

Synopsys FPGA Synthesis User Guide

June 2009

<http://solvnet.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^{plus}, 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: Introduction

The Synopsys FPGA Product Family	22
The FPGA Synthesis Tools	22
Synopsys FPGA Tool Features	23
Scope of the Document	26
The Document Set	26
Audience	26
Getting Started	27
Starting the Software	27
Getting Help	27
Requesting Technical Support	28
User Interface Overview	29

Chapter 2: FPGA Logic and Physical Synthesis Flows

Logic Synthesis Design Flow	32
Synplify Premier Synthesis Design Flows	35
Logic Synthesis with Enhanced Optimization	36
Design Plan-Based Logic Synthesis	38
Graph-Based Physical Synthesis	42
Graph-Based Physical Synthesis with Design Planner	46
Design Plan-based Physical Synthesis	48
Actel Physical Synthesis	52
Set up the Actel Physical Synthesis Project	52
Run Logic Synthesis for the Actel Physical Synthesis Flow	56
Validate Logic Synthesis Results for Actel Physical Synthesis	56
Set up Actel Physical Constraints	57
Run Actel Physical Synthesis	57
Analyze Results of Actel Physical Synthesis	57

Altera Physical Synthesis	60
Guidelines for Physical Synthesis in Altera Designs	60
Set up the Altera Physical Synthesis Project	61
Run Logic Synthesis for the Altera Physical Synthesis Flow	65
Validate Logic Synthesis Results for Altera Physical Synthesis	66
Run Altera Physical Synthesis	66
Analyze Results of Altera Physical Synthesis	66
Xilinx Physical Synthesis	69
Set up the Xilinx Physical Synthesis Project	70
Run Logic Synthesis for the Xilinx Physical Synthesis Flow	74
Validate Logic Synthesis Results for Xilinx Physical Synthesis	75
Run Xilinx Physical Synthesis	75
Analyze Results of Xilinx Physical Synthesis	75
Guidelines for Xilinx Timing Constraints for Physical Synthesis	77
Using IP Cores in Xilinx Physical Synthesis Flows	78
Placement and Routing Phases in Xilinx Physical Synthesis	78
Prototyping Design Flow	80

Chapter 3: Preparing the Input

Setting Up HDL Source Files	82
Creating HDL Source Files	82
Checking HDL Source Files	83
Editing HDL Source Files with the Built-in Text Editor	85
Setting Editing Window Preferences	88
Using an External Text Editor	90
Using Hyper Source	91
Using Mixed Language Source Files	95
Working with Constraint Files	98
When to Use Constraint Files over Source Code	98
Tcl Syntax Guidelines for Constraint Files	99
Using a Text Editor for Constraint Files	100
Using Synopsys Design Compiler Constraints	102
Checking Constraint Files	104
Generating Constraint Files for Forward Annotation	105
Using Input from Related Tools	106
Converting Synopsys DesignWare Components	107
Converting Verilog Library Components	107
Converting VHDL Library Components	109

Chapter 4: Working with IP Input

Generating IP with SYNCORE	114
Specifying FIFOs with SYNCORE	114
Specifying RAMs with SYNCORE	119
Specifying ROMs with SYNCORE	125
Specifying Adder/Subtractors with SYNCORE	130
Specifying Counters with SYNCORE	137
The ReadyIP Encryption Flow	143
Overview of the Synopsys ReadyIP Flow	143
Encryption and Decryption	144
Working with Encrypted IP	148
Encrypting Your IP	148
Preparing the IP Package	153
Evaluating Vendor IP	158
Working with Altera IP	161
Using Altera LPMs or Megafunctions in Synthesis	161
Implementing Megafunctions with Clearbox Models	165
Implementing Megafunctions with Grey Box Models	175
Including Altera MegaCore IP Using an IP Package	181
Including Altera Processor Cores Generated in SOPC Builder	186
Working with SOPC Builder Components	191
Working with Lattice IP	194
Working with Xilinx IP	195
Xilinx IP Cores	195
Including Xilinx Cores for Logic and Physical Synthesis	196
Including Xilinx EDK Cores	199
The Synplify-EDK Design Flow	199
Working with EDK Cores	204
Xilinx Hardware Development Flows	207

Chapter 5: Specifying Constraints

Using the SCOPE UI	212
Creating a Constraint File Using the SCOPE Window	212
Entering and Editing Constraints in the SCOPE Window	214
Setting SCOPE Display Preferences	217
Specifying Timing Constraints	219
Entering Default Constraints	219
Setting Clock and Path Constraints	220

