

# **Synopsys FPGA Synthesis** **Synplify Pro** **Quick Start Guide**

---

June 2009

<http://solvnnet.synopsys.com>

**SYNOPSYS®**

---

## Disclaimer of Warranty

Synopsys, Inc. makes no representations or warranties, either expressed or implied, by or with respect to anything in this manual, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose of for any indirect, special or consequential damages.

## Copyright Notice

Copyright © 2009 Synopsys, Inc. All Rights Reserved.

Synopsys software products contain certain confidential information of Synopsys, Inc. Use of this copyright notice is precautionary and does not imply publication or disclosure. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the prior written permission of Synopsys, Inc. While every precaution has been taken in the preparation of this book, Synopsys, Inc. assumes no responsibility for errors or omissions. This publication and the features described herein are subject to change without notice.

## Trademarks

### Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, Design Compiler, DesignWare, Formality, HDL Analyst, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SiVL, SCOPE, Simply Better Results, SNUG, SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

### Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, Direct Silicon Access, Discovery, Eclipse, Encore, EPIC, Galaxy, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierar-

---

chical Optimization Technology, High-performance ASIC Prototyping System, HSIM, HSIM<sup>plus</sup>, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

### **Service Marks (SM)**

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license. ARM and AMBA are registered trademarks of ARM Limited. Saber is a registered trademark of SabreMark Limited Partnership and is used under license. All other product or company names may be trademarks of their respective owners.

## **Restricted Rights Legend**

Government Users: Use, reproduction, release, modification, or disclosure of this commercial computer software, or of any related documentation of any kind, is restricted in accordance with FAR 12.212 and DFARS 227.7202, and further restricted by the Synopsys Software License and Maintenance Agreement. Synopsys, Inc., Synplicity Business Group, 600 West California Avenue, Sunnyvale, CA 94086, U. S. A.

Printed in the U.S.A  
June 2009



# Contents

---

## Chapter 1: Quick Start Overview

## Chapter 2: Process Flow

Top-Down and Compile Point Design Flows ..... 13

## Chapter 3: Set up Design Information

Select a Target Device 16

Set Implementation Options ..... 17

- State Machine Implementation ..... 17
- Resource Sharing ..... 18
- Pipelining ..... 18
- Retiming ..... 19
- Formal Verification ..... 20

## Chapter 4: Set up Timing Information

Set Timing Constraints ..... 24

Define Compile Points ..... 25

Set Constraints (Compile Point Synthesis) ..... 26

- Set Top-level Constraints ..... 26
- Set Compile Point Constraints ..... 26

Run ..... 28

## Chapter 5: Analyze Results

View the Log File ..... 31

- RTL View ..... 32
- Technology View ..... 33
- Design Hierarchy Exploration Tools ..... 33
- Advanced Find Capabilities ..... 36
- Filtered and Flattened Views ..... 37
- Crossprobing Across Views ..... 41

Cross-tool Crossprobing . . . . .	42
Other Options . . . . .	43
Mouse Strokes . . . . .	44
Use FSM Viewer . . . . .	45
Other Tools to Validate Synthesis Results . . . . .	48
Use Formal Verification . . . . .	48
Use syn_probe Attribute . . . . .	49
Identify RTL debugger . . . . .	49
<b>Chapter 6: Specify Directives and Attributes</b>	
syn_maxfan . . . . .	52
syn_keep . . . . .	52
syn_ramstyle . . . . .	53
<b>Chapter 7: Basics of Timing Constraints</b>	
Example . . . . .	57
Input Ports . . . . .	62
Output Ports . . . . .	64
Setting Multicycle Path Constraints . . . . .	66
<b>Chapter 8: Analyze Timing Results</b>	
Timing Information Display . . . . .	71
Critical Path Views . . . . .	71
Generate a Technology View for the Most Critical Path . . . . .	72
Generate Critical Path View in the Timing Analyzer . . . . .	73
Generate Critical Path View from the Log File . . . . .	76
<b>Chapter 9: Refine Options to Improve Timing</b>	
Compare Synthesis Results with Place-and Route Results . . . . .	80
Add Route Delay Constraints . . . . .	81
Forward Annotate Incremental Results . . . . .	82
<b>Chapter 10: Additional Features and Topics</b>	
Output Netlist . . . . .	87

## CHAPTER 1

# Quick Start Overview

---

**What does the Synplify Pro software Offer?** — The Synplify Pro software consists of a fast, high-performance, sophisticated logic synthesis engine that utilizes proprietary technology called Behavior Extracting Synthesis Technology® (BEST™) to deliver highly efficient FPGA and CPLD designs. Starting with Verilog and VHDL hardware description language input files, the software generates an optimized netlist in the most popular CPLD and FPGA vendor formats.

**Who Will Find This Guide Useful?** — This guide provides the steps and options of a design flow and can be used by:

- Engineers who want to evaluate the software without actually running it. The guide includes many graphic examples that show the capabilities of the software.
- Engineers who want to get a quick start to run synthesis.
- Managers who want to understand the capabilities and features of the software before purchasing.

**How is the information organized?** — The document is organized as described below. You can click on the links in the columns to take you to the sections:

---

	Describes...	Page
<b>Process Flow</b>	The top-down and Compile Point Synthesis flows.	11
<b>Set up Design Information</b>	The steps used to set up your project for Synplify Pro synthesis.	15
Add Source Files		15
Select a Target Device		16
Set Implementation Options		17
<b>Set up Timing Information</b>	How to set general timing constraints and define compile points for the Compile Point Synthesis flow.	23
Set Timing Constraints		24
Define Compile Points		25
Set Constraints (Compile Point Synthesis)		26
Run	How to synthesize the design.	28
<b>Analyze Results</b>	How to view the synthesis results using the log file and some built-in tools like the HDL Analyst tool for graphic analysis and the FSM viewer for state machine implementations. Also describes how to formally verify your results with LEC.	29
View the Log File		31
Use the HDL Analyst® Tool		31
Use FSM Viewer		45
<b>Other Tools to Validate Synthesis Results</b>		48
Use Formal Verification		48
Use syn_probe Attribute		49
Identify RTL debugger		49
<b>Specify Directives and Attributes</b>	How to use attributes and directives to fine-tune the way the design is synthesized.	51
syn_maxfan		52
syn_keep		52
syn_ramstyle		53

---



---

	Describes...	Page
<b>Basics of Timing Constraints</b>	Basic timing concepts used in the Synplify Pro tool.	<b>55</b>
Specifying Timing Information		55
Clock Descriptions		56
Clock Groups		59
Rise and Fall Constraints		60
Input and Output Delays		61
Multicycle Paths		65
I/O Standard		67
<b>Analyze Timing Results</b>	How to analyze timing results and use options to improve timing.	<b>69</b>
Critical Path Report		69
Timing Information Display		71
Critical Path Analysis in a Technology View		71
<b>Refine Options to Improve Timing</b>		<b>79</b>
Refine Timing Results		79
Compare Synthesis Results with Place-and Route Results		80
<b>Forward Annotate Incremental Results</b>	How to forward annotate incremental results (updates) in a Compile Point Synthesis flow so that your complete design project is up-to-date.	<b>82</b>

---

	Describes...	Page
<a href="#">Additional Features and Topics: Synplify Premier Physical Synthesis</a>	Additional features and options for synthesis.	<a href="#">85</a>
<a href="#">Synthesis Output Files</a>		
<a href="#">Using Multiple Clock Domains</a>		
<a href="#">Using Scripts and Batch Mode</a>		
<a href="#">Using the Tcl Find Command for Setting Constraints</a>		
<a href="#">Running Place and Route</a>		
<a href="#">Using Identify with Synplify Pro</a>		

---

## CHAPTER 2

# Process Flow

---

The Synplify Pro software is designed to give you the best overall circuit performance with a minimal amount of effort.

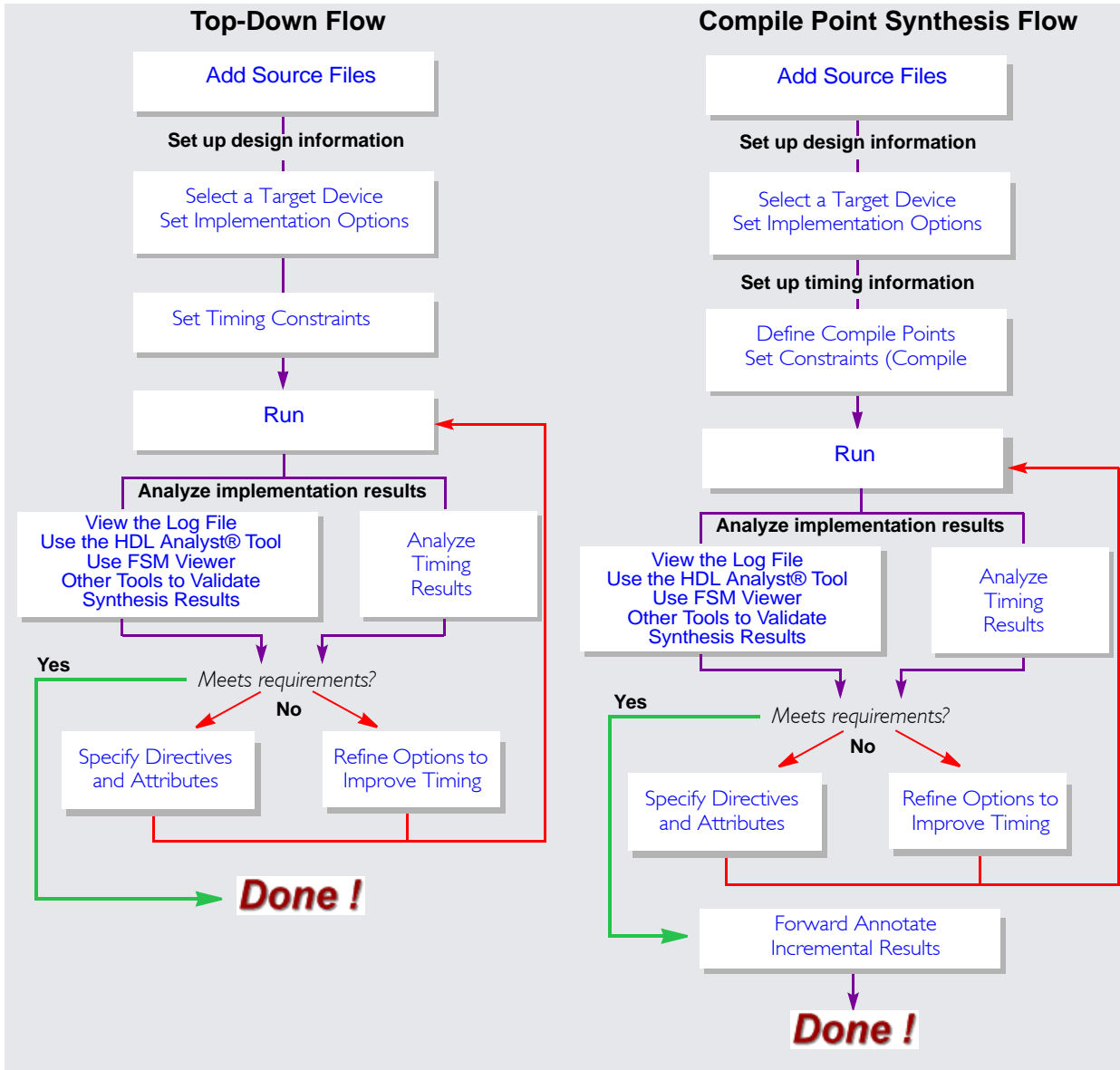
Topics include the following process flows:

- [Process Flow Diagram](#), on page 11
- [Top-Down and Compile Point Design Flows](#), on page 13

## Process Flow Diagram

The following figure shows you two Synplify Pro flows with simple steps to trade off between timing and area to help you reach your goals quickly.

The top-down flow is the traditional synthesis flow with a global approach to synthesis. With the Compile Point Synthesis flow, you can design incrementally and synthesize only what is necessary.



## Top-Down and Compile Point Design Flows

A Compile Point Synthesis flow differs from a traditional top-down flow in that it divides the design into parts that can be processed independently or synthesized incrementally, using a team design approach. Unlike other bottom-up solutions, it is highly automated and eliminates the need for time consuming and error prone scripts. Compile point synthesis is based on compile points, which are smaller synthesis units of the main design that are treated as individual blocks.

The Compile Point Synthesis flow is available for certain design families. Check the Device tab of the Implementation Options dialog box for applicable technology families. For Altera designs, you can use this flow with the Altera Quartus II Incremental Compilation methodology to preserve design implementation data so as to make incremental place and route updates. Similarly, you can use the Xilinx Incremental flows with the place-and-route tool for team-based design. See the Synplify Pro software documentation and the appropriate application notes for details.



## CHAPTER 3

# Set up Design Information

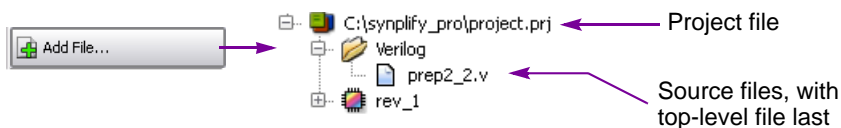
---

There are three basic tasks involved in setting up a design. Both the top-down and Compile Point Synthesis flows use the same design setup. Topics include:

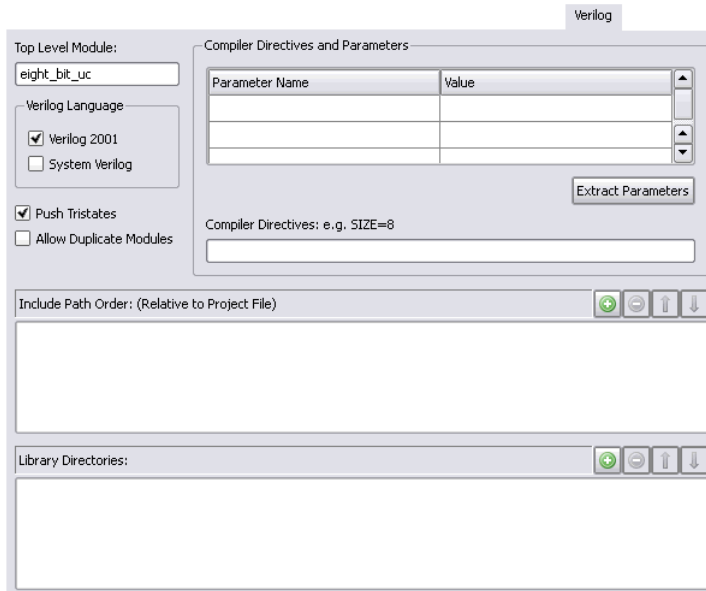
- [Add Source Files](#), on page 15
- [Select a Target Device](#), on page 16
- [Set Implementation Options](#), on page 17

## Add Source Files

After you have installed the software, set up the design project. Then, add source files.

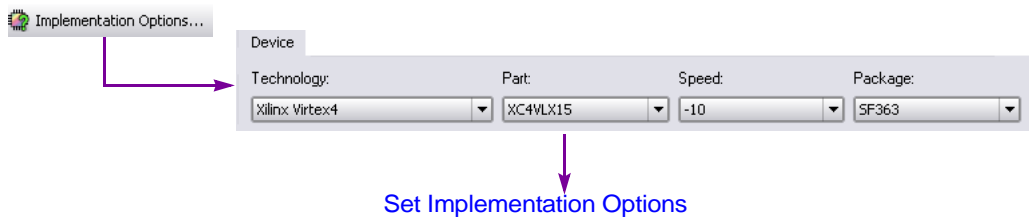


Source files need to be ordered such that the top-level file is last in the source file list. If you have mixed language source files (a combination of Verilog and VHDL files), specify the top-level file using Implementation Options ->Verilog or VHDL tab, as shown below.



## Select a Target Device

Choose the technology family using the Device panel of the Implementation Options dialog box. Select the part, package, and speed grade, as applicable. Remember that the speed grade you choose has a direct impact on timing estimates. For the Compile Point Synthesis flow, you must select an appropriate technology family.





