked by a generated clock.



GCC converts the circuit to a flip-flop and multiplexer as shown in the following figure.



## CHAPTER 13

# Optimizing for GoWin Designs

---

The following sections describe techniques for working with GoWin designs:

- [Instantiating GoWin Macros](#), on page 489
- [Combinational Logic Support](#), on page 490
- [Handling Tristates](#), on page 490

## Instantiating GoWin Macros

The synthesis tool provides GoWin macro libraries that you can use to instantiate components like I/Os, I/O pads, gates and flip-flops, RAMs, and other black box primitives.

1. To use a Verilog macro library, `gowin.v` file is automatically added to your project file when you select a GoWin target device.
2. To use a VHDL library, do the following:

Add the corresponding library and use clauses to the beginning of the design units that instantiate the macros, as in the following example:

```
library gowin;  
use gowin.components.all;
```

You do not need to add the macro library files to the source files for your project.

3. Instantiate the macros from the library as described in [Instantiating Black Boxes and I/Os in Verilog, on page 360](#) and [Instantiating Black Boxes and I/Os in VHDL, on page 362](#).

## Combinational Logic Support

The synthesis tool handles combinational logic as follows:

- Combinational logic is mapped to INV, LUT1, LUT2, LUT3, and LUT4 based on the number of inputs.
- All data path operators (adder, subtractor, comparator or multiplier) are mapped to ALU (if the data width is greater than the threshold of 4) and additional LUTs, if needed. Currently ADD, SUB, ADDSUB, and NE modes of the ALU are supported.
- MUX2\_LUT5, MUX2\_LUT6, MUX2\_LUT7, and MUX2\_LUT8 are used for wider MUX mapping.

## Limitation

Inference of LUT5, LUT6, LUT7, LUT8, MUX8, MUX16, and MUX32 is not supported.

## Handling Tristates

The synthesis tool handles tristates as follows:

- Internal tristates are not supported and are converted to MUXes whenever possible. An error is generated if an internal tristate cannot be converted to a MUX.
- Tristates connected to an output port are absorbed into the I/O pad.

# Handling I/Os and Buffers

The synthesis software handles I/Os and buffers as follows:

- IBUF/OBUF/TBUF/IOBUF primitives are used for inferring IO pads.
- If a register can be packed into IOs, the tool writes a property, `gowin_io_reg`, on the FF.

```
(*gowin_io_reg="TRUE" *) DFFE \q_z[0] (
    .Q(q_c[0]),
    .D(d_c[0]),
    .CLK(clk_c),
);
```

IO register packing is based on the timing. You can override this behavior by applying the `syn_useioff` attribute.

## Limitations

- No DDR inference support.
- No clock buffer (BUFG and BUFS) inference support.



## CHAPTER 14

# Working with Synthesis Output

---

This chapter covers techniques for optimizing your design for various vendors. The information in this chapter is intended to be used together with the information in [Chapter 9, Inferring High-Level Objects](#).

This chapter describes the following:

- [Passing Information to the P&R Tools](#), on page 494
- [Generating Vendor-Specific Output](#), on page 495

## Passing Information to the P&R Tools

The following procedures shows you how to pass information to the place-and-route tool; this information generally has no impact on synthesis. Typically, you use attributes to pass this information to the place-and-route tools. This section describes the following:

# Generating Vendor-Specific Output

The following topics describe generating vendor-specific output in the synthesis tools.

## Targeting Output to Your Vendor

You can generate output targeted to your vendor.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

The following table summarizes the outputs to set for the different vendors, and shows the P&R tools for which the output is intended.

Vendor	Output Netlist	P&R Tool
GoWin	.vm	

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

See [Specifying Result Options, on page 80](#) for details about setting the option.



## CHAPTER 15

# Running Post-Synthesis Operations

---

The following topic describes how to run post-synthesis operations.

## Simulating with the VCS Tool

The Synopsys VCS® tool is a high-performance, high-capacity Verilog simulator that incorporates advanced, high-level abstraction verification technologies into a single, open, native platform. You can launch this simulation tool from the synthesis tools on Linux and Unix platforms by following the steps below. The VCS tool does not run under the Windows operating system.

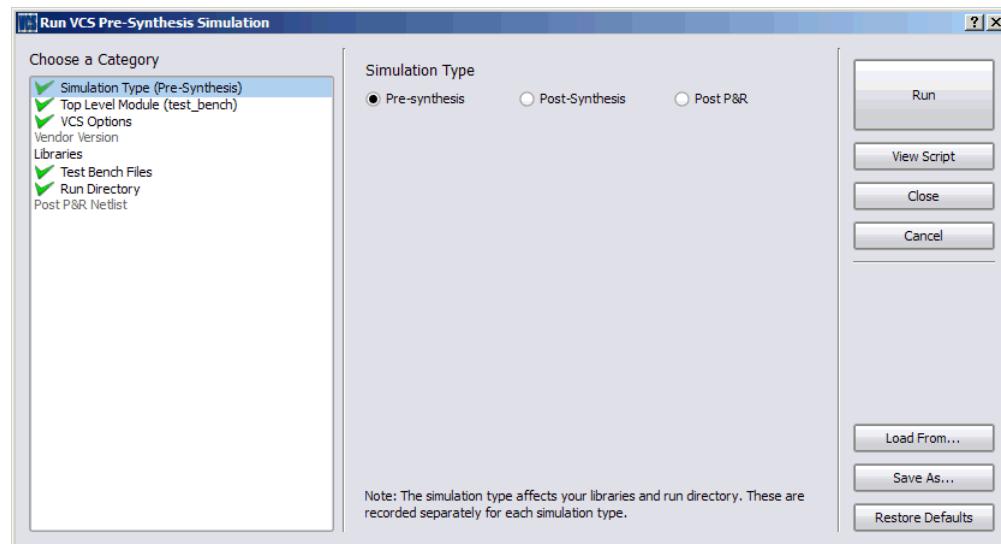
1. Set up the tools.
  - Install the VCS software and set up the \$VCS\_HOME environment variable to define the location of the software.
  - Set up the place-and-route tool.
  - In the synthesis software, either select Run->Configure and Launch VCS Simulator, or click the  icon.