Defining Clocks	222
Defining Input and Output Constraints	226
Specifying Standard I/O Pad Types	228
Specifying Xilinx Timing Constraints	228
Using -route for Physical Synthesis in Xilinx Designs	230
Specifying Timing Exceptions	230
Defining From/To/Through Points for Timing Exceptions	231
Defining Multi-cycle Paths	234
Defining False Paths	235
Using Collections	237
Comparing Methods for Defining Collections	237
Creating and Using Collections (SCOPE Window)	238
Creating Collections (Tcl Commands)	241
Using the Tcl Find Command to Define Collections	244
Using the Expand Tcl Command to Define Collections	246
Viewing and Manipulating Collections (Tcl Commands)	247
Using Auto Constraints	251
Translating Altera QSF Constraints	253
Converting and Using Xilinx UCF Constraints	255
Using Xilinx UCF Constraints in a Logic Synthesis Design	255
Using Xilinx UCF Constraints in a Physical Synthesis Design	258
Support for UCF Conversion	260
Using the Legacy UCF2SDC Utility	264

Chapter 6: Setting up a Logic Synthesis Project

Setting Up Project Files	270
Creating a Project File	270
Opening an Existing Project File	273
Making Changes to a Project	274
Setting Project View Display Preferences	275
Updating Verilog Include Paths in Older Project Files	277
Project File Hierarchy Management	279
Creating Custom Folders	279
Other Custom Folder Operations	282
Other Custom File Operations	283
Setting Up Implementations and Workspaces	285
Working with Multiple Implementations	285
Creating Workspaces	287
Using Workspaces	288

Setting Logic Synthesis Implementation Options	289
Setting Device Options	289
Setting Optimization Options	292
Specifying Global Frequency and Constraint Files	294
Specifying Result Options	296
Specifying Timing Report Output	297
Setting Verilog and VHDL Options	298
Entering Attributes and Directives	304
Specifying Attributes and Directives	304
Specifying Attributes and Directives in VHDL	305
Specifying Attributes and Directives in Verilog	307
Specifying Attributes Using the SCOPE Editor	308
Specifying Attributes in the Constraints File (.sdc)	310
Searching Files	312
Identifying the Files to Search	313
Filtering the Files to Search	313
Initiating the Search	314
Search Results	314
Archiving Files and Projects	315
Archive a Project	315
Un-Archive a Project	320
Copy a Project	323
Chapter 7: Setting up a Physical Synthesis Project	
Setting up for Physical Synthesis	328
Setting Options for Physical Synthesis	330
Setting Synplify Premier Netlist Restructuring Optimizations	330
Creating a Place and Route Implementation	332
Specifying Altera Place-and-Route Options	337
Specifying Xilinx Place-and-Route Options in a Tcl File	340
Specifying Xilinx Place-and-Route Options in an .opt File	341
Specifying Xilinx Global Placement Options	346
Setting Constraints for Physical Synthesis	347
Using Design Planner Floorplan Constraints	347
Translating Pin Location Files	348
Translating Actel I/O Constraints	348
Setting Physical Synthesis Constraints for Altera	349
Forward-Annotating Physical Synthesis Constraints	351
Forward Annotating Altera Physical Constraints	351

Backannotating Physical Synthesis Constraints	353
Backannotating Place-and-Route Data	353
Generating a Xilinx Coreloc Placement File	354

Chapter 8: Inferring High-Level Objects

Defining Black Boxes for Synthesis	358
Instantiating Black Boxes and I/Os in Verilog	358
Instantiating Black Boxes and I/Os in VHDL	360
Adding Black Box Timing Constraints	362
Adding Other Black Box Attributes	366
Defining State Machines for Synthesis	367
Defining State Machines in Verilog	368
Defining State Machines in VHDL	369
Specifying FSMs with Attributes and Directives	369
Inferring RAMs	372
Inference Versus Instantiation	372
Basic Guidelines for Coding RAMs	373
Specifying RAM Implementation Styles	378
Implementing Altera RAMs Automatically	379
Implementing Xilinx RAMs Automatically	383
Implementing Altera FLEX and APEX RAMs	385
Implementing Altera Stratix Multi-Port RAMs	388
Inferring Altera Stratix III LUTRAMs	389
Inferring Xilinx Block RAMs Using Registered Addresses	391
Inferring Xilinx Block RAMs Using Registered Output	393
Mapping Xilinx ROM to Block RAM	398
Initializing RAMs	400
Initializing RAMs in Verilog	400
Initializing RAMs in VHDL	401
Initializing Xilinx RAM	404
Inferring Shift Registers	410
Working with LPMs	416
Instantiating Altera LPMs as Black Boxes	417
Instantiating Altera LPMs Using VHDL Prepared Components	421
Instantiating Altera LPMs Using a Verilog Library	423