# Set Implementation Options

Set options for the synthesis run from the Options tab of the Implementation Options dialog box. Some of the global options that influence the tradeoff between speed and area include:

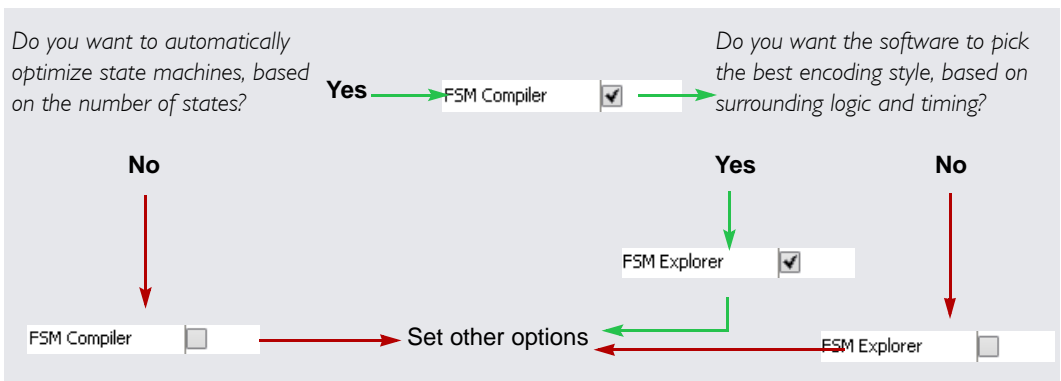
- [State Machine Implementation](#)
- [Resource Sharing](#)
- [Pipelining](#)
- [Retiming](#)

Some of these options can also be set on a per instance basis.

[Formal Verification](#) is a global option that provides a netlist compatible with the Cadence Conformal™ Equivalence Checker tool for design verification.

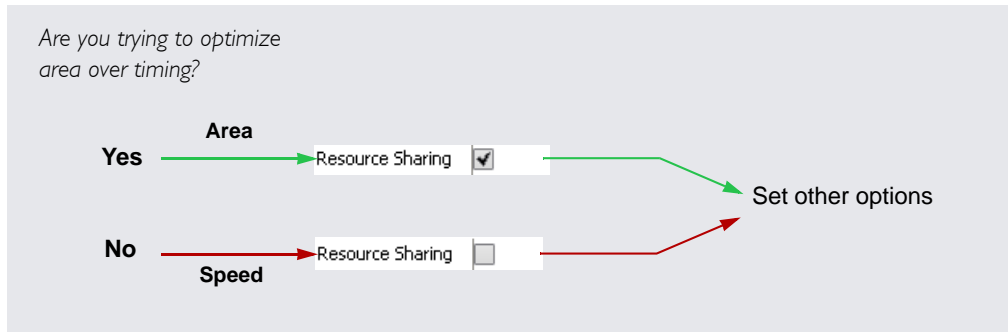
## State Machine Implementation

Use the FSM Compiler and the FSM Explorer to automatically select encoding styles that determine how the state machines are implemented. The encoding style affects timing estimates. You also can define state machine implementations for individual instances using attributes such as `syn_encoding`. You enable these options by selecting the FSM Compiler and FSM Explorer check boxes on the left side of the Project view.



## Resource Sharing

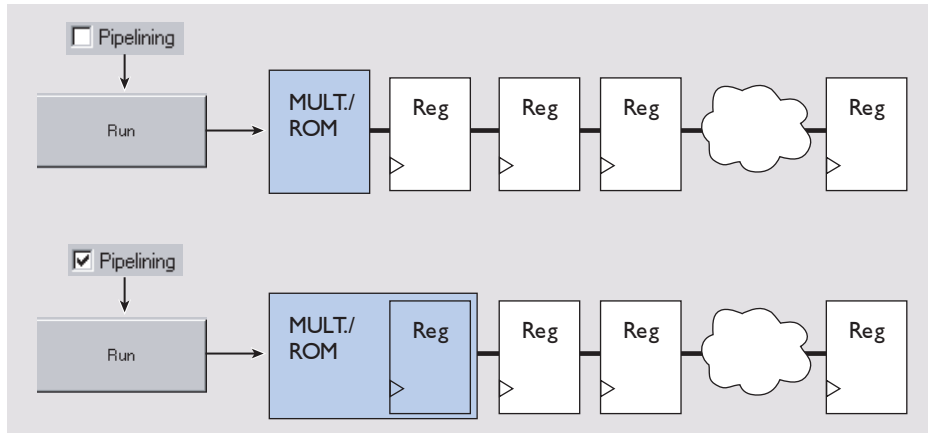
Resource sharing is another option that influences the tradeoff between speed and area. Turn on this option when you want to optimize area. You enable this option by selecting the Resource Sharing check box on the left side of the Project view.



## Pipelining

Pipelining is available for certain device families only. Check the Device or Options tab of the Implementation Options dialog box for applicable technology families. You can either set the options globally or on individual registers.

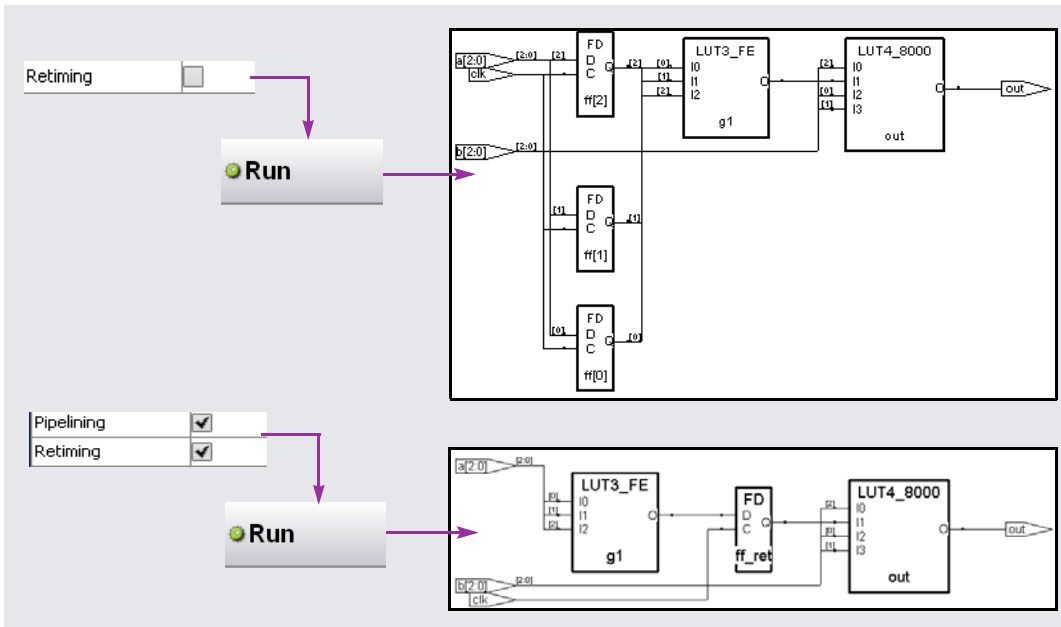
Pipelining is the process of moving adjacent registers into multipliers or ROMs so as to increase throughput and ensure faster circuit performance. It does not add new register stages. The following figure shows you how to set it globally.



## Retiming

Retiming is available for certain device families only and is a technique related to pipelining. Retiming improves the timing performance of sequential circuits by automatically moving registers (register balancing) across combinatorial gates or LUTs. This process improves timing while ensuring identical 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 algorithm retimes only edge-triggered registers. It does not retime level-sensitive latches.

When you turn on the retiming option, pipelining is turned on automatically.



## Formal Verification

This option is available for certain device technologies, and lets you use the Cadence Conformal Equivalence Checker tool for formal design verification. In verification mode, the software generates files that are used by Conformal Equivalence Checker to verify equivalence between the RTL code and the post-synthesis netlist.

Device

Technology:  Part:  Package:  Speed:

Device Mapping Options

Option	Value
Annotated Properties for Analyst	<input checked="" type="checkbox"/>
Fanout Guide	10000
Disable I/O Insertion	<input type="checkbox"/>
Update Compile Point Timing Data	<input type="checkbox"/>
Verification Mode	<input type="checkbox"/>
Retiming	<input type="checkbox"/>
Disable Sequential Optimizations	<input type="checkbox"/>
Fix Gated Clocks	3
Fix Generated Clocks	3
Use Xilinx Xflow	<input type="checkbox"/>

Option Description



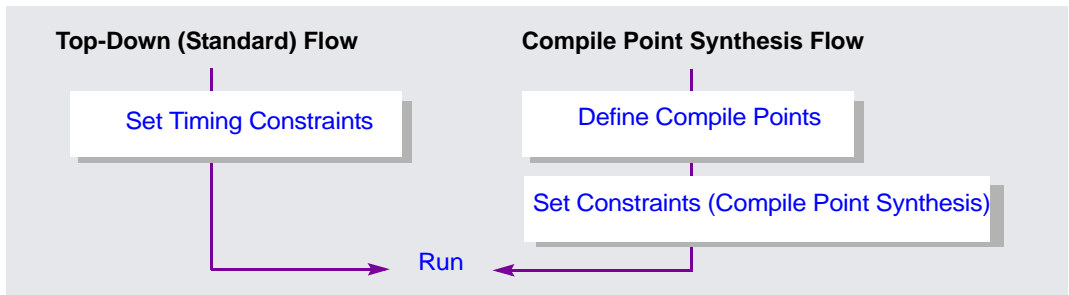
## CHAPTER 4

# Set up Timing Information

---

Synplify Pro synthesis is timing-driven. Consequently, the more precise and complete the information you supply, the more accurate the timing estimates, and the closer the synthesized implementation is to your design goals. The Synplify Pro tool begins optimizing for area once your timing performance constraints are set.

Define timing goals through constraints. Constraint setup steps differ for the two flows:



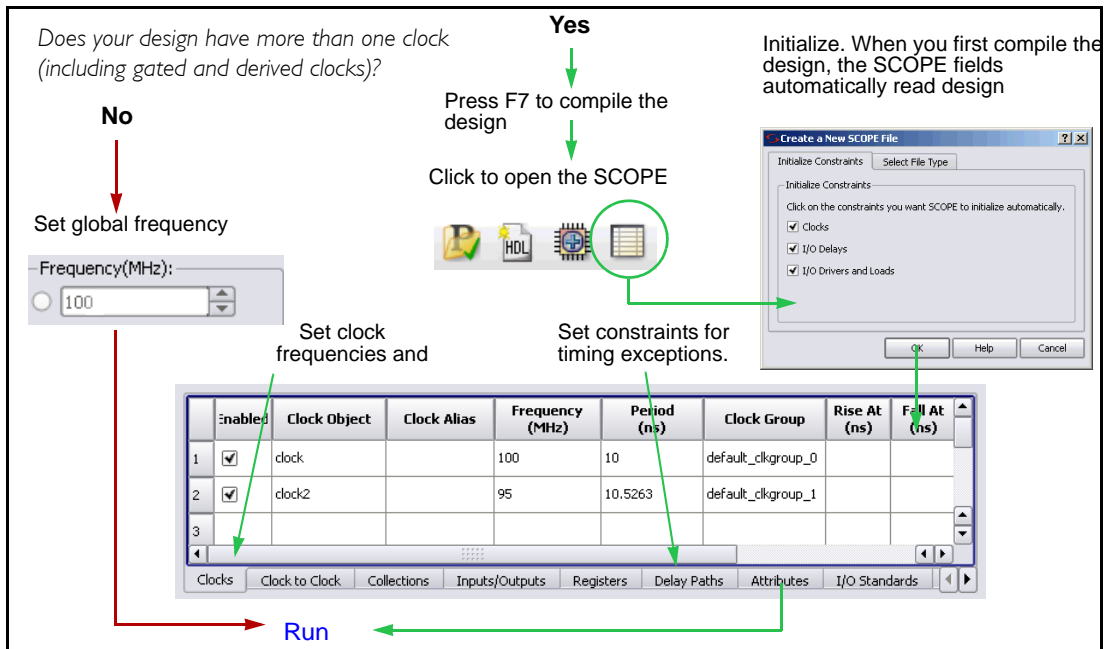
Refer to the following topics:

- [Set Timing Constraints](#), on page 24
- [Set Constraints \(Compile Point Synthesis\)](#), on page 26
- [Run](#), on page 28

# Set Timing Constraints

As a minimum, you must set a global clock frequency, which can be done in the Project view or through the Constraints tab of the Implementation Options dialog box. However, using a global frequency alone is not recommended because this option sets all clocks to the same value and assigns them to the same clock group. Synthesis assumes that all clocks are related and calculates timing for every path between the clocks.

When using multiple-clock designs, you should specify clock frequencies for each clock. Also, define timing exceptions like false path and multi-cycle path constraints. For defining timing constraints, use the SCOPE<sup>®</sup> interface which makes it easy to enter constraint definitions.





# Define Compile Points

Compile points are design modules that act as relatively independent synthesis units: they have their own constraint files and are optimized individually. They are resynthesized only as needed, based on an analysis of design dependencies and the nature of design changes. The synthesis process for compile points is called the Compile Point Synthesis flow. In this incremental flow, compile points are defined through the SCOPE interface.

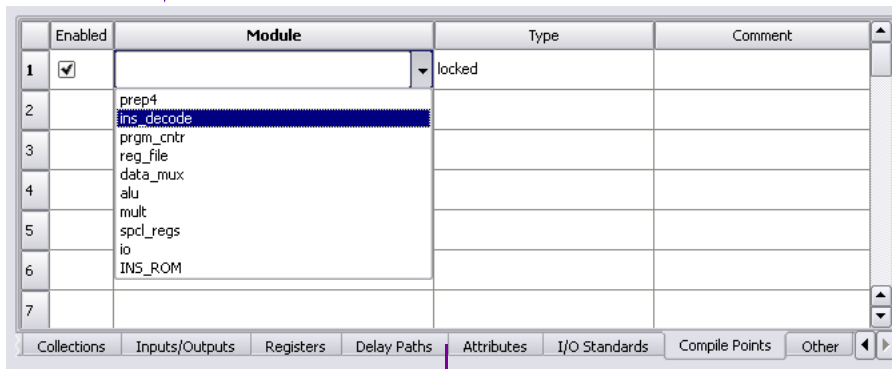
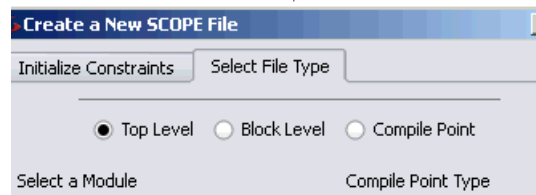
Press F7 to compile the design. With the compiled design, design information is automatically filled in the SCOPE fields

Click to open the SCOPE interface



Click the Select File Type tab, select Top Level, and click OK to initialize.

Click on the Compile Points tab, and select the compile point modules. Save the constraint file, and add the file to the project.



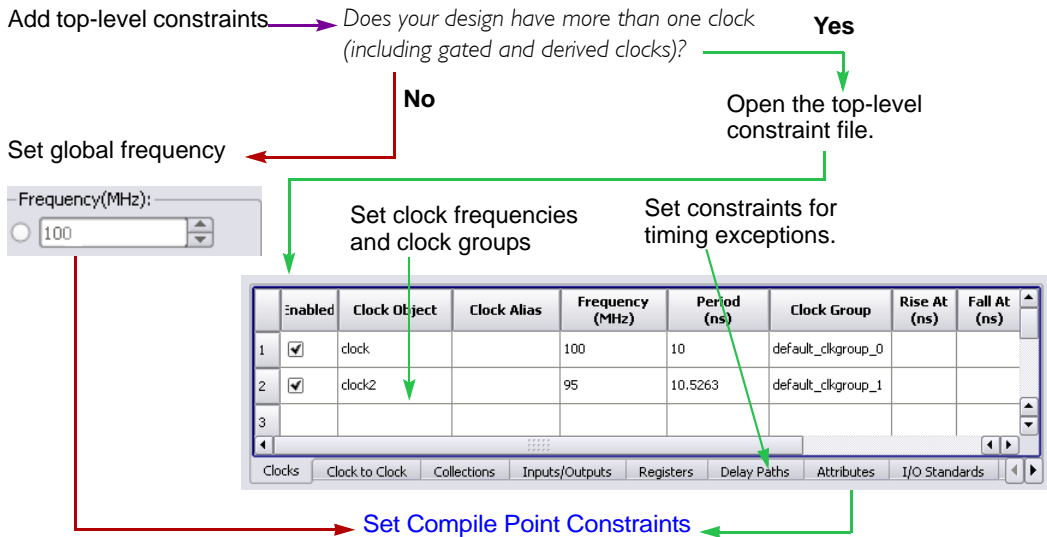
Set Constraints (Compile Point Synthesis)

# Set Constraints (Compile Point Synthesis)

Set constraints at the top level and for each compile point.