If you did not set up the \$VCS\_HOME environment variable, you are prompted to define it. The Run VCS Simulator dialog box opens. For descriptions of the options in this dialog box, see [Configure and Launch VCS Simulator Command, on page 349](#) of the *Reference Manual*.

2. Choose the category Simulation Type in the dialog box to configure the simulation options.

- Specify the kind of simulation you want to run.

RTL simulation	Enable Pre-Synthesis
Post-synthesis netlist simulation	Enable Post-Synthesis
Post-P&R netlist simulation	Enable Post P&R

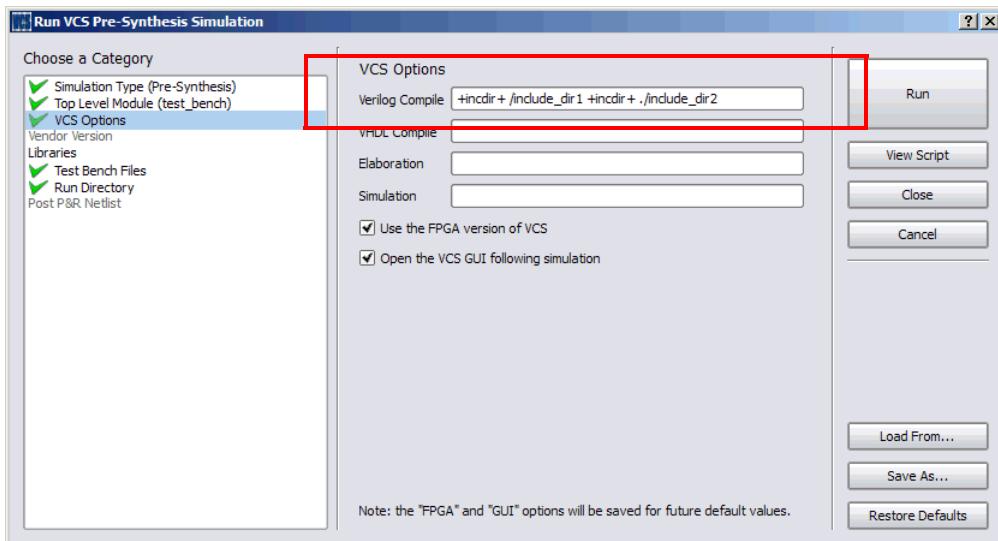


- Choose the category VCS Options in the dialog box to set options such as the following VCS commands.

To set...	Type the option in...
VLOGAN command options for compiling and analyzing Verilog, like the -q option	Verilog Compile
VHDLAN options for compiling and analyzing VHDL	VHDL Compile
VCS command options	Elaboration
SIMV command options, like -debug	Simulation

The options you set are written out as VCS commands in the script. If you leave the default settings the VCS tool uses the FPGA version of VCS and opens with the debugger (DVE) GUI and the waveform viewer. See the VCS documentation for details of command options.

3. If your project has Verilog files with `include statements, you must use the +incdir+ *fileName* argument when you specify the vlogan command. You enter the +incdir+ in the Verilog Compile field in the VCS Options dialog box, as shown below:



#### Example Verilog File:

```
`include "component.v"

module Top (input a, output x);

  ...

endmodule
```

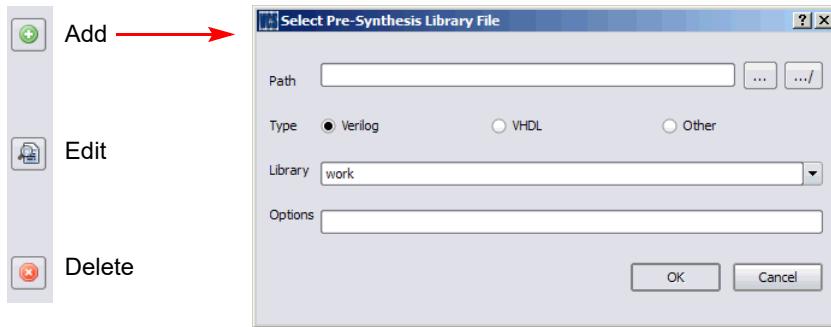
The syntax for the VCS commands must reflect the relative location of the Verilog files:

- If the Verilog files are in the same directory as the top.v file, specify:
- vlogan -work work Top.v +incdir+ ./

- If the Verilog files are in the a directory above the `top.v` file, specify:  
`vlogan -work work Top.v +incdir+ ../include1 +incdir+ ../include2`
- If the Verilog files are in directories below and above the `top.v` file, specify:  
`vlogan -work work Top.v +incdir+ ./include_dir1 +incdir../include_dir2`

#### 4. Specify the libraries and test bench files, if you are using them.

- To specify a library, click the green Add button, and specify the library in the dialog box that opens. Use the full path to the libraries. For pre-synthesis simulation, specifying libraries is optional.