Chapter 9: Specifying Design-Level Optimizations

Tips for Optimization	426
General Optimization Tips	426
Optimizing for Area	427
Optimizing for Timing	428
Pipelining	429
Prerequisites for Pipelining	429
Pipelining the Design	430
Retiming	433
Controlling Retiming	433
Retiming Example	435
Retiming Report	436
How Retiming Works	437
How Retiming Works With Synplify Premier Regions	439
Preserving Objects from Optimization	440
Using <code>syn_keep</code> for Preservation or Replication	441
Controlling Hierarchy Flattening	444
Preserving Hierarchy	444
Optimizing Fanout	446
Setting Fanout Limits	446
Controlling Buffering and Replication	448
Sharing Resources	450
Inserting I/Os	452
Optimizing State Machines	453
Deciding when to Optimize State Machines	453
Running the FSM Compiler	455
Running the FSM Explorer	458
Inserting Probes	461
Specifying Probes in the Source Code	461
Adding Probe Attributes Interactively	462
Working with Gated Clocks	464
Gated Clocks in Synopsys FPGA Designs	464
Prerequisites for Gated Clock Conversion	467
Synthesizing a Gated Clock Design	469
Using Gated Clocks for Black Boxes	471
Analyzing Gated Clock Conversion Reports	472
Working with Gated Clock Error Messages	473
Restrictions on Using Gated Clocks	475

Optimizing Generated Clocks	477
Generated-Clock Optimization Example	477
Enabling Generated-Clock Optimization	478
Conditions for Generated-Clock Optimization	479

Chapter 10: Fast Synthesis

About Fast Synthesis	482
Using Fast Synthesis	483
Fast Synthesis and Other Synthesis Options	485

Chapter 11: Floorplanning with Design Planner

Using Design Planner	488
Starting Design Planner	488
Copying Objects in the Design Planner Tool	490
Controlling Pin Display in the Design Plan Editor	491
Creating and Using a Design Plan File for Physical Synthesis	494
Assigning Pins and Clocks	495
Assigning Pins Interactively	495
Importing Pin Assignments from Pin Assignment Files	498
Assigning Clock Pins	498
Modifying Pin Assignments	500
Using Temporary Pin Assignments	501
Viewing Assigned Pins in Different Views	502
Viewing Pin Assignment Information	503
Working with Regions	505
Creating Regions	505
Using Region Tunneling	507
Moving and Sizing Regions	509
Viewing Intellectual Property (IP) Core Areas	510
Assigning Logic to Top-level Chip Regions	510
Assigning Logic to Regions	514
Replicating Logic Manually	515
Assigning Critical Paths from Island Timing to a Region	516
Checking Utilization	517
Working with Altera Regions	519
Creating Design Planner Regions for Altera Designs	520
Assigning Logic to Altera Design Planner Regions	521
Working with Xilinx Regions	523
Creating Regions for Xilinx Designs	525

Assigning Objects to Xilinx Regions	527
Assigning Xilinx Critical Paths to Design Planner Regions	527
Assigning Xilinx Block RAMs to Regions	534
Assigning Xilinx Block Multipliers to Regions	539
Assigning Xilinx DSP Blocks to Regions	540
Using Process-Level Hierarchy	542
Bit Slicing	543
Using Bit Slicing	543
Bit Slice Examples	547
Zippering	550
Zippering Guidelines	551
Using Zippering	551
Zippering Example	555

Chapter 12: Running Logical Compile Points

Logical Compile-Point Synthesis	560
Overview	560
Traditional Bottom-up Design and Compile Points	561
About Compile Points	562
Nesting: Child and Parent Compile Points	562
Advantages of Using Compile Points	563
Compile Point Types	564
Compile Point Feature Summary	567
Using <code>syn_hier</code> with Compile Points	568
Using <code>syn_allowed_resources</code> with Compile Points	568
<code>define_compile_point</code> and <code>define_current_design</code>	569
About Interface Logic Models (ILMs)	570
Compile Point Synthesis	571
Compile Point Optimization	571
Forward-annotation of Compile-point Timing Constraints	572
Using Compile-point Synthesis	573
Synplify Pro and Synplify Premier Compile-point Flow	573
Xilinx Compile-point Synthesis Flow	583
Using Xilinx Compile-point Synthesis	583