## Set Top-level Constraints

In a Compile Point Synthesis flow, set top-level constraints as you do in a normal synthesis flow. See the following figure.



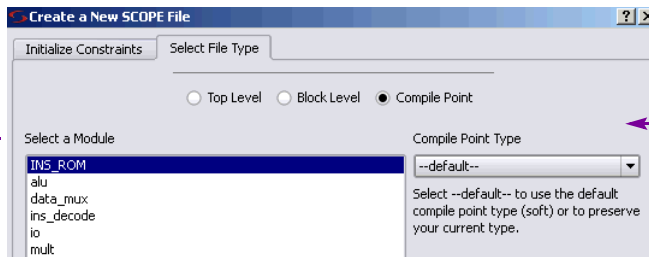
## Set Compile Point Constraints

Compile points can be nested. Parent compile points contain compile points within. Child compile points are nested within compile points. Parent constraints do not propagate to the child compile point, so you must set constraints for each compile point. However, compile point constraints are considered during synthesis of the parent.

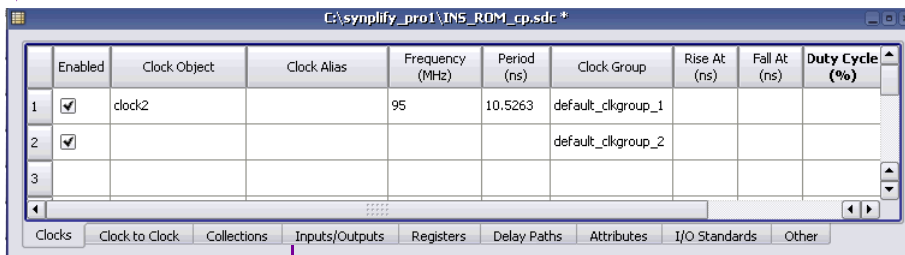
Open the SCOPE interface



Go to the Select File Type tab, click Compile Point, select a compile point module, and then click OK to initialize.



Define clocks, specify I/O delays, and set port constraints for the compile point



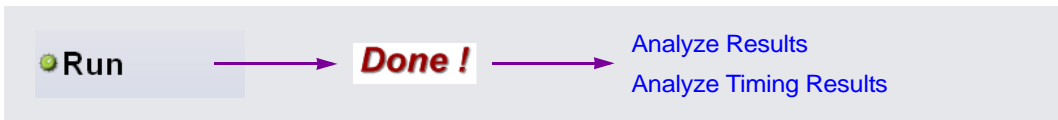
Save compile point constraint file.

Add file to project

Run

# Run

After you have set all the options and constraints, click Run. View the results of synthesis in the log file, or analyze them graphically using various built-in tools.



## CHAPTER 5

# Analyze Results

---

There are many ways in which you can check the implementation results. Use any of the following tools:

- [View the Log File](#), on page 31
- [Use the HDL Analyst® Tool](#), on page 31
- [Use FSM Viewer](#), on page 45
- [Other Tools to Validate Synthesis Results](#), on page 48

A summary is shown in the figure below.

**SELECTED INFORMATION**

Set to view critical path essentials

Set to view messages

**DETAILED INFORMATION**

View Log Messages Views

Log Parameter

Worst Slack	-0.741
clock - Estimated Frequency	118.6 MHz
clock - Requested Frequency	130.0 MHz

Log Watch

	Source Location	Log Location	Time
[2] CG346	Read full_case directive	eight_bit_uc_srr (...)	11:3
6 FX271	Instance "uc_alu.aluout_int_7_0_0_1[5]" with 2 loads has ...	eight_bit_uc_srr (...)	11:3
2 CL201	Trying to extract state machine for register state	eight_bit_uc_srr (...)	11:3
2 CD720	Setting time resolution to ns	eight_bit_uc_srr (...)	11:3
2		eight_bit_uc_srr (...)	11:3
10 CG364	Synthesizing module eight_bit_uc	eight_bit_uc_srr (...)	11:3
FX214	Generating ROM rom.Data_1[1:0]	eight_bit_uc_srr (...)	11:3
CD630	Synthesizing work.ins_rom.first	eight_bit_uc_srr (...)	11:3
MF135	Found RAM, 'regs.mem_regfile[7:0]', 32 words by 8 bits	eight_bit_uc_srr (...)	11:3
CL134	Found RAM mem_regfile, depth=32, width=8	eight_bit_uc_srr (...)	11:3
FX339	Found addmux in view:work.eight_bit_uc(verilog) inst uc_a...	eight_bit_uc_srr (...)	11:3

TCL Script Messages

Main Hierarchical Area

rev\_1 (eight\_bit\_uc)

Compiler Report

Mapper Report

Timing Report

Performance Summary

Worst slack in design: -0.741

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type
clock	130.0 MHz	118.6 MHz	7.692	8.433	-0.741	declared

Log File Links:

rev\_1

Hierarchical Area Report (eight...

Performance Summary

Worst slack in design: -0.741

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type
clock	130.0 MHz	118.6 MHz	7.692	8.433	-0.741	declared

Clock Relationships

Clocks	rise to rise	fall to fall	rise to fall	fall to
clock				

Use the HDL Analyst®

RTL View

Technology View

Critical Path

Do the results meet your design goals?

**No**

Specify Directives and Attributes

Refine Options to Improve Timing

**Yes**

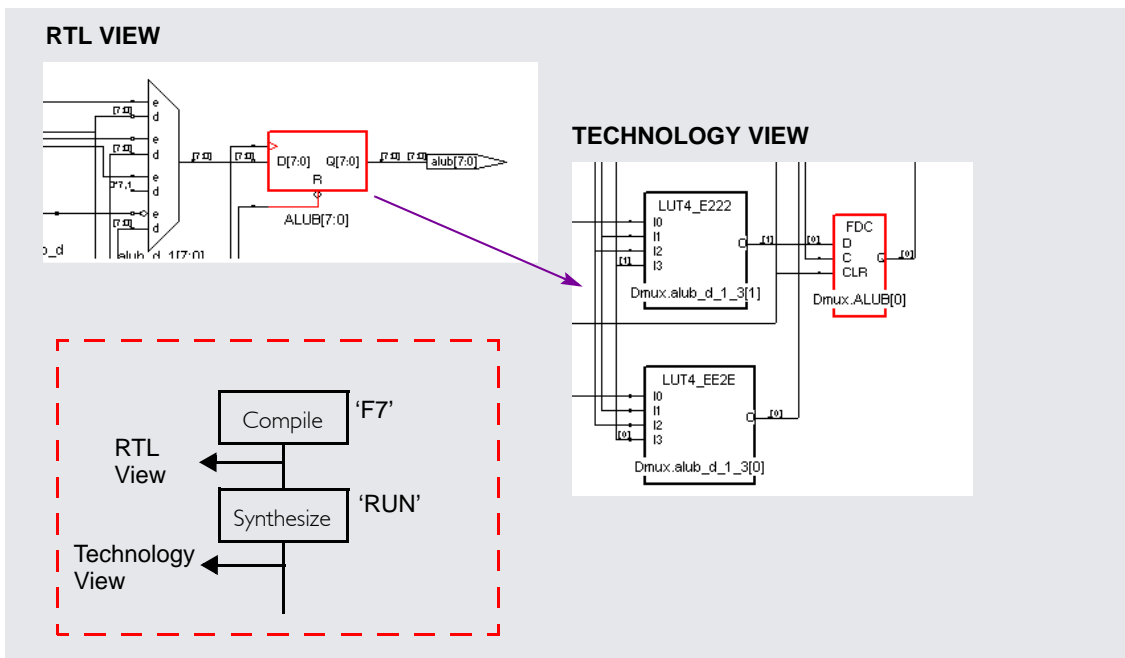
Synthesis is complete!

## View the Log File

The log file contains detailed information about the synthesis results, including details of the most critical paths, the number of primitives in the netlist, and the FSM encoding style. It is a good idea to run place and route even if the log file reports a timing goal failure. This is due to the fact the Synplify Pro tool is not knowledgeable about the final placement and routing of your design. The timing data given in the log file is an estimate.

## Use the HDL Analyst® Tool

The HDL Analyst environment provides graphical means to debug and analyze your design at two different stages of the design process: compilation ([RTL View](#)) and mapping ([Technology View](#)). The software maintains the names extracted from your code, so you can more easily follow the mapping of the logic from RTL to Technology view.



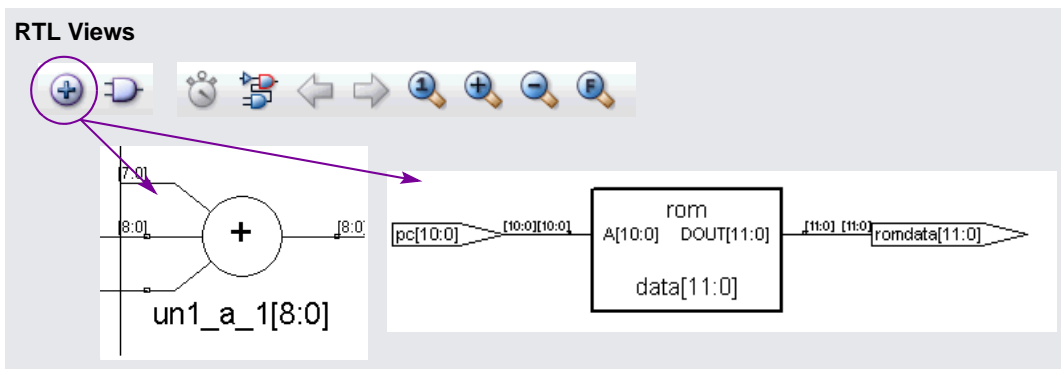
In addition to the RTL and Technology views and the timing analysis features (described separately in [Analyze Timing Results, on page 69](#)), the HDL Analyst tool has the following features that help you debug your implementation.

- [Design Hierarchy Exploration Tools](#)
- [Advanced Find Capabilities](#)
- [Filtered and Flattened Views](#)
- [Crossprobing Across Views](#)
- [Cross-tool Crossprobing](#)
- [Other Options](#)
- [Mouse Strokes](#)

## RTL View

This view shows the design after it is compiled. The software extracts the structure implied by the HDL language. The RTL view does not display target-specific components, so components are represented in a generic style.

To open the RTL view you must have a compiled design (use F7 to compile or click Run to synthesize). Then click on the icon, as shown in this figure:



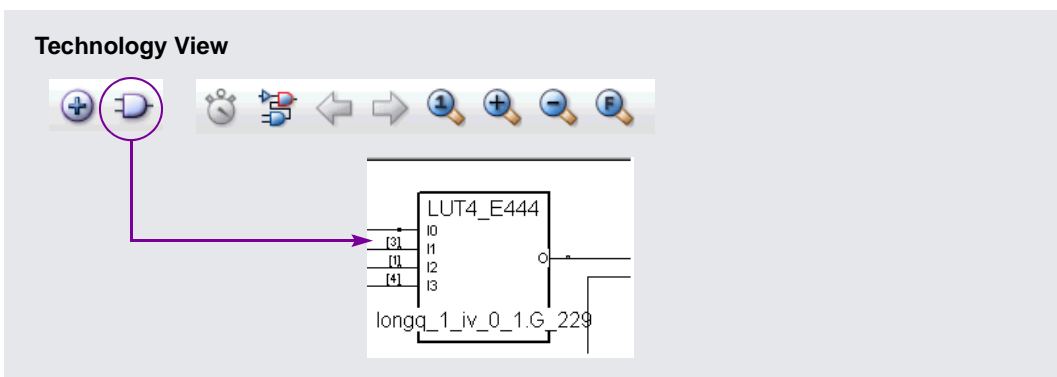


## Technology View

This view shows the design after synthesis is complete. The design is optimized and mapped to components that are specific to the target architecture. If the technology has the structure, a DFF with synchronous reset is mapped to a synchronous reset DFF.

The technology view shows you how the generic RTL structure is implemented for your technology. You also can view the critical path.

To open the Technology view, you must have a synthesized design (click Run to synthesize). Then click on the icon shown in the following figure:



## Design Hierarchy Exploration Tools

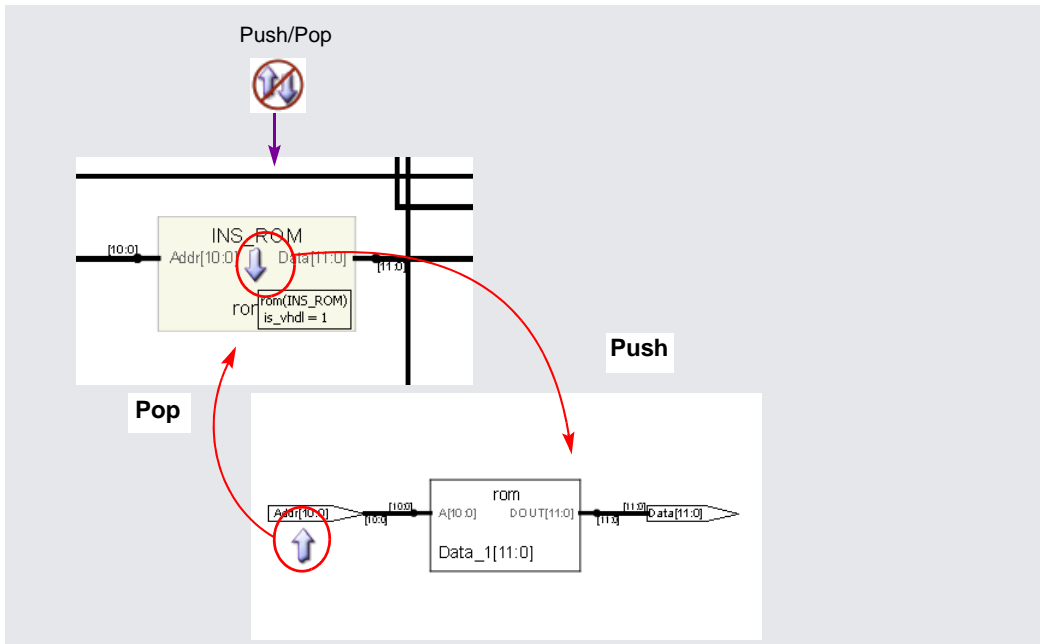
Since most large designs are hierarchical, the Synplify Pro HDL Analyst tool helps you view hierarchy details and put the details in context. You can browse and navigate hierarchy with Push/Pop mode and the Hierarchy Browser.

### Push/Pop Mode

You can navigate design hierarchy by pushing down into a high-level schematic object and popping back up. Pushing down into an object takes you to a lower-level schematic that shows the internal logic of the object. Popping up from a lower level brings you back to the parent higher-level object. You cannot pop up from the top level.

You can push down into the following kinds of schematic objects:

- Non-hidden hierarchical instances. To push into a hidden instance, unhide it first.
- Technology-specific primitives (not logic primitives)
- Inferred ROMs<sup>1</sup> and state machines in RTL views. Inferred ROMs, RAMs, and state machines do not appear in Technology views, because they are resolved into technology-specific primitives.



Click on the Push/Pop icon to traverse the design hierarchy. These cursors appear (↓) and (↑) when the design hierarchy can be traversed. This cursor appears (⊘) when the design hierarchy has reached its lowest or highest level.

Push or pop as follows:

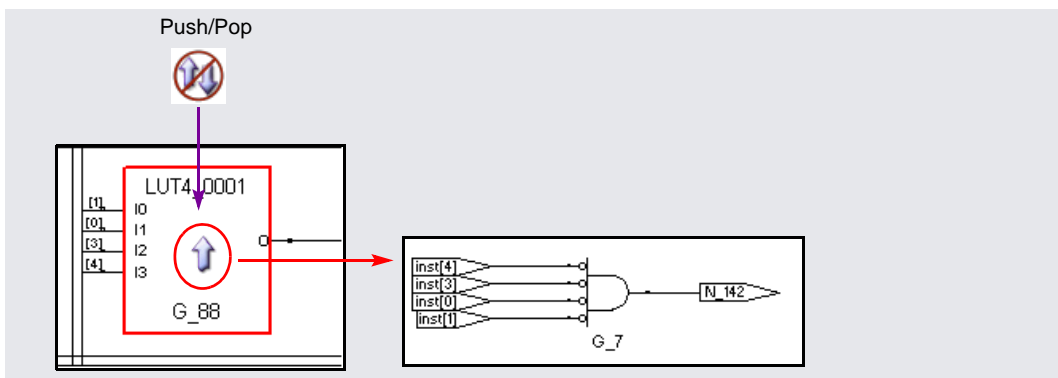
- To *pop*, place the pointer in an empty area of the schematic background, then click or use the appropriate mouse stroke. See [Mouse Strokes, on page 44](#).