- For post-synthesis and post-P&R synthesis, by default the dialog box displays the UNISIM and SIMPRIM libraries in the P&R tool path. You can add and delete libraries or edit them, using the buttons on the side. To restore the defaults, click the Verilog Defaults or VHDL Defaults button, according to the language you are using.
- If you have test bench files, choose the category Test Bench Files in the dialog box to specify them. Use the buttons on the side to add, delete, or edit the files.

#### 5. Specify the top-level module and run directory.

- Choose the category Top Level Module in the dialog box to specify the top-level module or modules for the simulation.
- If necessary, choose the category Run Directory near the bottom of the dialog box to edit the default run directory listed in the field. The default location is in the implementation results directory.

#### 6. Generate the VCS script.

- To view the script before generating it, click the View Script button on the top right of the dialog box. A window opens with the specified VCS commands and options.
  - To generate the VCS script, click Save As, or run VCS by clicking the Run button in the upper right. The tool generates the XML script in the directory specified.
7. To run VCS from the synthesis tool interface, do the following:
- If you do not already have it open, open the Run VCS Simulator dialog box by clicking the  icon.
  - To use an existing script, click the Load From button on the lower right and select the script in the dialog box that opens. Then click Run in the Run VCS Simulator dialog box.
  - If you do not have an existing script, specify the VCS options, as described in the previous five steps. Click Run.

The tool invokes VCS from the synthesis interface, using the commands in the script.

## Limitations

If Verilog include paths have been added to your project file, these paths are not automatically added to the VCS script. Add the Verilog include paths manually by using one of the following workarounds:

- From the Run VCS Simulator dialog box, add `+includer+includePath` in the Verilog Compile options field.
- Modify the VCS script file, adding the `+includer+includePath` to all or any relevant vlogan commands.



# Index

---

## Symbols

.adc file 349  
.fdc file 115

## A

adc constraints 349  
adc file  
    creating 349  
    object names 353  
adc file, using 347  
Alt key  
    column editing 38  
    mapping 307  
analysis design constraint file (.adc) 349  
analysis design constraints  
    design scenarios 348  
analysis design constraints (adc) 347  
analysis design constraints (adc), using  
    with fdc 350  
Analyst view  
    traversing hierarchy 221  
archive utility  
    using 100  
archiving projects 100  
area, optimizing 381  
asterisk wildcard  
    Find command 237, 295  
attributes  
    adding 89  
    adding in SCOPE 92  
    adding in Verilog 91  
    adding in VHDL 90  
    effects of retiming 392  
    for FSMs 372  
    handling properties 96  
    pipelining 386  
    syn\_hier (on compile points) 446

VHDL package 90  
auto constraints, using 354  
AutoConstraint\_design\_name.sdc 357

## B

B.E.S.T 311  
backslash  
    escaping dot wildcard in Find  
        command 237, 295  
batch mode 18  
    using find and expand 147  
Behavior Extracting Synthesis  
    Technology. *See* B.E.S.T  
black boxes 360  
    adding constraints 365  
    adding constraints in SCOPE 367  
    adding constraints in Verilog 367  
    adding constraints in VHDL 366  
    gated clock attributes 476  
    instantiating in Verilog 360  
    instantiating in VHDL 362  
    passing VHDL boolean generics 49  
    passing VHDL integer generics 50  
    pin attributes 368  
    timing constraints 365