Chapter 13: Using Multiprocessing

Multiprocessing With Compile Points	588
Setting Maximum Parallel Jobs	588
License Utilization	589

Chapter 14: Synthesizing and Analyzing the Log File

Synthesizing Your Design	592
Running Logic Synthesis	592
Running Physical Synthesis	592
Checking Log Results	596
Viewing the Log File	596
Analyzing Results Using the Log File Reports	599
Using the Log Watch Window	600
Handling Messages	602
Checking Results in the Message Viewer	602
Filtering Messages in the Message Viewer	604
Filtering Messages from the Command Line	606
Automating Message Filtering with a Tcl Script	607
Handling Warnings	609
Validating Logic Synthesis for Physical Synthesis	609

Chapter 15: Analyzing with HDL Analyst and FSM Viewer

Working in the Schematic Views	614
Differentiating Between the Views	615
Opening the Views	615
Viewing Object Properties	617
Selecting Objects in the RTL/Technology Views	620
Working with Multisheet Schematics	621
Moving Between Views in a Schematic Window	623
Setting Schematic View Preferences	623
Managing Windows	625
Exploring Design Hierarchy	627
Traversing Design Hierarchy with the Hierarchy Browser	627
Exploring Object Hierarchy by Pushing/Popping	628
Exploring Object Hierarchy of Transparent Instances	634
Finding Objects	635
Browsing to Find Objects	635
Using Find for Hierarchical and Restricted Searches	637
Using Wildcards with the Find Command	640
Using Find to Search the Output Netlist	643
Crossprobing	645
Crossprobing within an RTL/Technology View	646
Crossprobing from the RTL/Technology View	647
Crossprobing from the Text Editor Window	649

Crossprobing from the Tcl Script Window	652
Crossprobing from the FSM Viewer	652
Analyzing With the HDL Analyst Tool	654
Viewing Design Hierarchy and Context	655
Filtering Schematics	658
Expanding Pin and Net Logic	660
Expanding and Viewing Connections	664
Flattening Schematic Hierarchy	665
Minimizing Memory Usage While Analyzing Designs	670
Using the FSM Viewer	670
 Chapter 16: Analyzing Designs in Physical Analyst	
Analyzing Physical Synthesis Results	678
Analyzing Physical Synthesis Results Using Various Tools	678
Comparing Performance Results	680
Running Multiple Implementations	681
Checking Altera Pre-Placement Physical Synthesis Results	681
Using Physical Analyst	683
Opening the Physical Analyst Interface	683
Zooming in the Physical Analyst	685
Moving Between Views in the Physical Analyst	686
Using the Physical Analyst Context Window	687
Displaying and Selecting Objects	689
Setting Visibility for Physical Analyst Objects	689
Displaying Instances and Sites in Physical Analyst	690
Displaying Nets in Physical Analyst	694
Selecting Objects in Physical Analyst	696
Querying Physical Analyst Objects	699
Viewing Properties in Physical Analyst	699
Using Tool Tips to View Properties in Physical Analyst	702
Finding Objects	704
Using Find to Locate Physical Analyst Objects)	704
Finding Physical Analyst Objects by Their Locations	708
Using Markers to Find Physical Analyst Objects	709
Identifying Encrypted IP Objects in Physical Analyst	711
Crossprobing in Physical Analyst	713
Crossprobing from the Physical Analyst View	713
Crossprobing from a Text File to Physical Analyst	716

Crossprobing from the RTL View to Physical Analyst	717
Crossprobing from the Technology View to Physical Analyst	719
Analyzing Netlists in Physical Analyst	721
Filtering the Physical Analyst View	721
Expanding Pin and Net Logic in Physical Analyst	722
Expanding and Viewing Connections in Physical Analyst	727

Chapter 17: Analyzing Timing

Analyzing Timing in Schematic Views	730
Viewing Timing Information	730
Annotating Timing Information in the Schematic Views	731
Analyzing Clock Trees in the RTL View	733
Viewing Critical Paths	733
Using the Stand-alone Timing Analyst	736
Entering Constraints into the .adc File	741
Using the Island Timing Analyst	743
Working in the Island Timing Analyst Interface	743
Generating the Island Timing Report Automatically	745
Generating the Island Timing Report Interactively	747
Defining Group Range and Global Range for Island Timing	748
Viewing the Island Timing Report	749
Analyzing Timing with Physical Analyst	750
Viewing Critical Paths in Physical Analyst	750
Tracing Critical Paths Forward in Physical Analyst	753
Tracing Critical Paths Backward in Physical Analyst	755
Handling Negative Slack	756