---

**1. An inferred ROM is one that is created during synthesis; the ROM is not instantiated in your RTL code.**

- To *push* into an object, place the mouse pointer over the object and click or use the appropriate mouse stroke. See [Mouse Strokes](#), on page 44.

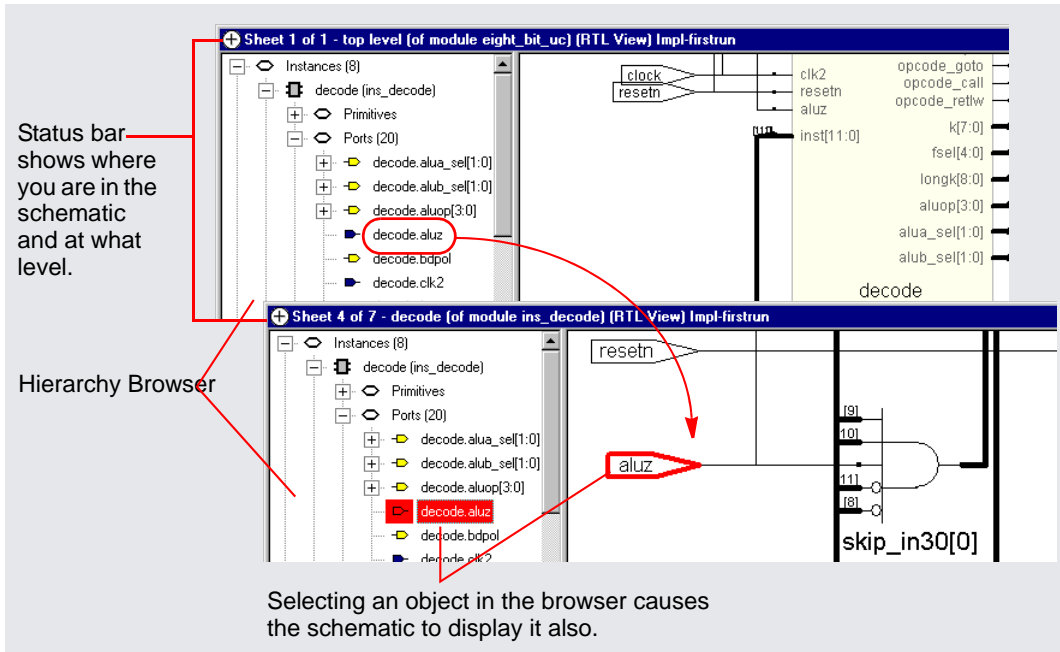
The following figure shows an additional use for the Push/Pop icon. You can push into a lookup table in a Technology view and check the mapped functions.



## Hierarchy Browser Navigation

When you open an RTL view or a Technology view, the Hierarchy Browser appears in the left pane of the view. The browser in the RTL view displays the hierarchy specified in the RTL design description; in the Technology view it displays the hierarchy after mapping.

A schematic and its associated Hierarchy Browser are linked so that you can crossprobe objects between them. Selecting an object in one displays it in the other.



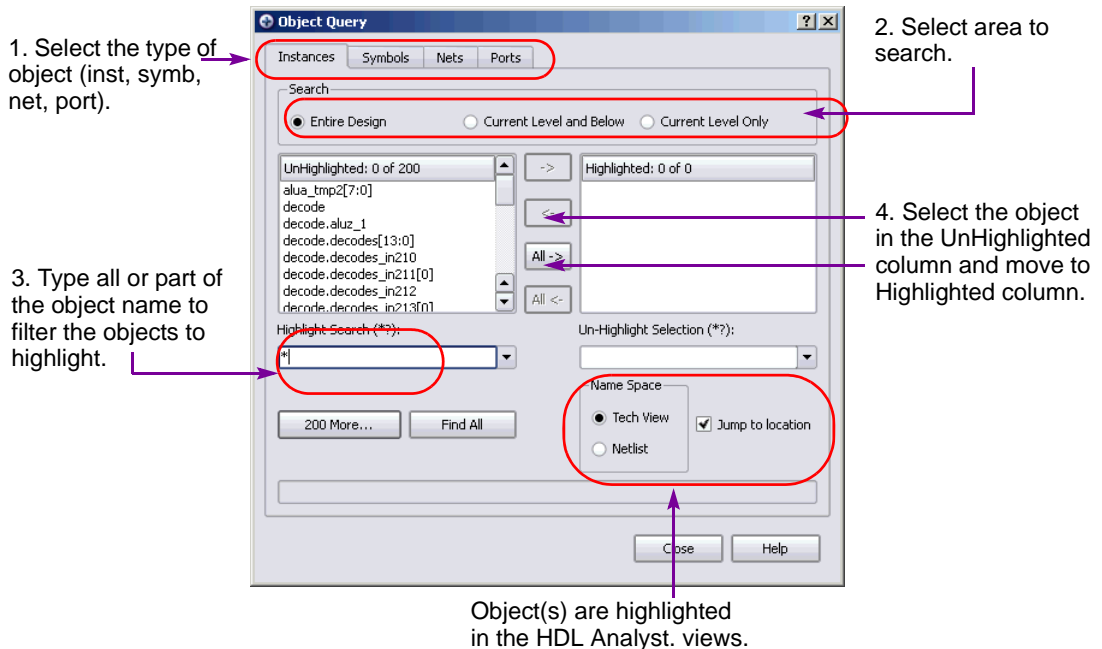
## Advanced Find Capabilities

You can locate objects using the command Edit -> Find. In an HDL Analyst view, this displays the Object Query dialog box, which lists candidate schematic objects by type (Instances, Symbols, Nets, or Ports) and lets you use wildcards to find objects by name.

You can find objects at any level of your design:

- Entire Design (all levels at once)
- Current Level & Below
- Current Level Only

All found objects are selected, whether or not they are displayed in the current schematic, so that you then can perform other operations on them.

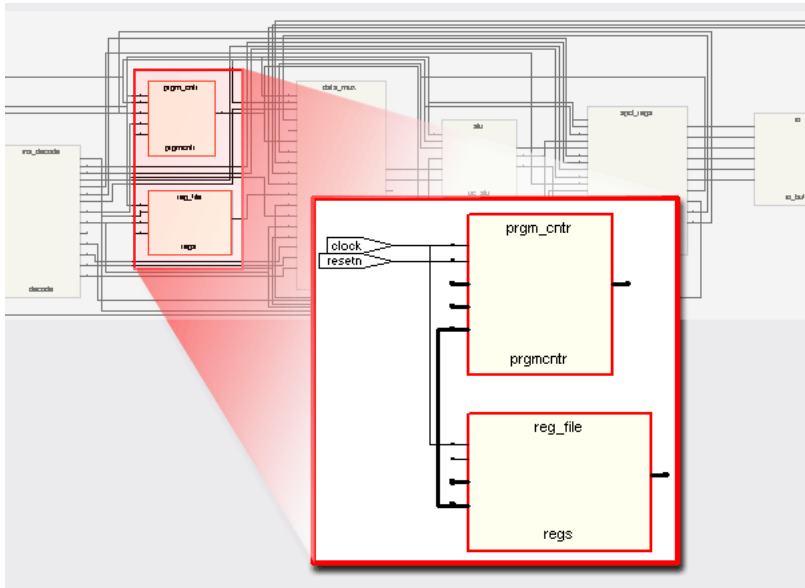


## Filtered and Flattened Views

Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Flattening eliminates the hierarchy of the design. Default flattening operations are global: the entire design is flattened from the current level down.

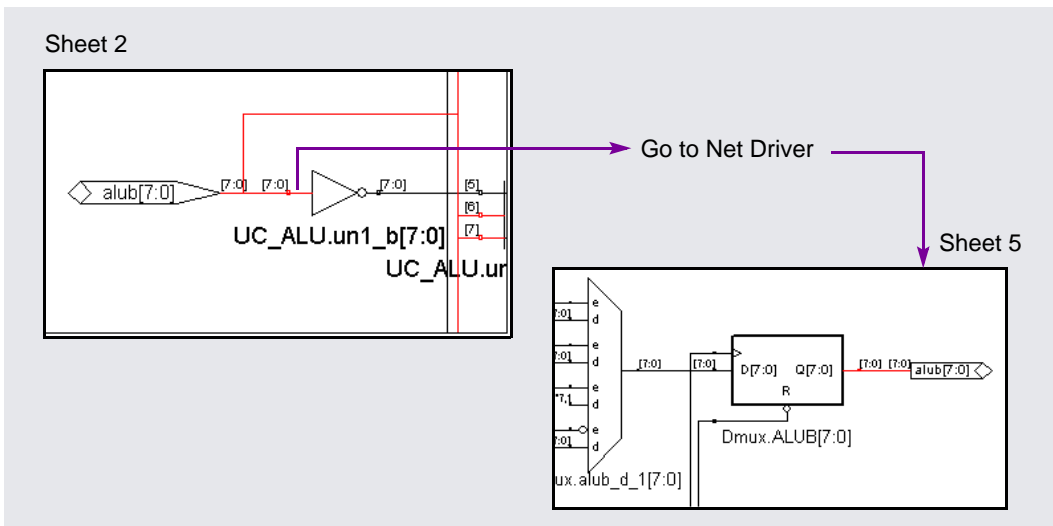
### Filtering

A filtered schematic shows a subset of your design.

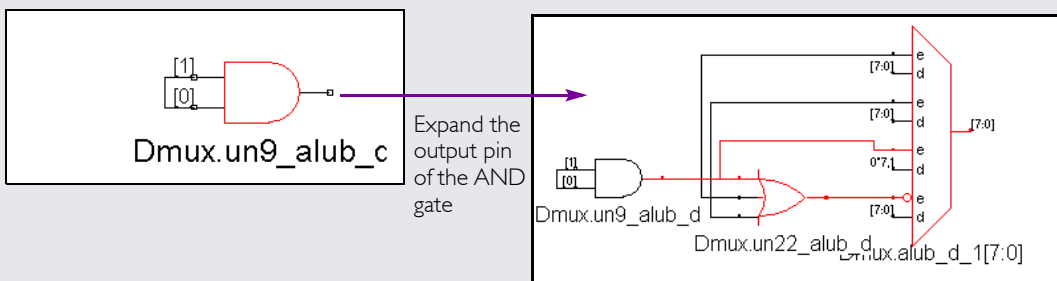


Any command that results in a filtered schematic is a filtering command. Some commands, like the Expand commands, increase the amount of logic displayed, but they are still considered filtering commands because the resulting view is still a subset. Other commands like Filter Schematic remove objects from the current display.

You can use filtering commands to trace signals at a single level of hierarchy or across the entire design. If you select a net, then right-click in the view and select Go to Net Driver, the view highlights the element driving the selected net. If the driver is on another schematic sheet, the software automatically moves to the correct sheet.



Similarly, you can start with the output of an AND gate, and use the Expand command to see what this gate drives:



Right-click in the HDL Analyst views to access the pop-up menu. The following table lists the most common filtering commands:

Filtering Command	Description
Filter Schematic, Isolate Paths	Reduces the displayed logic.
Dissolve Instances (filtered view)	Makes selected instances transparent, exposing their lower-level details.

Filtering Command	Description
Expand Expand to Register/Port Expand Paths Expand Inwards Select Net Driver Select Net Instances	Displays logic connected to the current selection.
Show Critical Path Flattened Critical Path Hierarchical Critical Path	Shows critical paths.

## Flattening

A flattened schematic contains no hierarchical objects. The flattening commands shown below are from the HDL Analyst menu. The most versatile commands are Dissolve Instances and Flatten Current Schematic, which you can also use for selective flattening.

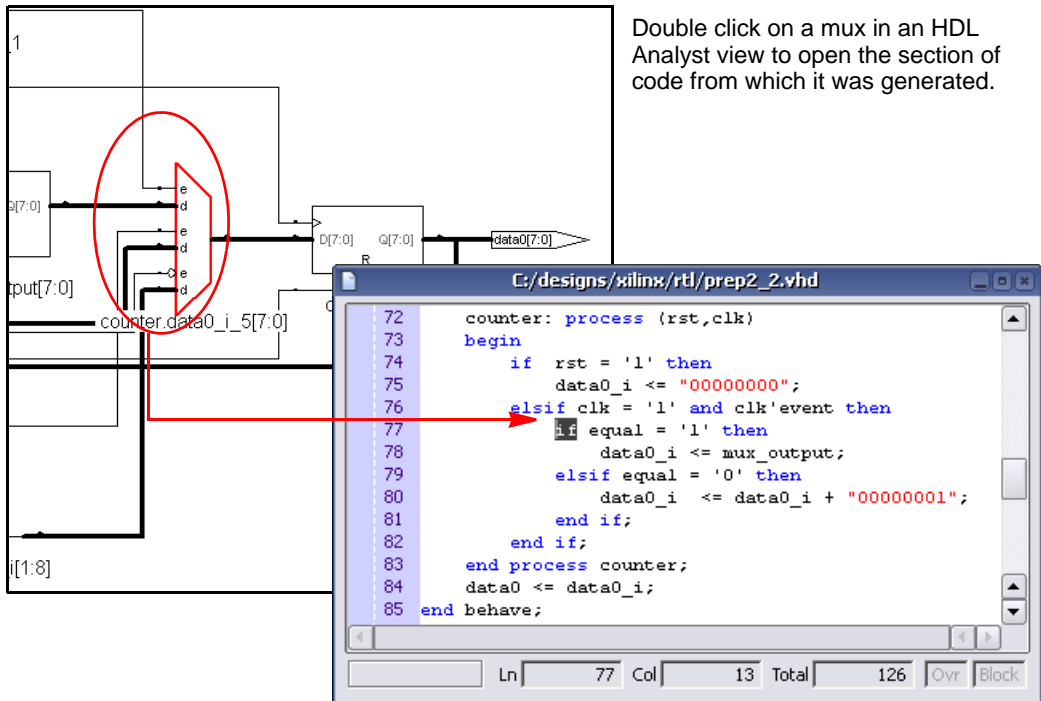
Command	Unfiltered Schematic	Filtered Schematic
Dissolve Instances	Flattens selected instances	--
Flatten Current Schematic (Flatten Schematic)	Flattens at the current level and all lower levels. RTL view: flattens to generic logic level Technology view: flattens to technology-cell level	Flattens only non-hidden transparent hierarchical instances; opaque and hidden hierarchical instances are not flattened.
RTL -> Flattened View	Creates a new, unfiltered RTL schematic of the entire design, flattened to the level of generic logic cells.	
Technology -> Flattened View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of technology cells.	



Command	Unfiltered Schematic	Filtered Schematic
Technology -> Flattened to Gates View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of Boolean logic gates. Creates a filtered, flattened Technology view schematic that shows only the instances with the worst slack times and their path.	
Technology -> Flattened Critical Path	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of Boolean logic gates. Creates a filtered, flattened Technology view schematic that shows only the instances with the worst slack times and their path.	
Unflatten Schematic	Undoes any flattening done by Dissolve Instances and Flatten Current Schematic at the current schematic level. Returns to the original schematic, as it was before flattening (and any filtering).	

## Crossprobing Across Views

You can crossprobe between HDL Analyst views, the Text Editor, the Tcl window, and the FSM viewer, because the views are tightly linked to your code. Double-clicking on a component in a HDL Analyst view takes you to the section of code from which it was generated. If you double click a mux, you open the section of code from which it was generated. In this example, the mux is generated from the VHDL case statement shown.



Conversely, you can select a section of code from the Text Editor and the corresponding RTL gates for that code are highlighted in the RTL view. You must first have an RTL view open. Then, select the code in the source file. You can crossprobe from any text file, for instance, place-and-route files.

To see just the structures that correspond to the selected code on one sheet, click Filter icon. This isolates the selected structures on one schematic sheet. Click the Back icon to return to the previous view.