black\_box compile point 425

bookmarks  
    in source files 38  
    using in log files 169

bottom-up design flow  
    compile point advantages 418

browsers 280

buffering  
    controlling 404

## C

c\_diff command, examples 153  
c\_intersect command, examples 153

- c\_list command
  - different from c\_print 154
  - example 156
  - using 156
- c\_print command
  - different from c\_list 155
  - using 156
- c\_symdiff command, examples 153
- c\_union command, examples 152
- case sensitivity
  - Find command (Tcl) 139
- clock constraints
  - setting 123
- clock conversion report
  - accessing 465
  - analyzing 466
- clock groups
  - effect on false path constraints 137
- clock trees 339
- clocks
  - implicit false path 137
- collections
  - adding objects 151
  - concatenating 151
  - constraints 150
  - copying 156
  - creating from common objects 151
  - creating from other collections 150
  - creating in SCOPE 148
  - creating in Tcl 151
  - crossprobing objects 149
  - definition 147
  - diffing 151
  - highlighting in HDL Analyst views 154
  - listing objects 156
  - listing objects and properties 155
  - listing objects in a file 155
  - listing objects in columnar format 155
  - listing objects with c\_list 154
  - special characters 154
  - Tcl window and SCOPE
    - comparison 147
  - using Tcl expand command 144
  - using Tcl find command 143
  - viewing 154
- column editing 38
- comments
  - source files 38
- compile point types
  - black\_box 425
  - hard 423
  - locked 424
- compile points
  - advantages 418
  - analyzing results 446
  - automatic timing budgeting 429
  - child 421
  - constraint files 426
  - constraints for forward-annotation 435
  - constraints, internal 435
  - creating constraint file 444
  - defined 418
  - defining in constraint files 441
  - feature summary 425
  - incremental synthesis 449
  - manual compile point flow 436
  - nested 420
  - optimization 433
  - order of synthesis 433
  - parent 421
  - preserving with syn\_hier 446
  - resynthesis 434
  - setting constraints 444
  - setting type 442
  - syn\_hier 446
  - synthesis process 432
  - synthesizing 436
  - types 422
  - using syn\_allowed\_resources attribute 446
- compile-point synthesis
  - interface logic models 428
- compile-point synthesis flow
  - defining compile points 441
  - setting constraints 444
- compiler directives 34
  - using 34
- compiler directives (Verilog)
  - specifying 85
- constants
  - extracting from VHDL source code 87
- constraint files 51
  - applying to a collection 150
  - black box 365
  - compile point 426, 435
  - editing 130

- 
- effects of retiming 392
  - opening 115
  - options 79
  - setting for compile points 444
  - specifying through points 133
  - when to use 51
  - constraints
    - using FDC template command 118
  - context
    - for object in filtered view 314
  - context help editor 34
    - SystemVerilog 34
  - continue on error 78
  - create\_fdc\_template
    - using 118
  - critical paths
    - delay 340
    - flat view 340
    - hierarchical view 340
    - slack time 340
    - using -route 383
    - viewing 339
  - crossprobing 303
    - and retiming 392
  - collection objects 149
  - filtering text objects for 308
  - from FSM viewer 310
  - from log file 169
  - from message viewer 183
  - from text files 306
  - Hierarchy Browser 238, 303
  - importance of encoding style 310
  - new HDL Analyst views 239
  - paths 307
    - RTL view 304
    - schematic views 238
    - Technology view 304
    - Text Editor view 304
    - text file example 307
    - to FSM Viewer 310
    - to place-and-route file 276
    - Verilog file 304
    - VHDL file 304
    - within RTL and Technology views 303
  - current level
    - expanding logic from net 319
    - expanding logic from pin 319
    - searching current level and below 291
  - custom folders
    - creating 65
    - hierarchy management 65
- D**
- default enum encoding 87
  - define\_attribute 95
  - design guidelines 380
  - design hierarchy
    - viewing 245, 312
  - design size
    - amount displayed on a sheet 276
  - design views
    - moving between views 217, 275
  - device options
    - See also* implementation options
  - directives
    - adding 89
    - adding in Verilog 91
    - adding in VHDL 90
    - black box 365, 366
    - for FSMs 372
    - handling properties 96
    - specifying for Verilog compiler 85
    - syn\_state\_machine 410
    - syn\_tco 366
      - adding black box constraints 365
    - syn\_tpd 366
      - adding black box constraints 365
    - syn\_tsu 366
      - adding black box constraints 365
  - directory
    - examples delivered with synthesis tool 21
  - dissolving instances for flattening hierarchy 325
  - dot wildcard
    - Find command 237, 295
  - drivers
    - preserving duplicates with syn\_keep 396
    - selecting 321
- E**
- Editing window 37

editor  
  compiler directives 34  
editor view  
  context help 34  
emacs text editor 42  
encoding styles  
  and crossprobing 310  
  default VHDL 87  
  FSM Compiler 409  
error messages  
  gated clock report 469  
errors  
  continuing 78  
  definition 37  
  filtering 183  
  sorting 183  
  source files 36  
  Verilog 36  
  VHDL 36  
examples delivered with synthesis tool,  
  directory 21  
expand  
  batch mode 147  
Expand command  
  connection logic 321  
  pin and net logic 251, 318  
  using 318  
expand command (Tcl). *See* Tcl expand  
  command  
Expand Inwards command  
  using 251, 318  
Expand Paths command  
  different from Isolate Paths 322  
Expand to Register/Port command  
  using 318  
expanding  
  connections 321  
  pin and net logic 251, 318

**F**

false paths  
  I/O paths 137  
  impact of clock group assignments 137  
  ports 137  
  registers 137  
  setting constraints 137

fanouts  
  buffering vs replication 404  
  hard limits 403  
  soft global limit 402  
  soft module-level limit 403  
  using syn\_keep for replication 397  
  using syn\_maxfan 402

files  
  .fdc 115  
  .prf file 186  
  .prj 23  
  filtered messages 187  
  fsm.info 410  
  log 166  
  message filter (prf) 186  
  output 495  
  project (.prj) 23  
  rom.info 283  
  searching 97  
  statemachine.info 332

Filter Schematic command,  
  using 249, 316

Filter Schematic icon, using 250, 316

filtering 249, 315  
  advantages over flattening 249, 315  
  using to restrict search 291