## Cross-tool Crossprobing

The Synplify Pro software allows easy crossprobing between Synplify Pro processes and several other third-party tools:

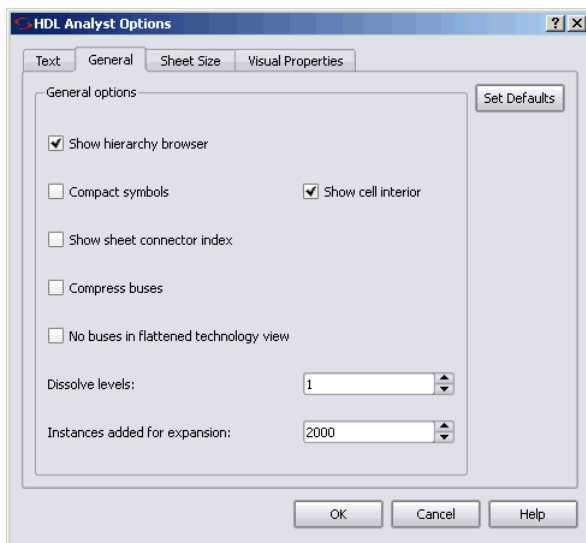
- Altera Quartus Place & Route software allows bidirectional crossprobing between Quartus II Floorplanner and the Synplify Pro synthesis tool's HDL Analyst tool.

- Xilinx Place & Route software allows crossprobing from the Xilinx timing file (.twr) to the HDL Analyst tool.

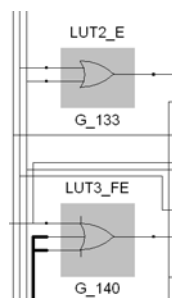
The Synplify Pro software also crossprobes between Synplify Pro processes. When two copies of the Synplify Pro tool are running simultaneously on one machine, enabling crossprobing allows both HDL Analyst tools to communicate with each other. For example, if external crossprobing is engaged and the same design is open in two copies of the software, selecting an instance in the HDL Analyst tool in one copy of the Synplify Pro tool results in the same instance being highlighted in the HDL Analyst tool of the second Synplify Pro tool.

## Other Options

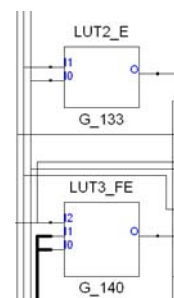
There are a number of useful options you can set using the Options -> HDL Analyst Options command. For example, you can view cell interiors in the Technology view, like the LUTs in this figure:



Show Cell Interior On



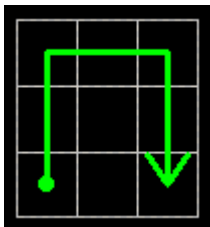
Show Cell Interior Off



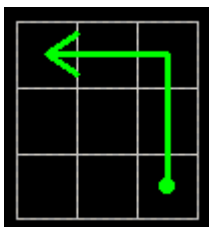
You can copy images from any schematic view to the clipboard. To do so, open a schematic view, and select Edit -> Copy Image. The cursor changes to a camera. Hold down the left button and drag the cursor to define the rectangular area you want to capture, and then release the button. The image is copied to the clipboard.

## Mouse Strokes

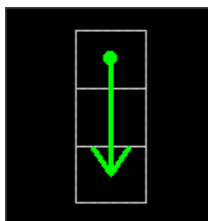
To access several frequently used menu commands, you can use the mouse while holding down the right mouse button as you draw the pattern. The mouse strokes you draw are interpreted on an invisible grid of one or three rows, depending on the stroke.



Redo stroke



Back stroke



Push stroke

For information about all the mouse strokes, select Help -> Mouse Stroke Tutor.

## Use FSM Viewer

The FSM (finite state machine) viewer is a graphic analysis tool that displays state transition bubble diagrams and implementation information for FSMs in the design. You can use this viewer to view state machines implemented by either the FSM Compiler or the FSM Explorer.

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 using the Push/Pop icon or the command from the popup menu.

The FSM viewer opens. The viewer consists of a transition bubble diagram and a table for the encoding and transitions.

2. The following table summarizes basic analysis operations.

To View...	Do this...
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.

The screenshot shows the FSM Viewer interface with the following components:

- Bubble Diagram:** A state transition diagram with nodes for LapDisplay, StopSecond, StopFirst, CountCont, and StopDiff. A purple arrow points to it with the label "Bubble Diagram".
- Transitions Table:**

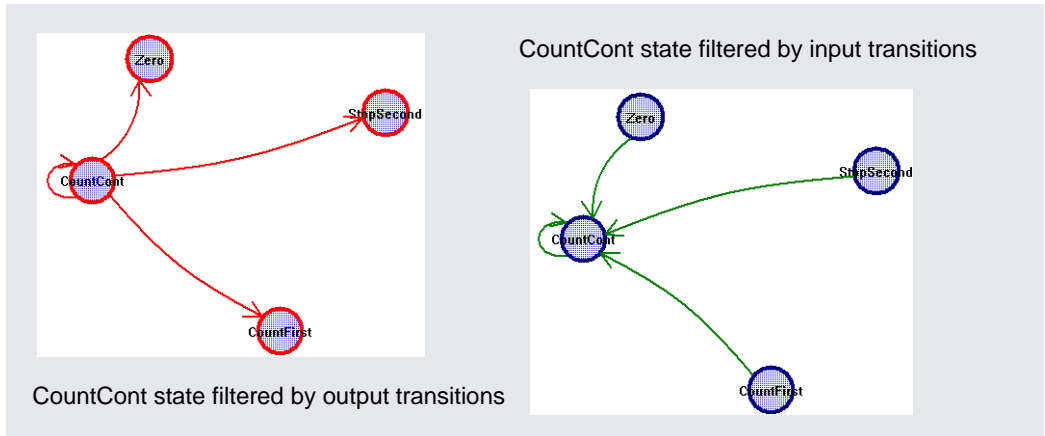
	From State	To State	Condition
2	Zero	LapDisplay	Lap&Start&Reset
3	StopSecond	StopSecond	!Lap&Start&Reset
4	StopFirst	StopSecond	Lap&Start&Reset
5	CountCont	StopSecond	Start&Reset
6	StopSecond	StopDiff	Lap&Start&Reset
7	StopDiff	StopDiff	!Lap&Start&Reset
8	StopFirst	StopFirst	!Lap&Start&Reset
9	CountFirst	StopFirst	Start&Reset
10	StopSecond	CountCont	Start&Reset
- State to Register Mapping Table:**

State	Register
Zero	cur_state[0]
CountFirst	cur_state[1]
CountCont	cur_state[2]
StopFirst	cur_state[3]
StopDiff	cur_state[4]
StopSecond	cur_state[5]
LapDisplay	cur_state[6]
- State Bit Matrix:**

State	cur_state[6]	cur_state[5]	cur_state[4]	cur_state[3]	cur_state[2]	cur_state[1]	cur_state[0]
Zero	0	0	0	0	0	0	1
CountFirst	0	0	0	0	0	1	0
CountCont	0	0	0	0	1	0	0
StopFirst	0	0	0	1	0	0	0
StopDiff	0	0	1	0	0	0	0
StopSecond	0	1	0	0	0	0	0
LapDisplay	1	0	0	0	0	0	0

- To view just one state, click the bubble for the state to select it. Then 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 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 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 crossprobe from the FSM viewer, open the view to which you want to crossprobe (source code file for example) and click the state bubble in the FSM viewer.

The software highlights the corresponding object or code in the open view. You can also crossprobe from the transition table if you use a onehot encoding style. With all other styles, you cannot crossprobe from the transition table because the number of registers in the table may not match the number of registers in the RTL or Technology views. A one-to-one correspondence might not exist because of optimizations during synthesis.

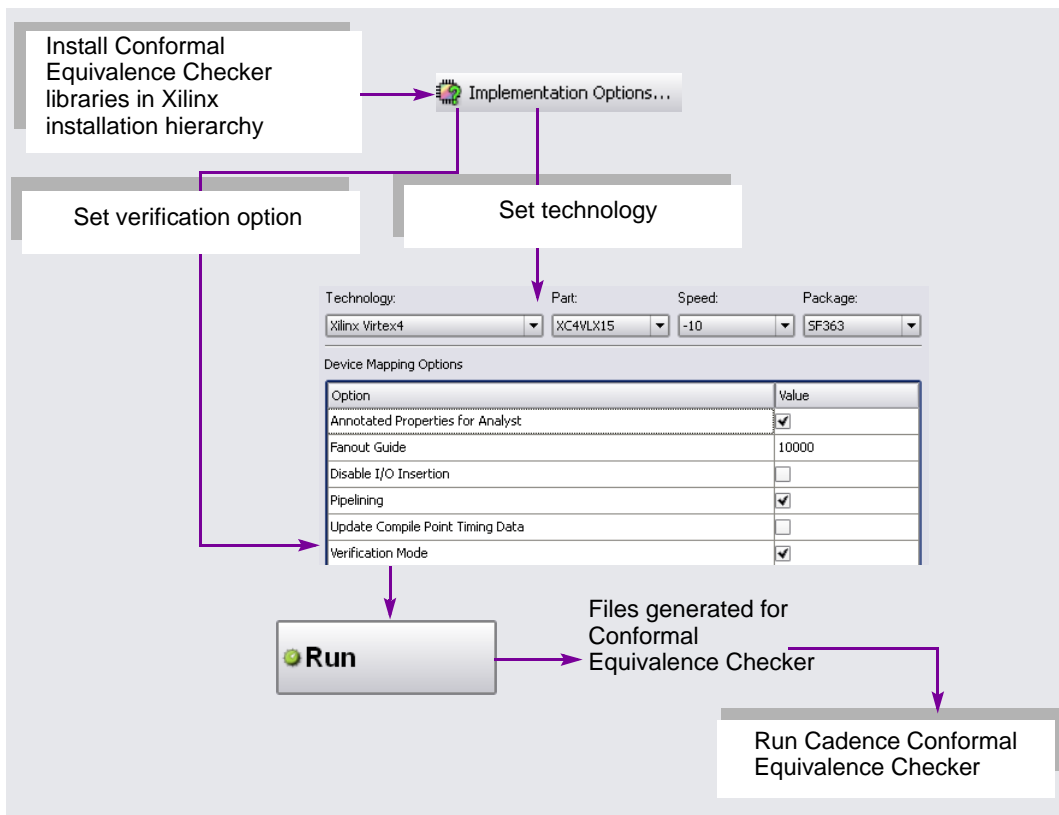
## Other Tools to Validate Synthesis Results

Validate Synplify Pro synthesis results with the following tools:

- [Use Formal Verification](#)
- [Use `syn\_probe` Attribute](#), on page 49
- [Identify RTL debugger](#), on page 49

### Use Formal Verification

For certain Xilinx technologies, you can validate your results using the Cadence Conformal Equivalence Checker tool. See the application note for more information. The following figure is an overview of the flow.





## 6. Validate Synthesis Results with Simulation

You can optionally generate a post-synthesis netlist file in Verilog (.vm) or VHDL (.vhm) format. This is a structural netlist of the synthesized design, and differs from the original RTL you used as input for synthesis.

Typically, you use this netlist for gate-level simulation, to verify your synthesis results. Some designers prefer to simulate before and after synthesis, and also after place and route. This approach helps to isolate the stage of the design process where a problem occurred.

## Use `syn_probe` Attribute

The `syn_probe` attribute works as a debugging aid, inserting probe points for testing and debugging the internal signals of a design without HDL source modification.

## Identify RTL debugger

The Identify product is the first and only software tool that allows FPGA designers and ASIC prototyping designers to functionally debug their hardware directly in their RTL source code. This allows functional verification with RTL designs 10,000 times faster than RTL simulators, and enables the use of in-system stimulus for applications like networking, audio and video, and hardware/software designs. Identify software allows designers to directly select signals and conditions in their RTL source code for debugging and the results are viewed directly in the RTL source code. The Identify tool can also save results in standard VCD format that can be used with most waveform viewers. See [Using Identify with Synplify Pro, on page 89](#) for more information.



## CHAPTER 6

# Specify Directives and Attributes

---

Directives and attributes influence the way your design is synthesized (compile stage) and optimized (map stage), respectively, and can be used to improve design quality of results (QoR). Refer to the following topics:

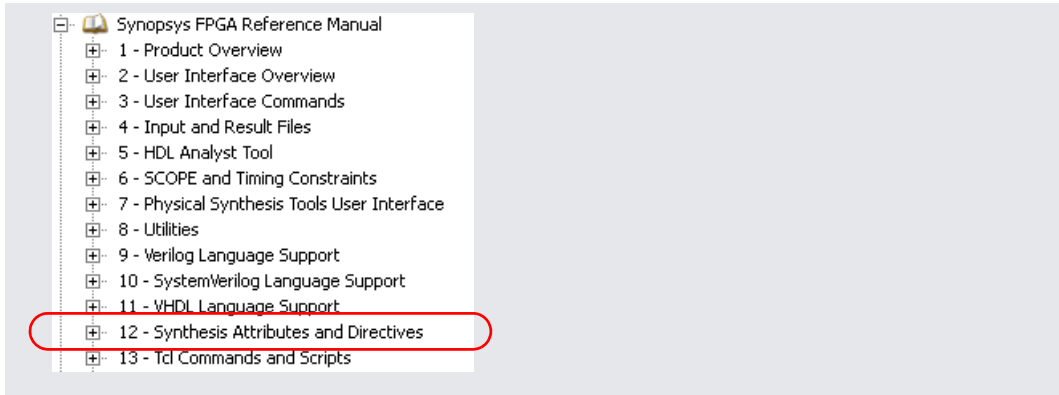
- [Using Directives and Attributes](#)
  - [syn\\_maxfan](#)
  - [syn\\_keep](#)
  - [syn\\_ramstyle](#)

## Using Directives and Attributes

Directives must be specified in the HDL source code because they affect the compile stage and the inference of high-level structures. Attributes can be specified using any of the following methods:

- source code
- the SCOPE interface
- Tcl constraint file (.sdc)
- RTL or Technology views (right-click for pop-up menu)

There are two classes of attributes: target-specific attributes and general attributes. For details about directives and attributes, select Help -> Help, and then scroll down to the section shown below. It is strongly advised that you review this section of help. For discussions of target-specific attributes, check the help section on your target technology.



A few commonly used attributes and directives are described below.

## syn\_maxfan

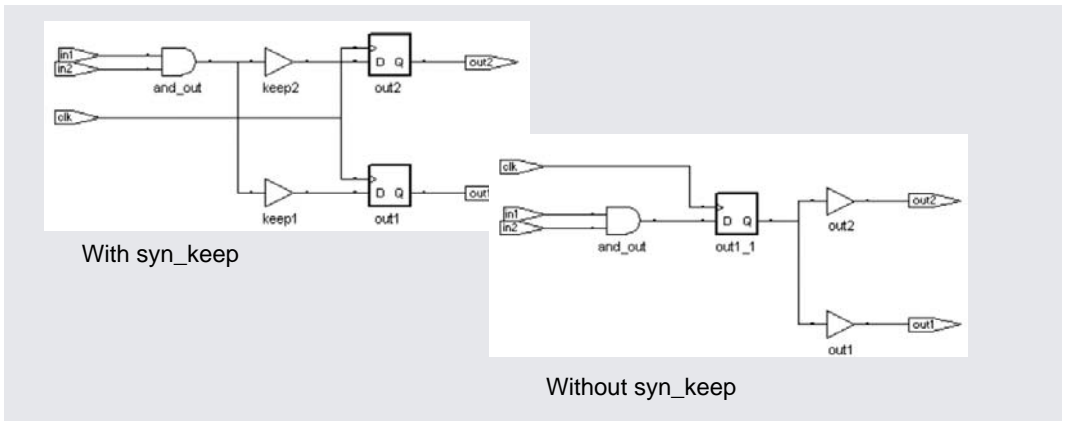
Sets a guide for maximum fanout. You can set this globally from the Implementation Options dialog box, or set it on individual modules. The following figure shows the `syn_maxfan` attribute entered in the SCOPE Attributes pane for an individual module:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	Data_1[11:0]	syn_maxfan	0	integer	Overrides the default fanout	
2								
3								

Attributes

## syn\_keep

Preserves the specified net from being optimized away during synthesis. The software places a temporary buffer on the net as a placeholder so that it is not optimized. You can view this buffer in the HDL Analyst views, but it is not in the final netlist.



## syn\_ramstyle

Determines how inferred RAMs are implemented. The valid values vary depending on the target technology family. You can turn off RAM inference by setting the value of this attribute to registers.



## CHAPTER 7

# Basics of Timing Constraints

---

Synplify Pro synthesis is timing-based. Its algorithms are governed by the timing requirements of your design as set by the constraints, attributes and directives you specify. The more precise the timing information you provide, the better the results produced.