Find command  
  291  
  browsing with 289  
  hierarchical search 292  
  long names 290  
  message viewer 183  
  reading long names 294  
  search scope, effect of 295  
  search scope, setting 292  
  searching the mapped database 293  
  searching the output netlist 300  
  setting limit for results 293  
  using in RTL and Technology views 291  
  using wildcards 237, 295  
  wildcard examples 297

find command  
  nuances and differences 298

Find command (Tcl)  
  *See also* Tcl find command

Fix Gated Clocks option. *See* gated  
  clocks

Flatten Current Schematic command

- 
- transparent instances [324](#)
  - using [324](#)
  - Flatten Schematic command
    - using [324](#)
  - flattening [259, 322](#)
    - See also* dissolving
    - compared to filtering [249, 315](#)
    - dissolving instances [259, 325](#)
    - hidden instances [325](#)
    - transparent instances [324](#)
    - using syn\_hier [400](#)
    - using syn\_netlist\_hierarchy [400](#)
  - forward-annotation
    - compile point constraints [435](#)
  - FPGA Design Constraints Editor
    - using TCL View [126](#)
  - frequency
    - setting global [78](#)
  - from constraints
    - specifying [132](#)
  - FSM Compiler
    - advantages [408](#)
    - enabling [408](#)
  - FSM encoding
    - user-defined [373](#)
    - using syn\_enum\_encoding [373](#)
  - FSM view
    - crossprobing from source file [306](#)
  - FSM Viewer [329](#)
    - crossprobing [310](#)
  - fsm.info file [410](#)
  - FSMs
    - See also* FSM Compiler, FSM Explorer
    - attributes and directives [372](#)
    - defining in Verilog [370](#)
    - defining in VHDL [371](#)
    - definition [370](#)
    - optimizing with FSM Compiler [407](#)
    - properties [332](#)
    - state encodings [331](#)
    - transition diagram [329](#)
    - viewing [329](#)
- ## G
- gated clocks
    - attributes for black boxes [476](#)
- ## H
- conversion example [458](#)
  - conversion requirements [457](#)
  - error messages [469](#)
  - examples [455](#)
  - restrictions [480](#)
  - gated-clock conversion
    - excluding elements [472](#)
  - generated-clock conversion [484](#)
  - generics
    - extracting from VHDL source code [87](#)
    - passing boolean [49](#)
    - passing integer [50](#)
  - global optimization options [76](#)
- ## HDL Analyst
- See also* RTL view, Technology view
  - critical paths [339](#)
  - crossprobing [238, 303](#)
  - filtering schematics [249, 315](#)
  - Push/Pop mode [283, 286](#)
  - traversing hierarchy with mouse strokes [281](#)
  - traversing hierarchy with Push/Pop mode [224, 282](#)
  - using [245, 311](#)
- ## HDL Analyst tool
- deselecting objects [211, 273](#)
  - selecting/deselecting objects [210, 273](#)
- ## HDL Analyst views
- highlighting collections [154](#)
- ## HDL views, annotating timing information
- [337](#)
- ## hidden instances
- consequences of saving [313](#)
  - flattening [325](#)
  - restricting search by hiding [291](#)
  - specifying [313](#)
  - status in other views [313](#)
- ## hierarchical design
- expanding logic from nets [319](#)
  - expanding logic from pins [251, 318](#)
- ## hierarchical instances
- dissolving [259, 325](#)
  - hiding. *See* hidden instances, Hide Instances command
  - multiple sheets for internal logic [314](#)

- pin name display 317  
viewing internal logic 247, 313
- hierarchical objects  
pushing into with mouse stroke 224, 282  
traversing with Push/Pop mode 224, 282
- hierarchical search 291
- hierarchy  
flattening 259, 322  
traversing 221, 280
- hierarchy browser  
clock trees 339  
controlling display 276  
crossprobing from 238, 303  
defined 221, 280  
finding objects 227, 288  
traversing hierarchy 280
- hierarchy management (custom folders) 65
- 
- I**
- I/O insertion 406
- I/O pads  
specifying I/O standards 126
- I/O paths  
false path constraint 137
- I/O standards 126
- I/Os  
auto-constraining 355  
constraining 125  
preserving 407  
Verilog black boxes 360  
VHDL black boxes 362
- implementation options 74  
device 74  
global frequency 78  
global optimization 76  
part selection 74  
specifying results 80
- implementations  
copying 73  
deleting 73  
multiple. *See* multiple implementations.  
overwriting 73  
renaming 73
- incremental synthesis  
compile points 449
- input constraints, setting 124
- Instance Hierarchy tab 229
- instances  
preserving with syn\_noprune 396  
properties 202, 267  
properties of pins 268
- ILM *See* interface logic models
- interface logic models 428
- interface timing 429
- Isolate Paths command  
different from Expand Paths 322
- 
- J**
- job management  
up-to-date checking 161
- 
- K**
- keywords  
completing words in Text Editor 38
- 
- L**
- library extensions 43
- license  
specifying in batch mode 18
- log file  
remote access 173
- log files  
checking information 166  
pipelining description 387  
retiming report 392  
setting default display 166  
state machine descriptions 409  
viewing 166
- logic  
expanding between objects 321  
expanding from net 252, 319  
expanding from pin 251, 318
- logic preservation  
syn\_hier 401  
syn\_keep for nets 396  
syn\_keep for registers 396  
syn\_noprune 396

- 
- s**
    - syn\_preserve 396
    - logical folders
      - creating 65
  - M**
    - manual compile points
      - flow 436
    - memory usage
      - maximizing with HDL Analyst 327
    - Message viewer
      - filtering messages 184
      - keyboard shortcuts 183
      - saving filter expressions 186
      - searching 183
      - using 182
      - using the F3 key to search forward 183
      - using the Shift-F3 key to search backward 183
    - messagefilter.txt file 193
    - messages
      - demoting 189
      - filtering 184
      - promoting 189
      - saving filter information from command line 186
      - saving filter information from GUI 186
      - severity levels 191
      - suppressing 189
      - writing messages to file 187
    - mixed designs
      - troubleshooting 49
    - mixed language files 46
    - mouse strokes
      - pushing/popping objects 223, 281
    - multicycle paths
      - setting constraints 123
    - multiple implementations 72
      - running from project 72
    - multisheet schematics 274
      - for nested internal logic 314
      - searching just one sheet 291
      - transparent instances 274
  - N**
    - name spaces
      - output netlist 300
  - P**
    - technology view 293
    - navigating among design views 217, 275
    - netlists for different vendors 495
    - nets
      - expanding logic from 252, 319
      - preserving for probing with syn\_probe 396
      - preserving with syn\_keep 396
      - properties 202, 267
      - selecting drivers 321
    - new Hierarchy Browser 228
    - New property 270
    - notes
      - filtering 183
      - sorting 183
    - notes, definition 37
  - O**
    - objects
      - finding on current sheet 291
      - flagging by property 268
      - selecting/deselecting 273
    - open\_design
      - with find and expand 147
    - optimization
      - for area 381
      - for timing 382
      - generated clock 484
      - logic preservation. See logic preservation.
      - preserving hierarchy 401
      - preserving objects 396
      - tips for 380
    - OR 135
    - orig\_inst\_of property 271
    - output constraints, setting 124
    - output files 495
      - specifying 80
    - output netlists
      - finding objects 300
    - overutilization 179

- pad types
  - industry standards [126](#)
- parameter passing [50](#)
  - boolean generics [49](#)
- parameters
  - extracting from Verilog source code [84](#)
- part selection options [74](#)
- path constraints
  - false paths [137](#)
- pathnames
  - using wildcards for long names (Find) [294](#)
- paths
  - crossprobing [307](#)
  - tracing between objects [321](#)
  - tracing from net [252, 319](#)
  - tracing from pin [251, 318](#)
- pattern matching
  - Find command (Tcl) [139](#)
- pattern searching [97](#)
- PDF
  - cutting from [38](#)
- pin names, displaying [317](#)
- pins
  - expanding logic from [251, 318](#)
  - properties [202, 267](#)
- pipelining
  - adding attribute [386](#)
  - definition [384](#)
  - whole design [385](#)
- ports
  - false path constraint [137](#)
  - properties [202, 267](#)
- POS interface
  - using [133](#)
- post-synthesis constraints with adc [348](#)
- preferences
  - crossprobing to place-and-route file [276](#)
  - displaying Hierarchy Browser [276](#)
  - displaying labels [277](#)
  - RTL and Technology views [276](#)
  - sheet size (UI) [276](#)
- primitives
  - pin name display [317](#)
- pushing into with mouse stroke [224, 282](#)
- viewing internal hierarchy [312](#)
- prj file [23](#)
- probes
  - adding in source code [413](#)
  - definition [413](#)
  - retiming [394](#)
- Product of Sums interface. *See* POS interface
- project command
  - archiving projects [100](#)
  - copying projects [106](#)
  - unarchiving projects [103](#)
- project file hierarchy [65](#)
- project files
  - adding files [60](#)
  - adding source files [56](#)
  - creating [56](#)
  - definition [56](#)
  - deleting files from [60](#)
  - opening [59](#)
  - replacing files in [60](#)
  - updating include paths [64](#)
  - VHDL library [58](#)
- project files (.prj) [23](#)
- project status report
  - remote access [173](#)
- projects
  - archiving [100](#)
  - copying [106](#)
  - restoring archives [103](#)
- properties
  - displaying with tooltip [202, 267](#)
  - finding objects with Tcl find -filter [140](#)
  - orig\_inst\_of [271](#)
  - reporting for collections [155](#)
  - viewing for individual objects [202, 268](#)
- Push/Pop mode
  - HDL Analyst [281](#)
  - keyboard shortcut [283](#)
  - using [223, 224, 281, 282](#)
- Q**
- question mark wildcard, Find command [237, 295](#)

## R

RAMs  
  initializing 374

register balancing. *See* retiming

registers  
  false path constraint 137

Registers panel  
  using SCOPE 122

remote access  
  status reports 173

replication  
  controlling 404

reports  
  gated clock conversion 466

resource sharing  
  optimization technique 381  
  overriding option with syn\_sharing 406  
  using 406

resource usage 178

resource utilization. *See* resource usage

resynthesis  
  compile points 434  
  forcing with Resynthesize All 434  
  forcing with Update Compile Point Timing Data 434

retiming  
  effect on attributes and constraints 392  
  example 391  
  overview 388  
  probes 394  
  report 392  
  simulation behavior 394

rom.info file 283

ROMs  
  viewing data table 283

RTL view  
  *See also* HDL Analyst  
  analyzing clock trees 339  
  crossprobing collection objects 149  
  crossprobing description 303  
  crossprobing from 304  
  crossprobing from Text Editor 306  
  defined 266  
  description 265  
  filtering 249, 315  
  finding objects with Find 291

finding objects with Hierarchy Browser 227, 288  
flattening hierarchy 259, 322  
highlighting collections 154  
opening 199, 267  
selecting/deselecting objects 273  
setting preferences 276  
state machine implementation 410  
traversing hierarchy 280