Refer to the following topics:

- [Specifying Timing Information](#)
- [Clock Descriptions](#)
- [Clock Groups](#)
- [Rise and Fall Constraints](#)
- [Input and Output Delays](#)
- [Multicycle Paths](#)
- [I/O Standard](#)

## Specifying Timing Information

The following are some of the ways in which you provide timing information to the Synplify Pro tools (use the tabs in the SCOPE UI):

- **Clock definitions**  
For designs with multiple clocks, use a combination of individual clock constraints and a default global frequency to define all the clocks in your design accurately. See [Clock Descriptions, on page 56](#) for an explanation.

- **Clock domains**  
Assign unrelated clocks to separate clock groups. Clocks that are in the same clock group are considered to be related and affect timing calculations. See [Clock Groups, on page 59](#) for more information. Use Rise and Fall values to define relationships between source and destination clocks. See [Rise and Fall Constraints, on page 60](#) for details.
- **Multicycle paths and false paths**  
If your design has multicycle paths or false paths that you do not want the Synplify Pro tool to consider during timing based synthesis, you may identify these in the constraints file. The software considers paths to asynchronous sets and resets as false paths, so you do not have to constrain these paths explicitly.
- **Resource sharing**  
Turn this OFF to get better circuit performance. When OFF, the tool does not share the arithmetic modules in your design across multiple functions.
- **Symbolic FSM (finite state machine) compiler**  
Turn this ON to get better circuit performance. When on, the synthesis software looks for state machines in your design, analyzes and optimizes them, and encodes them based on size.
- **FSM Explorer**  
Turn this ON to get better circuit performance. When on, the FSM Explorer tries different encoding styles and picks the best style for the state machine based on overall design constraints.

## Clock Descriptions

You must accurately describe timing in your design, because it can significantly impact what the software considers the most critical path. The most important information you can provide is the performance of each clock in your design, including the top-level clocks, derived clocks, gated clocks, or other signals in your design acting as clocks.

Set a global frequency. The software applies this default frequency to all clocking signals that you have not specifically identified with individual clock constraints.



---

**Note:** Do not over-constrain your clocks. Keep the value within 5%-10% of the actual target.

---

Here is an example of some clocking situations.

## Example

This design has two top-level clocks, `clk` and `ins_clk`. When the constraint file is initialized, the SCOPE interface shows both clocks assigned the default frequency of 8MHz. Setting the default to a small value allows parts of the design that are not timing-critical to be optimized for area.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)
1	<input checked="" type="checkbox"/>	clk		8	125	default_clkgroup_0				
2	<input checked="" type="checkbox"/>	ins_clk		8	125	default_clkgroup_1				
3										

Now set `clk` and `ins_clk` to the frequencies you want and enable the constraints.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)
1	<input checked="" type="checkbox"/>	clk		75	13.3333	default_clkgroup_0				
2	<input checked="" type="checkbox"/>	ins_clk		33	30.303	default_clkgroup_1				
3										

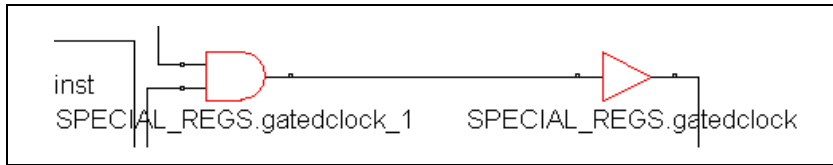
The design includes three other clocking situations: a divided clock off a register, a gated clock, and some unregistered clock logic. The default frequency of 8MHz is applied to the unregistered logic, and the log file reports it as a system clock (System).

Starting Clock	Requested Frequency
Clk	75.0 MHz
SPECIAL_REGS.gatedclock	10.0 MHz
UC_ALU.halfclk	25.0 MHz
ins_clk	33.0 MHz
System	1.0 MHz

To identify the divided clock for the software, locate the register in the RTL view (see [RTL View, on page 32](#) for a description). Drag and drop or type the name of the register into the SCOPE Clocks pane and specify a frequency that is half (37.5Mhz) of clk (75.0Mhz). When you type a name in the SCOPE interface, be sure to type the name accurately because the constraint file is written out in Tcl format, which is case-sensitive.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)
1	<input checked="" type="checkbox"/>	clk		75	13.3333
2	<input checked="" type="checkbox"/>	ins_clk		33	30.303
3	<input checked="" type="checkbox"/>	SPECIAL_REGS.gatedclock		10	100
4	<input checked="" type="checkbox"/>	UC_ALU.halfclk		37.5	26.6667

The gated clock is the most interesting clocking situation. To attach a clock to the signal coming off an inferred AND gate, attach a `syn_keep` synthesis directive to the signal or wire. This directive does two things: it creates a virtual buffer (shown in the following figure) and it does not optimize the net. The virtual buffer provides a “*synthesis handle*” to which you can attach attributes like clock definitions or multicycle paths. You can view this buffer in the RTL view (the schematic view generated after the compile stage) and drag-and-drop it into the SCOPE spreadsheet if you need to attach a constraint to it.

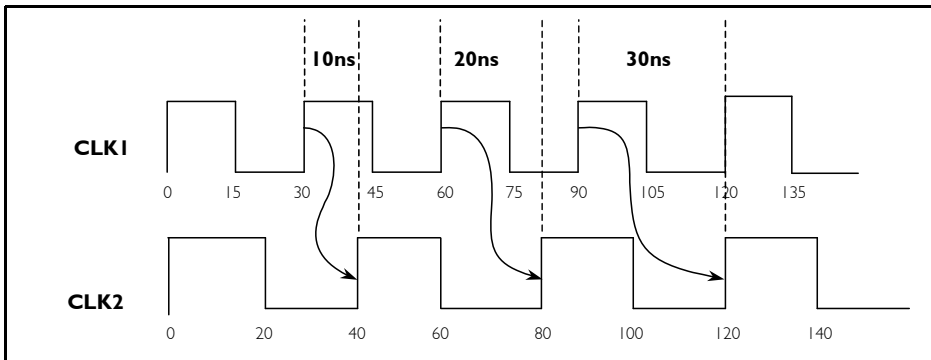


By preventing an internal net from being optimized away, the `syn_keep` directive guarantees that the net name is maintained. This can be helpful for probing internal nets or to prevent nets from being optimized.

## Clock Groups

The timing engine uses clock groups to define clocking schemes. It assumes that clocks in the same clock group are synchronized with each other and treats them as related clocks. Typically, clocks in a clock group are derived from the same base clock. The timing analyzer automatically calculates the relationships between clocks in a clock group and analyzes all paths between them. It calculates the time available based on the periods of two related clocks.

The waveforms in the following figure show how the software determines the worst posedge-to-posedge timing between clocks CLK1 and CLK2. All paths that begin at CLK1 rising and end at CLK2 rising are constrained at 10ns.



Conversely, clocks in different clock groups are considered unrelated or asynchronous. Paths between clocks from different groups are automatically marked as false paths and ignored during timing analysis and optimization.

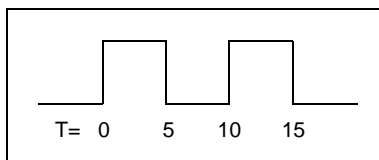
By default, all clocks in a design are assigned to separate clock groups and named `default_clkgroup<n>`. If clocks are related, you must re-assign them to the same clock group using the Clock Group field in the SCOPE interface. See [Using Multiple Clock Domains, on page 87](#) for more information.

## Rise and Fall Constraints

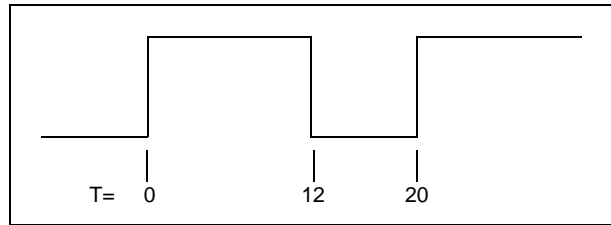
By default, the Synplify Pro tool assumes an ideal clock network; the clock arrives at all clocked registers at the time specified in the Rise and Fall fields. By default, the constraints assume a 50% duty cycle clock with the rising edge at 0 and the falling edge at  $\text{period}/2$ .

The synthesis tool computes relationships between the source clock and destination clock on a path by using the Rise and Fall numbers. To understand how the relationships between source and destination clocks are computed using the Rise and Fall numbers, consider the following example.

- Clk1 is a clock with a period of 10 ns in clock group `default_clkgroup`. Since none of the other fields are specified, this is a 50% duty cycle clock rising at 0 and falling at 5. There is no propagation delay between the clock source and the clock ports on the registers clocked by Clk1.



- Clk2 is a 200 MHz (5ns) clock, also in the `default_clkgroup`. This means that the timing analyzer considers all paths from Clk1 to Clk2 and from Clk2 to Clk1.
- Clk3 is a clock with a period of 20 ns in `clkgrp2`. This means all paths between Clk3 and either Clk1 or Clk2 are automatically treated as false paths. In addition, Clk3 has a Rise of 0 and a Fall of 12, which means it has a 60% duty cycle.



- Clk4 is a virtual clock. This means that there can be no port or instance named Clk4 in this design. However, there may be top-level ports on the chip which are clocked by Clk5 outside the chip. Input arrival times and output required times for such ports can be specified relative to Clk5.

## Input and Output Delays

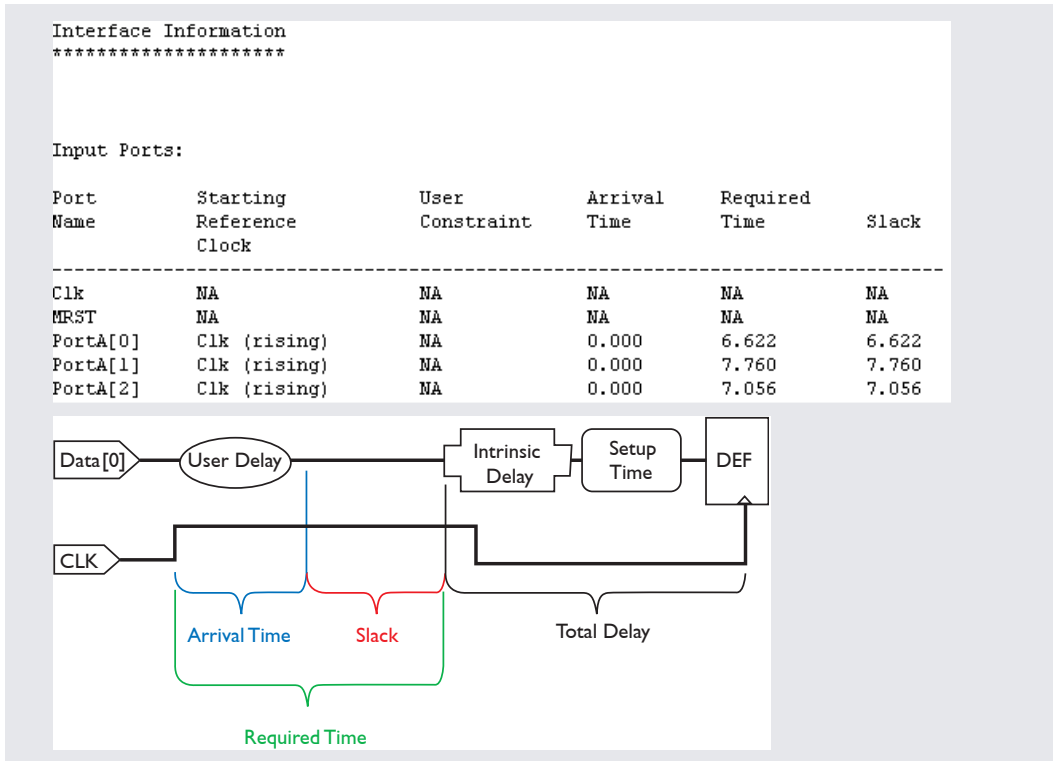
It is important to understand the role of input and output delays as they apply to Synplify Pro synthesis. You need to recognize which constraint is most effective in a given situation: a synthesis or place-and-route constraint.

The Synplify Pro engine can minimize logic between an input and a register or between a register and an output. If there is no logic between these points, the synthesis engine cannot affect the constraint. In this case, it would be better to use a place-and-route constraint to minimize the routing between the two points.

The log file contains a section for the interface. It reports required times, arrival times, slack, and user constraint information for inputs and outputs according to the register that they are driving or driven by, and the clock which controls that register.

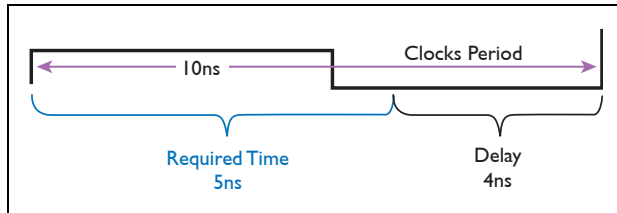
## Input Ports

A typical log file report for input ports looks like the one below. Let's take a closer look at how the numbers are calculated and see how you can influence them with an `input_delay` constraint.



In the following example, `data1[0]` feeds a register that is controlled by `clk` (the reference clock). The period for `clk` is set at 10ns, so `data1[0]` is required to arrive at the input a certain time after the controlling edge of clock so that it can propagate to the register in time for the next controlling edge. The required time is determined as follows:

$$\text{Required Time} = \text{Clock Period} - (\text{intrinsic} + \text{setup time})$$



Add an input delay of 3ns to data1[0]. Do this in the SCOPE Inputs/Outputs pane:

	Enabled	Port	Type	Clock Edge	Value (ns)	Route (ns)	Comment
1	<input checked="" type="checkbox"/>	data1[1:]	input_delay		3		
2	<input type="checkbox"/>	<output default>	output_delay				

This constraint indicates that data1[0] arrives 3ns later than the default 0ns. This reduces the slack margin available. When you resynthesize, the log file reflects these changes.

```

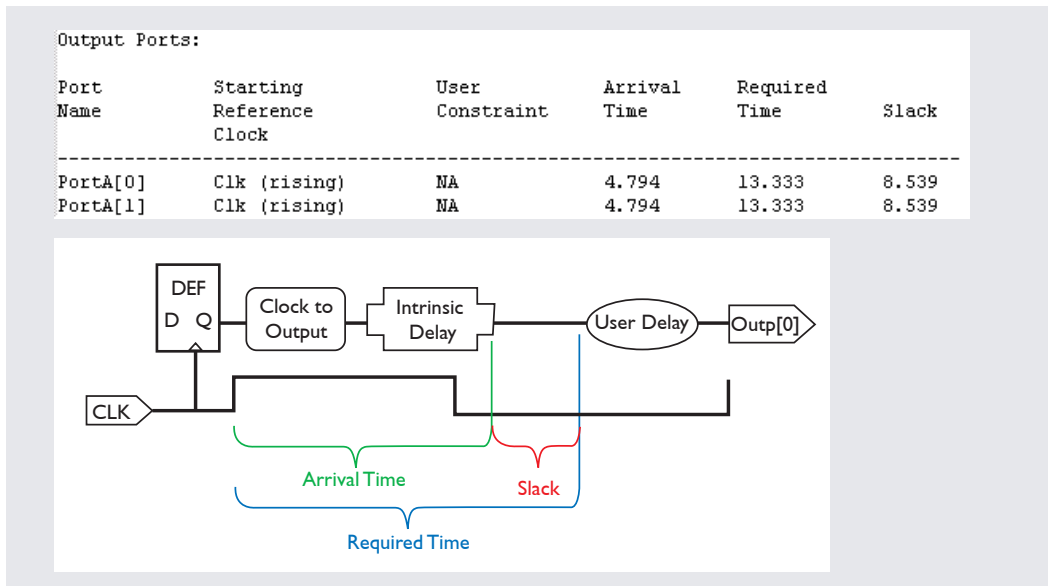
Interface Information
*****

Input Ports:

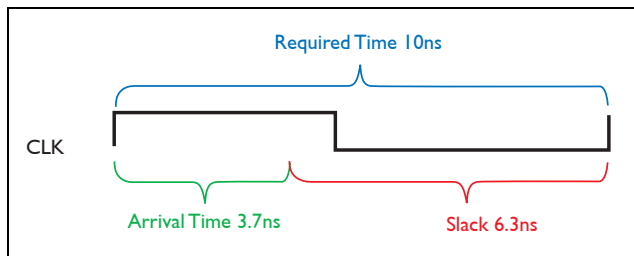
Port Name      Starting Reference Clock      User Constraint      Arrival Time      Required Time      Slack
-----
clk            NA                               NA                   NA                NA                NA
data1[0]      clk                             3.0                 3.0               6.0               3.0
    
```

## Output Ports

A typical log file report for output ports looks like the one below. Look at how the numbers are calculated and see how you can influence them with an `output_delay` constraint. The required time is derived from the period of the reference clock for the register minus any user delay. The arrival time is the clock to output time plus intrinsic delay. The slack then is the required time minus the arrival time.