## S

schematic view  
  setting preferences (New Analyst) 218

schematics  
  multisheet. *See* multisheet schematics  
  page size 276  
  selecting/deselecting objects 210, 273

SCOPE  
  adding attributes 92  
  adding probe insertion attribute 415  
  case sensitivity for Verilog designs 139  
  collections compared to Tcl script window 147  
  creating compile-point constraint file 444  
  defining compile points 439  
  drag and drop 130  
  editing operations 131  
  I/O pad type 126  
  multicycle paths 136  
  pipelining attribute 385  
  Registers panel 122  
  setting compile point constraints 444  
  setting constraints (FDC) 114  
  state machine attributes 372

search  
  browsing objects with the Find command 289  
  browsing with the Hierarchy Browser 227, 288  
  finding objects on current sheet 291  
  setting limit for results 293  
  setting scope 292  
  using the Find command in HDL Analyst views 291

search in Analyst  
  browsing objects with the Find command 233

*See also* search

set command  
collections 156

set\_option command 76

sheet connectors  
navigating with 275

sheet size  
setting number of objects 276

Shift-F3 key  
Message Viewer 183

Show Cell Interior option 312

Show Context command  
different from Expand 314  
using 314

simulation, effect of retiming 394

slack 342  
setting margins 339

slack time display 336

Slow property 269

source code  
adding pipelining attribute 386  
commenting with synthesis on/off 88  
crossprobing from Tcl window 309  
defining FSMs 370  
fixing errors 39  
opening automatically to  
crossprobe 305  
optimizing 380  
when to use for constraints 51

source files  
*See also* Verilog, VHDL.  
adding comments 38  
adding files 56  
checking 36  
column editing 38  
copying examples from PDF 38  
creating 32  
crossprobing 306  
editing 37  
editing operations 37  
mixed language 46  
specifying default encoding style 87  
specifying top level file for mixed  
language projects 47  
specifying top-level file 87  
state machine attributes 372  
using bookmarks 38

special characters

Tcl collections 154

STA 344

STA, generating custom timing  
reports 344

STA, using analysis design constraints  
(adc) 347

stand-alone timing analyst. *See* STA

starting Synplify Pro 18

state machines  
*See also* FSM Compiler, FSM Explorer,  
FSM viewer, FSMs.

attributes 372  
descriptions in log file 409  
implementation 410  
parameter and 'define comparison 371

statemachine.info file 332

syn\_allow\_retimig  
compile points 446  
using for retiming 389

syn\_allowed\_resources  
compile points 446

syn\_encoding attribute 372

syn\_enum\_encoding directive  
FSM encoding 373

syn\_hier attribute  
controlling flattening 400  
preserving hierarchy 401  
using with compile points 446

syn\_isclock  
black box clock pins 369

syn\_keep  
replicating redundant logic 397

syn\_keep attribute  
preserving nets 396  
preserving shared registers 396

syn\_keep directive  
effect on buffering 404

syn\_maxfan attribute  
setting fanout limits 402

syn\_noprune directive  
preserving instances 396

syn\_pipeline attribute 386

syn\_preserve  
effect on buffering 404  
preserving power-on for retiming 390

---

**T**  
 syn\_preserve directive  
     preserving FSMs from optimization [372](#)  
     preserving logic [396](#)  
 syn\_probe attribute [413](#)  
     inserting probes [413](#)  
     preserving nets [396](#)  
 syn\_replicate attribute  
     using buffering [404](#)  
 syn\_sharing directive  
     overriding default [406](#)  
 syn\_state\_machine directive  
     using with value=0 [410](#)  
 syn\_tco attribute  
     adding in SCOPE [367](#)  
 syn\_tco directive [366](#)  
     adding black box constraints [365](#)  
 syn\_tpd attribute  
     adding in SCOPE [367](#)  
 syn\_tpd directive [366](#)  
     adding black box constraints [365](#)  
 syn\_tsu attribute  
     adding in SCOPE [367](#)  
 syn\_tsu directive [366](#)  
     adding black box constraints [365](#)  
 syn\_useioff  
     preventing flops from moving during  
         retiming [390](#)  
 synhooks  
     automating message filtering [187](#)  
 Synopsys  
     FPGA product family [14](#)  
 synplify\_pro command-line  
     command [18](#)  
 SYNPLIFY\_REMOTE\_REPORT\_LOCATION  
     [174](#)  
 syntax  
     checking source files [36](#)  
 syntax check [36](#)  
 synthesis check [36](#)  
 synthesis\_on/off  
     using [88](#)  
 SystemVerilog keywords  
     context help [34](#)  
  