In the following example, initially the required time is equal to the clock period, because no user constraint is set. The arrival time of 3.7ns is the sum of the clock-to-output delay and the intrinsic delay, leaving a slack margin of 6.3ns.



Add a 3 ns output delay constraint using the SCOPE Inputs/Outputs pane:



	Enabled	Port	Type	Clock Edge	Value (ns)	Route (ns)	Comment
1	<input checked="" type="checkbox"/>	data[1:]	output_delay		3		
2							

The result of the `output_delay` constraint is to reduce the required time, thus reducing the available slack. The resulting log file change is shown below:

Output Ports:

Port Name	Starting Reference Clock	User Constraint	Arrival Time	Required Time	Slack
outp[0]	clk (rising)	3.000	3.700	7.000	3.300

The diagram shows a clock signal (CLK) with a rising edge. A blue bracket above the signal indicates a 'Required Time 7ns' from the rising edge to the output. A green bracket above the signal indicates a 'User Constraint 3ns' from the rising edge to the output. A green bracket below the signal indicates an 'Arrival Time 3.7ns' from the rising edge to the output. A red bracket below the signal indicates a 'Slack 3.3ns' from the arrival time to the user constraint.

## Multicycle Paths

The multicycle path constraint identifies certain paths that take multiple clock cycles to complete and which you should exclude from single-clock-cycle based synthesis. You can specify a multicycle path from an origination register (`from`), to a destination register (`to`), or as passing through a net (`through`). By setting the cycle multiplier to a high value (e.g. 100), you identify the false paths to be ignored.

You set a multicycle path constraint from the SCOPE Path Delays tab, by selecting a Delay Type of Multicycle. You can use one of the following methods to specify the register or net: drag-and-drop it from an RTL view, select from the cell pulldown menu, or manually type in the full hierarchical name. Regard-

less of whether you are coding VHDL or Verilog, all SCOPE names are case-sensitive because the constraints are recorded in Tcl, which is case-sensitive. The following figure shows a from-to multicycle constraint.

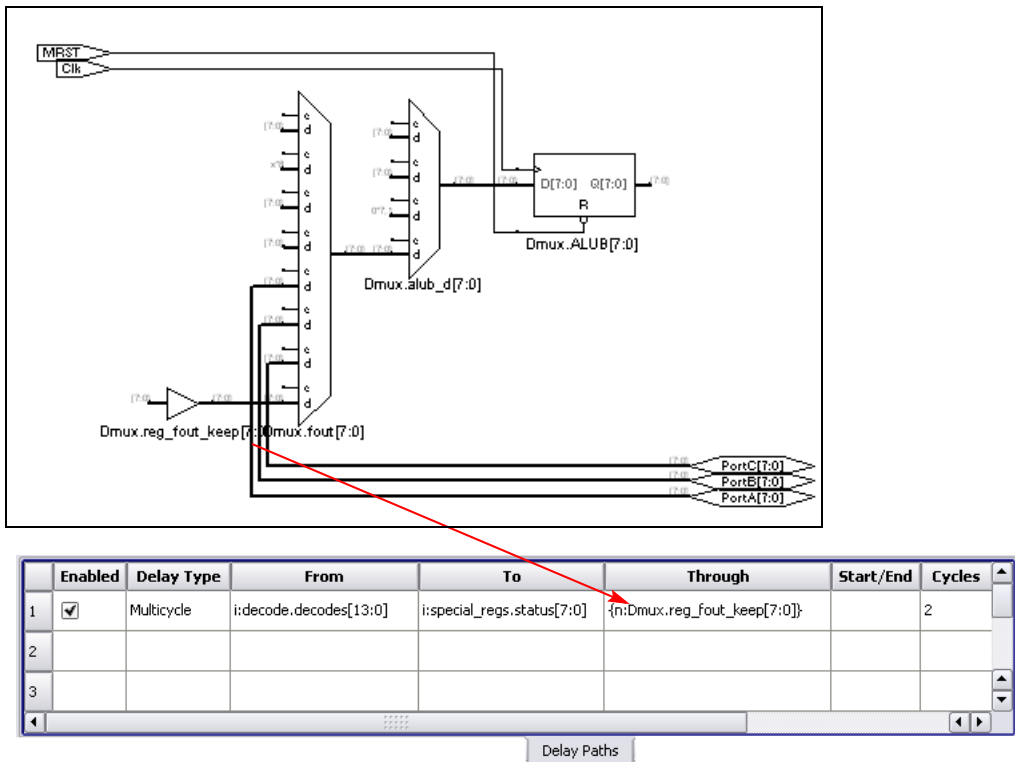
	Enabled	Delay Type	From	To	Through	Start/End	Cycles	Max Delay(ns)
1	<input checked="" type="checkbox"/>	Multicycle	i:dmux.alua[7:0]	i:uc_alu.aluz			2	
2								
3								

Delay Paths

You can also edit constraints (\*.sdc file) with a text editor. To do so, select the file in the Project window, right-click, and select Edit as Text from the popup menu.

## Setting Multicycle Path Constraints

To and from constraints are the simplest to set because they start or end at a register. The following example shows how to set a multicycle path through constraint. The filtered RTL view generated after the compilation stage shows a partial circuit where the signals all propagate to the register Dmux.ALUB in one clock cycle. However, reg\_fout is allowed two clock cycles. If this causes the Synplify Pro tool to identify the path from reg\_fout to ALUB as a critical path, you must define the two-cycle propagation correctly for the tool. You cannot define the multicycle path with a to constraint on Dmux.ALUB, because this would cause the other signals to be allowed two clock cycles, which is not correct.



Instead, to single out the specific two-cycle path, use a through constraint on the `reg_fout` net highlighted above in addition to the from and to points. Make sure you precede the net name with an `n:` (to identify the object as a net) if you type it manually in the SCOPE interface.

## I/O Standard

Available for certain technology families, you can use the I/O Standards panel of the SCOPE interface to specify a standard I/O pad type to use in the design. See help in the tool for information.



## CHAPTER 8

# Analyze Timing Results

---

There are many tools to analyze the timing of your design:

- [Critical Path Report](#)
- [Timing Information Display](#)
- [Critical Path Analysis in a Technology View](#)

## Critical Path Report

You can check details of the critical path in the log file.

- Note the start and end points of the critical path. For multi-clock designs, look for the clock with the worst slack.
- Check fanout. To limit fanout on a given net, use the `syn_maxfan` attribute. When enabled, the synthesis software replicates logic to reduce the fanout. For an example of adding an attribute, see [Specify Directives and Attributes](#).

The following figure shows a critical path in the log file.

```

Path information for path number 1:
- Setup time:                0.392
= Required time:             6.275

- Propagation time:          7.914
= Slack (non-critical) :    -1.639

Starting point:               PrgmCntr.PC[7] / Q
Ending point:                 SPECIAL_REGS.INST[0] / D
The start point is clocked by Clk [rising] on pin C
The end point is clocked by  SPECIAL_REGS.gatedclock [rising] on pin C
    
```

Instance / Net Name	Type	Pin Name	Pin Dir	Delay	Arrival Time	Fan Out
PrgmCntr.PC[7]	FDPE	Q	Out	1.995	1.995	
pc[7]	Net					3
ROM.G_464	LUT4	I1	In		1.995	
ROM.G_464	LUT4	0	Out	1.471	3.465	
G_464	Net					4
ROM.N_8_i	LUT3	I0	In		3.465	
ROM.N_8_i	LUT3	0	Out	1.992	5.457	
N_8_i	Net					12
ROM.dataaien0_2comp0tri0	BUFT	T	In		5.457	
ROM.dataaien0_2comp0tri0	BUFT	0	Out	1.111	6.567	
romdata[0]	Net					4
SPECIAL_REGS.inst_3_0_and4[0]	LUT4	I0	In		6.567	
SPECIAL_REGS.inst_3_0_and4[0]	LUT4	0	Out	1.347	7.914	
inst_3[0]	Net					2
SPECIAL_REGS.INST[0]	FDC	D	In		7.914	

You can also use the Log Watch window to quickly check parameters like Worst Slack, Requested Frequency, Estimated Frequency, and so on. To view the critical path graphically, use the procedure described in [Generate a Technology View for the Most Critical Path](#).

## Timing Information Display

To help you analyze timing, enable HDL Analyst -> Show Timing Information in a Technology view. This annotates all instances, showing their timing numbers. Two timing numbers are displayed above each instance:

---

Delay (first number)	For combinational logic, delay is the cumulative path delay to the output of the instance, which includes the net delay of the output. For flip-flops, delay is the portion of the path delay attributed to the flip-flop. The delay can be associated with either the input path or the output path, whichever is worse, because the flip-flop is the end of one path and the start of another.
Slack Time (second number)	This is the slack time of the worst path that goes through the instance. A negative value indicates that timing constraints could not be met. The command <b>Show Critical Path</b> uses the slack time, together with the slack margin, to determine which instances to display.

---


## Critical Path Analysis in a Technology View

This section describes the critical path views and how to analyze critical paths:

- [Critical Path Views](#)
- [Generate a Technology View for the Most Critical Path](#)
- [Generate Critical Path View in the Timing Analyzer](#)
- [Generate Critical Path View from the Log File](#)
- [Critical Path Report](#)

### Critical Path Views

The HDL Analyst tool makes it simple to find and examine critical paths and the relevant source code. Display the most critical path in the Technology view, 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. The view displayed is a filtered view in the same window, with hierarchical logic shown in transparent instances.
- 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 the top-level design.
- To generate a flattened critical path in a new window, select HDL Analyst -> Technology -> Flattened Critical Path. This 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 return to the previous window.

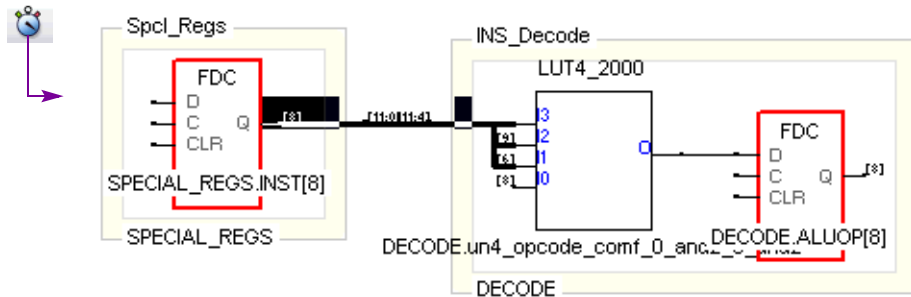
## Generate a Technology View for the Most Critical Path

This is the easiest way to generate a technology view for the most critical path:

1. Generate a Technology view.
2. Select the Show Critical Path icon from the toolbar.

This filters the critical path and displays timing numbers above each element. The critical path is the path with the worst (most negative, or smallest positive) slack of all your clock domains. If you have three clock domains and the worst clock domain has a slack of -2.4ns, this is the path that is displayed in the Technology view. The timing numbers above each element are the cumulative delay through that point, including estimated routing delay based on fanout.



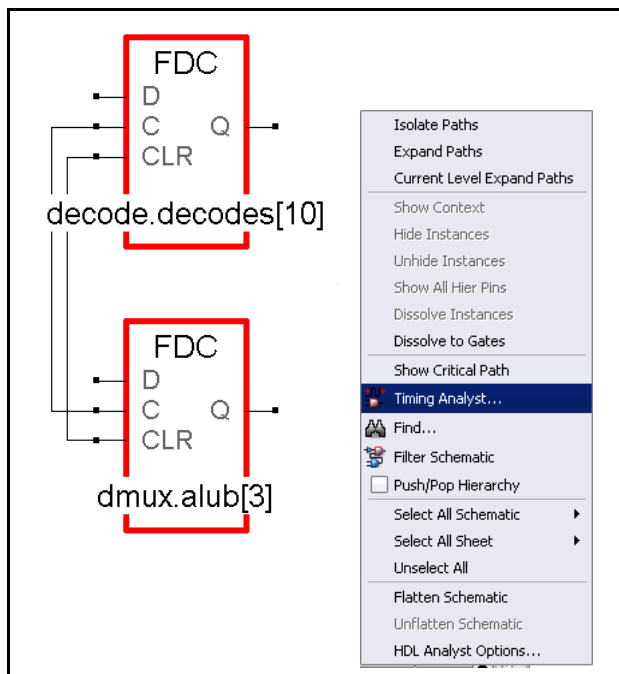


3. Analyze the critical path.

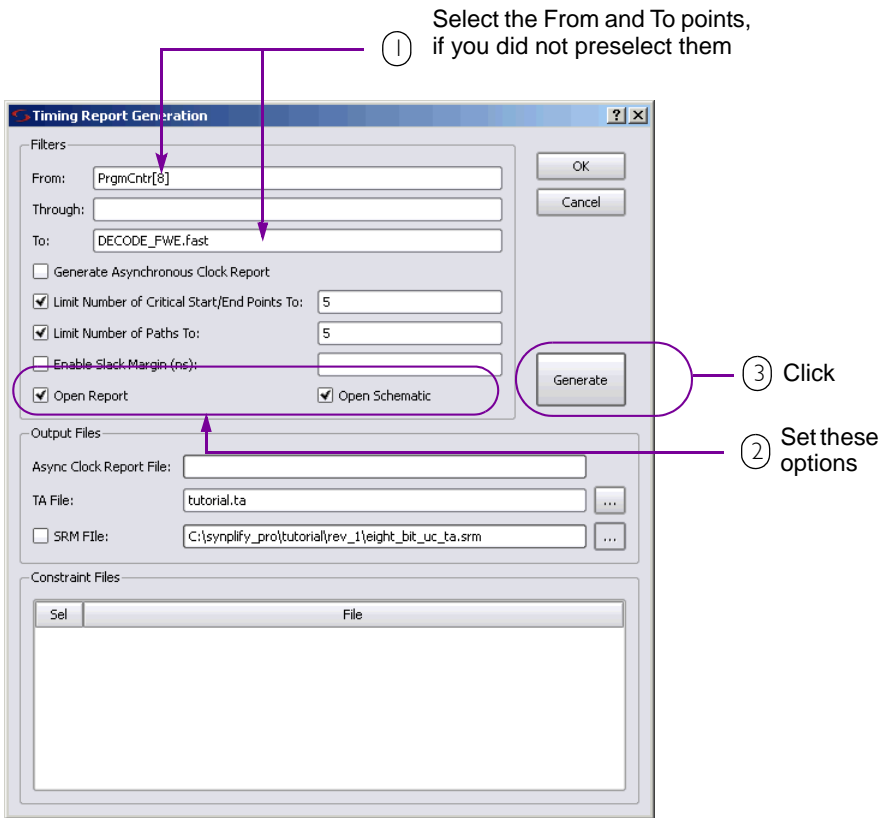
## Generate Critical Path View in the Timing Analyzer

Use the Timing Analyzer to generate views for any critical path or for point-to-point analysis between registers without re-running synthesis.

Select the instances in a Technology view, then right-click and select Timing Analyst from the pop-up menu.



Set the appropriate information in the Timing Report Generation dialog box. Buses are grouped to reduce the number of signals shown. Buses cannot be split into their individual elements.



The software generates a critical path view and a timing report like the one below:

```

Path information for path number 1:
  Requested Period:          40.000
  - Setup time:             0.518
  = Required time:          39.482

  - Propagation time:       12.974
  = Slack :                 26.508

  Starting point:           inst1.DATA0[6] / Q
  Ending point:             inst1.DATA0[7] / D
  The start point is clocked by prep2_2|CLK [rising] on pin C
  The end point is clocked by prep2_2|CLK [rising] on pin C

```

Instance / Net Name	Type	Pin Name	Pin Dir	Delay	Arrival Time	Fan Out
inst1.DATA0[6]	FDC	Q	Out	2.712	2.712	
DATA0_internal[6]	Net					3
inst1.compare_output_NE_4	LUT4	I1	In		2.712	
inst1.compare_output_NE_4	LUT4	O	Out	1.620	4.332	
compare_output_NE_4	Net					1
inst1.compare_output_NE	LUT4	I2	In		4.332	
inst1.compare_output_NE	LUT4	O	Out	3.273	7.605	
DATA0_lcry	Net					18
inst1.DATA0_6[0]	LUT4	I2	In		7.605	
inst1.DATA0_6[0]	LUT4	O	Out	1.620	9.225	
DATA0_6[0]	Net					1
inst1.DATA0_qxu[0]	LUT4	I1	In		9.225	
inst1.DATA0_qxu[0]	LUT4	O	Out	1.890	11.114	
DATA0_qxu[0]	Net					2
inst1.DATA0_cry[0]	MUXCY_L	S	In		11.114	
inst1.DATA0_cry[0]	MUXCY_L	LO	Out	0.899	12.013	

## Generate Critical Path View from the Log File

You can generate a view for any critical path listed in the log file with this method:

1. Open the RTL and Technology views. If you want to see details of the path, select HDL Analyst -> Technology -> Flattened View to open a flattened Technology view.
2. Select View Log and open the log file.
3. Scroll down to the critical path you want to view.
4. Hold down the Alt key and select the columns with the objects in the critical path.
5. Hold down the right mouse button and select Filter in Analyst. The objects in the critical path are highlighted in the log file and the critical path is selected in the open views.