 ta file [344](#)  
 Tcl expand  
     using [138](#)  
 Tcl expand command  
     crossprobing objects [149](#)  
     usage tips [144](#)  
     using in SCOPE [149](#)  
 Tcl files  
     guidelines [52](#)  
     naming conventions [52](#)  
     wildcards [53](#)  
 Tcl find  
     batch mode [147](#)  
     filtering results by property [140](#)  
     search patterns [138](#)  
     using [138](#)  
 Tcl find command  
     annotating properties [140](#)  
     case sensitivity [139](#)  
     crossprobing objects [149](#)  
     database differences [149](#)  
     pattern matching [139](#)  
     Tcl window vs SCOPE [148](#)  
     usage tips [143](#)  
     useful -filter examples [143](#)  
     using in SCOPE [149](#)  
 Tcl Script window  
     crossprobing [309](#)  
     message viewer [182](#)  
 Tcl script window  
     collections compared to SCOPE [147](#)  
 TCL View [126](#)  
 Technology view  
     *See also* HDL Analyst  
     critical paths [339](#)  
     crossprobing [303, 304](#)  
     crossprobing collection objects [149](#)  
     crossprobing from source file [306](#)  
     filtering [249, 315](#)  
     finding objects [293](#)  
     finding objects with Find [291](#)  
     finding objects with Hierarchy  
         Browser [227, 288](#)  
     flattening hierarchy [259, 322](#)  
     general description [265](#)  
     highlighting collections [154](#)

- opening [267](#)
- selecting/deselecting objects [273](#)
- setting preferences [276](#)
- state machine implementation in [410](#)
- traversing hierarchy [280](#)
- text editor**
  - built-in [37](#)
  - external [42](#)
  - using [37](#)
- Text Editor view
  - crossprobing [304](#)
- Text Editor window
  - colors [40](#)
  - crossprobing [40](#)
  - fonts [40](#)
- text files
  - crossprobing [306](#)
- The Synopsys FPGA Product Family [14](#)
- through constraints [133](#)
  - AND lists [134](#)
  - OR lists [133](#)
- time stamp, checking on files [61](#)
- timing analysis [336](#)
- timing analysis using STA [344](#)
- timing budgeting
  - compile points [429](#)
- timing exceptions, adding constraints after synthesis [348](#)
- timing exceptions, modifying with adc [348](#)
- timing failures [342](#)
- timing information commands [336](#)
- timing information in HDL views [337](#)
- timing information, critical paths [340](#)
- timing optimization [382](#)
- timing report, stand-alone [344](#)
- timing reports
  - specifying format options [82](#)
- timing reports, custom [344](#)
- tips**
  - memory usage [327](#)
- to constraints
  - specifying [133](#)
- top level
  - specifying [86](#)
- top-down design flow
  - compile point advantages [419](#)
- transparent instances
  - flattening [324](#)
  - lower-level logic on multiple sheets [274](#)

**U**

- up-to-date checking [161](#)
- copying job logs to log file [163](#)
- limitations [164](#)

**V**

- vendor-specific netlists [495](#)
- Verilog**
  - 'define statements [85](#)
  - adding attributes and directives [91](#)
  - adding probes [413](#)
  - black boxes [360](#)
  - black boxes, instantiating [360](#)
  - case sensitivity for Tcl Find command [139](#)
  - checking source files [36](#)
  - choosing a compiler [84](#)
  - creating source files [32](#)
  - crossprobing from HDL Analyst view [304](#)
  - defining FSMs [370](#)
  - defining state machines with parameter and 'define [371](#)
  - editing operations [37](#)
  - extracting parameters [84](#)
  - include paths, updating [64](#)
  - initializing RAMs [374](#)
  - mixed language files [46](#)
  - specifying compiler directives [85](#)
  - specifying top-level module [86](#)
  - using library extensions [43](#)
- Verilog 2001
  - setting global option from the Project view [84](#)
  - setting option per file [84](#)
- Verilog library files
  - using library extensions [43](#)
- Verilog model (.vmd) [428](#)
- VHDL**
  - adding attributes and directives [90](#)

---

- adding probes [413](#)
- black boxes [362](#)
- black boxes, instantiating [362](#)
- case sensitivity for Tcl Find command [139](#)
- checking source file [36](#)
- constants [87](#)
- creating source files [32](#)
- crossprobing from HDL Analyst view [304](#)
- defining FSMs [371](#)
- editing operations [37](#)
- extracting generics [87](#)
- global signals in mixed designs [49](#)
- initializing RAMs with variable declarations [377](#)
- initializing with signal declarations [375](#)
- mixed language files [46](#)
- specifying top-level entity [86](#)
- VHDL files
  - adding library [58](#)
  - adding third-party package library [58](#)
- vi text editor [42](#)

## W

- warning messages
  - definition [37](#)
- warnings
  - feedback muxes [383](#)
  - filtering [183](#)
  - sorting [183](#)
- Watch window [177](#)
  - moving [177, 182](#)
  - multiple implementations [73](#)
  - resizing [177, 182](#)
- wildcards
  - effect of search scope [295](#)
  - Find command (Tcl) [139](#)
  - message filter [185](#)
- wildcards (Find)
  - examples [297](#)
  - how they work [237, 295](#)