Instance / Net Name	Type	Pin Name	Pin Dir	Delay
decode_decode_fast[12]	FDC	Q	Out	0.380
decode_fast[12]	Net	-	-	0.632
daux_alub_1_0[7]	LUT4	I0	In	-
daux_alub_1_0[7]	LUT4	0	Out	0.195
alub[7]	Net	-	-	0.704
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_cry_6	MULT_AND	I1	In	-
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_cry_6	MULT_AND	LO	Out	0.000
regsA_result_cry_6_7	Net	-	-	0.000
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_cry_6_0	MUXCY_L	DI	In	-
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_cry_6_0	MUXCY_L	I0	Out	0.690
regsA_result_cry_6	Net	-	-	0.000
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_s_7	XORCY	CI	In	-
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_s_7	XORCY	0	Out	0.452
uc_alu.U1.unl_a_AdderTree2_forloop2\..1..m1_forloop_1\..1..regsA_result_s_7	Net	-	-	0.610
uc_alu.U1.unl_a_AdderTree2_forloop2\..2..m1_forloop_1\..1..regsA_result_cry_6	MUXCY_L	I0	Out	0.690
regsA_result_cry_6_0	Net	-	-	0.000
uc_alu.U1.unl_a_AdderTree2_forloop2\..2..m1_forloop_1\..1..regsA_result_cry_7	MUXCY_L	CI	In	-
uc_alu.U1.unl_a_AdderTree2_forloop2\..2..m1_forloop_1\..1..regsA_result_cry_7	MUXCY_L	LO	Out	0.044
regsA_result_cry_7_0	Net	-	-	0.000
uc_alu.U1.unl_a_AdderTree2_forloop2\..2..m1_forloop_1\..1..regsA_result_cry_9	MUXCY_L	CI	In	-
uc_alu.U1.unl_a_AdderTree2_forloop2\..2..m1_forloop_1\..1..regsA_result_cry_9	MUXCY_L	I0	Out	0.044
regsA_result_cry_9_0	Net	-	-	0.000
uc_alu.U1.unl_a_AdderTree2_forloop2\..2..m1_forloop_1\..1..regsA_result_s_9	XORCY	CI	In	-

Undo

Redo

---

Cut

Copy

Paste

---

Select All

Toggle Bookmark

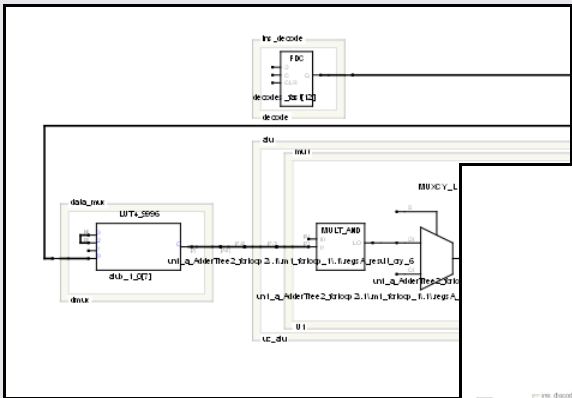
Next Bookmark

Previous Bookmark

**Filter in Analyst**

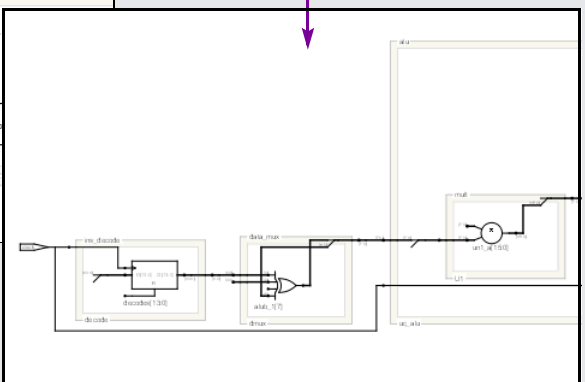
Select in Analyst

Select From...



Path in Technology

Path in RTL View





## CHAPTER 9

# Refine Options to Improve Timing

---

The vast majority of the time, you only need to identify the clocks and multi-cycle/false paths to get the best results.

- Occasionally, you might have to fine-tune the synthesis process (see [Refine Timing Results, on page 79](#)).
- You can also check your synthesis results against the place-and-route results and refine your synthesis constraints (see [Compare Synthesis Results with Place-and Route Results, on page 80](#)).
- In the Compile Point Synthesis flow, you can annotate incremental results after you resynthesize a compile point (see [Forward Annotate Incremental Results, on page 82](#)).

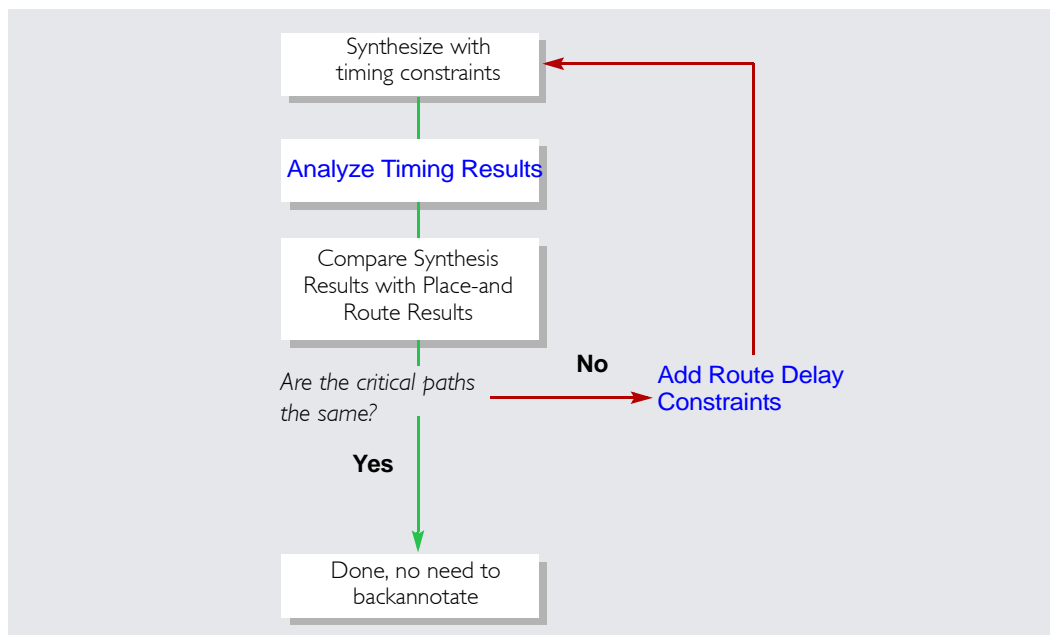
## Refine Timing Results

You can refine timing results in a few ways:

- Adjust the SCOPE timing constraints.
- Add attributes or directives, in the source code, in the SCOPE interface, or in a constraint file.
- Rework the RTL source code.

## Compare Synthesis Results with Place-and Route Results

Synplify Pro synthesis is timing-driven, so it is important to ensure that the place-and-route and Synplify Pro tools are both working on the same critical path. You only need to follow the tips here if the design *does not* meet timing. The flowchart shows that your goal is a critical path correlation between the place-and-route (P&R) and Synplify Pro results. Follow these steps to check results.



1. Place and route the design and run timing analysis.
2. Check the critical path and note its start and end points.
3. Compare the P&R start and end points to the start and end points of the path generated after synthesis.

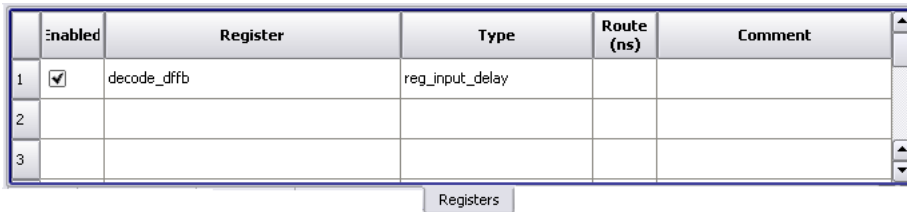
If P&R identifies a critical path from `alu.dffa` to `decode.dffb` while the Synplify Pro critical path goes from `spec_reg.dffr` to `dtc.dffc`, you need to make sure that Synplify Pro optimization routines are concentrated on the same path as the P&R tool. You can do this by adding a route delay constraint.



## Add Route Delay Constraints

The following example shows you how to add a route delay constraint.

1. Double-click the constraints file to open the SCOPE interface.
2. Go to the Registers pane, and add a small amount of incremental route delay to one of the registers on the critical path identified during P&R. The following figure shows the constraint applied to the end point in the previous example, decode.dffb.



	enabled	Register	Type	Route (ns)	Comment
1	<input checked="" type="checkbox"/>	decode_dffb	reg_input_delay		
2	<input type="checkbox"/>				
3	<input type="checkbox"/>				

Registers

Do not overload the path. Keep the value of this constraint low, just enough to raise the criticality of this path so that Synplify Pro optimization focuses on this path. You can then place and route the design again and check results. Typically this process is only necessary once, occasionally twice.

# Forward Annotate Incremental Results

In the Compile Point Synthesis flow, you can annotate incremental results after you resynthesize a compile point.

1. Resynthesize the design:
  - Make the changes you need to fix errors or improve your design. Define any required constraints and set the proper implementation options.
  - 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. The following figure illustrates this.

**First Run Log Summary**

```
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  -
=====
```

Syntax changes only; not resynthesized

Logic changes; compile point resynthesized

2. To force the software to generate a new model file for the compile point, select click Implementation Options 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.

3. 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.
4. Use the output files generated after synthesis to forward annotate incremental synthesis information to your place-and-route tool.



## CHAPTER 10

# Additional Features and Topics

---

This section includes additional information to accompany your design flows. Topics include:

- [Synplify Premier Physical Synthesis](#)
- [Synthesis Output Files](#)
- [Using Multiple Clock Domains](#)
- [Using Scripts and Batch Mode](#)
- [Using the Tcl Find Command for Setting Constraints](#)
- [Running Place and Route](#)
- [Using Identify with Synplify Pro](#)
- [For More Information](#)

## Synplify Premier Physical Synthesis

The Synplify Premier tool, as with all Synplicity synthesis products, provides the capability to compile, synthesize and optimize a design. However, physical synthesis also provides the benefit of physical placement that is recognized by the tool during optimization. With access to physical information, the tool can correlate between front- and back-end environments and provide more accurate estimates than can be achieved through using standard wire load model synthesis. With the Synplify Premier tool, you can floorplan, place, and optimize a design based on physical constraints as well as timing constraints, and physical and technology libraries. The output is a fully-optimized netlist with an architecture and locations best suited for the design based on the

floorplan, technology, and constraints. Analysis reports include a design summary, physical-based timing information, and congestion analysis. See Help in the tool or the section: [Synplify Premier Synthesis Design Flows, on page 2-35](#) of the *User Guide*.

## Synthesis Output Files

This section lists the output files that are written to the results specified directory for the implementation (Implementation Options->Implementation Results tab).

- `.areasrr`—Reports area-specific information on each module in the design, such as sequential and combinational ATOMS, RAMs, DSPs, and black boxes.
- `.edf`—Output EDIF netlist. See [Output Netlist](#), below, for more information.
- `.fse`—contains information about FSMs in the design.
- `.htm`—display report files in an HTML viewer that provides links for easier navigation to the various sections of the file.
- `.info`—design component files contain detailed information about components such as state machines or ROMs.
- `.ncf`—place-and-route constraints file.
- `pfl`—output file created when filtering messages in the Messages window.
- `.sap`—generated when you use the Annotated Properties for Analyst option and used by the Find command when searching for design properties.
- `.srd`—saves mapping information between synthesis runs; file is used internally by the tool.
- `.srm`—output by the mapper stage of the process, contains the actual technology-specific mapped design. This is the representation of the design displayed through the Technology view.
- `.srr`—synthesis log file that provides messages and information on the synthesis run as well as timing and area reports.

- `.srs`—output by the compiler stage of the process, contains the RTL-level (schematic) view of the design. This is the representation displayed in the RTL view.
- `.ta`—stand-alone timing report that contains the timing parameters specified when using the Analysis->Timing Analyst command. You can also graphically display the results of this report using the `ta.srm` file.
- `.tap`— this file is generated when the Annotated Properties for Analyst switch is enabled (Implementation Options->Options tab) and is used to annotate timing properties for the RTL view, Design Planner and the Find command.
- `ta.srm`—contains the graphical representation of the stand-alone timing report (`.ta`) that you can display in the Technology view.
- `.tasrr`—timing analysis log file, output only when you run the stand-alone timing report program.
- `.vif`— verification interface format file that contains Tcl commands to forward annotate sequential optimizations for formal verification.

## Output Netlist

The output netlist is usually written out as an EDIF file. However, the netlist can be written out in other formats appropriate to the technology and place-and-route tool that you are using, such as `.acf` or `.vqm` for Altera or the `.edf` format for Xilinx. Choose the format from the Implementation Options->Implementation Results tab (Result Format field).

To find information on the features specified in the sections below, you can use the online help system in the tool (Help-> Help, or F1), or the *User Guide* and *Reference Manual* (Help->Online Documents).

## Using Multiple Clock Domains

When using multiple clock domains in a design, you can define the relationship between clocks using the Clock Group parameter. By default, each clock is automatically assigned to a separate clock group (called `default_clkgroup<n>`). Clocks in different clock groups are treated by the timing analyzer as unrelated, meaning any paths between them are treated as false paths and

ignored during timing analysis. Clocks defined in the same clock group are assumed to be related and all timing paths between them are calculated during timing analysis. Also, by default, all inferred and other clocks that use the global frequency are assigned the same clock group. To group related clocks and separate unrelated clocks, use the SCOPE editor ->Clocks tab->Clock Group parameter.

See Help in the tool or the section: [Defining Other Clock Requirements, on page 5-225](#) of the *User Guide*.

## Using Scripts and Batch Mode

You can create scripts for running synthesis projects to run in batch mode. See Help in the tool or the sections: [Using Batch Mode, on page 21-876](#) and [Working with Tcl Scripts and Commands, on page 21-878](#) of the *User Guide*.

## Using the Tcl Find Command for Setting Constraints

Use the Tcl find command to search for design objects and properties, such as registers, clocks parameters, ports, and so on. You can group the search results into collections, and use these collections to apply constraints to the different design objects. See Help in the tool or the sections:

- [Tcl find Command, on page 14-1260](#) of the *Reference Manual*
- [Using Collections, on page 5-237](#) of the *User Guide*



## Running Place and Route

You can create a place-and-route implementation that will launch from within the tool or from batch mode following the synthesis run. See Help in the tool or the section: *Running Place-and-Route after Synthesis, on page 20-849* of the *User Guide*.

## Using Identify with Synplify Pro

The Identify RTL Debugger is a dual-component system that consists of an Identify Instrumentor that allows you to select a design instrumentation at the HDL level, then create an on-chip hardware probe, and the Identify Debugger tool that interacts with the on-chip hardware probe and from which you can interactively debug the design. The combination of these tools allows you to probe the HDL design in the target environment and debug the design faster, and more efficiently.

See Help in the tool or the section: *Working with the Identify RTL Debugger, on page 20-868* of the *User Guide*.

## For More Information

- The Synopsys FPGA synthesis documentation set is available in the tool through the integrated help system (Help->Help) and PDF documents accessible through Help->Online Documents.
- Synopsys Technical Support:  
<http://solvnet.synopsys.com>