Chapter 18: Optimizing for Specific Targets

Optimizing Actel Designs	760
Using Predefined Actel Black Boxes	760
Using ACTGen Macros	761
Working with Radhard Designs	762
Improving Performance in Actel Physical Synthesis Designs	763
Optimizing Altera Designs	764
Design Tips for APEX and FLEX Designs	765
Determining ROM Implementation	765
Working with Altera EABs and ESBs	767
Working with Altera PLLs	769
Instantiating Special Buffers as Black Boxes in Altera Designs	770

Specifying Altera I/O Locations	772
Packing I/O Cell Registers in Altera Designs	774
Specifying HardCopy and Stratix Companion Parts	775
Specifying Core Voltage in Stratix III Designs	776
Using LPMs in Simulation Flows	777
Improving Altera Physical Synthesis Performance	779
Working with Quartus II	779
Configuring Max+Plus II for FLEX and ACEX1K	781
Configuring Max+Plus II for MAX Designs	784
Optimizing Lattice Designs	785
Instantiating Lattice Macros	785
Using Lattice GSR Resources	787
Inferring Carry Chains in Lattice XPLD Devices	788
Inferring Lattice PIC Latches	788
Controlling I/O Insertion in Lattice Designs	794
Forward-Annotating Lattice Constraints	795
Optimizing Xilinx Designs	797
Designing for Xilinx Architectures	797
Specifying Xilinx Macros	798
Specifying Global Sets/Resets and Startup Blocks	800
Inferring Wide Adders	801
Instantiating CoreGen Cores	804
Instantiating Virtex PCI Cores	805
Packing Registers for Xilinx I/Os	807
Specifying Xilinx Register INIT Values	810
Inserting Xilinx I/Os and Specifying Pin Locations	812
Working with Xilinx Buffers	818
Specifying RLOCs	819
Specifying RLOCs and RLOC_ORIGINS with the synthesis Attribute	821
Using Clock Buffers in Virtex Designs	822
Working with Clock Skews in Xilinx Virtex-5 Physical Designs	824
Instantiating Special I/O Standard Buffers for Virtex	825
Reoptimizing With EDIF Files	826
Improving Xilinx Physical Synthesis Performance	827
Running Post-Synthesis Simulation	828
Working with Xilinx Place-and-Route Software	829

Chapter 19: Working with Synthesis Output

Passing Information to the P&R Tools	832
Specifying Pin Locations	832
Specifying Locations for Actel Bus Ports	833

Specifying Macro and Register Placement	833
Passing Technology Properties	834
Specifying Padtype and Port Information	834
Generating Vendor-Specific Output	836
Targeting Output to Your Vendor	836
Customizing Netlist Formats	837
Invoking Third-Party Vendor Tools	838
Configuring Tool Tags	838
Invoking a Third-Party Tool	839

Chapter 20: Running Post-Synthesis Operations

VIF Formal Verification Flow	844
Overview of the VIF Flow	844
Generating a VIF File	845
Using a Tcl Script for VIF Conversion	847
Handling Equivalency Check Failures	848
Running Place-and-Route after Synthesis	849
Simulating with the VCS Tool	851
Resynthesizing with QuickLogic Information	856
Quartus II Incremental Compilation	857
Quartus II Incremental Compilation Flow	857
Working with Xilinx Incremental Flows	862
Incremental Flow for Xilinx Designs	862
SmartGuide Global Placement Flow	863
Partition Flow	863
Working with the Identify RTL Debugger	868
Launching from the Synplify Pro or Synplify Premier Tool	868
Launching from the Synplify Tool	870
Handling Problems with Launching Identify	872
Using the Identify Tool	873

Chapter 21: Process Optimization and Automation

Using Batch Mode	876
Running Batch Mode on a Project File	876
Running Batch Mode with a Tcl Script	877
Working with Tcl Scripts and Commands	878
Using Tcl Commands and Scripts	878

Generating a Job Script	879
Creating a Tcl Synthesis Script	879
Using Tcl Variables to Try Different Clock Frequencies	881
Using Tcl Variables to Try Several Target Technologies	882
Running Bottom-up Synthesis with a Script	883
Automating Flows with synhooks.tcl	884

CHAPTER 1

Introduction

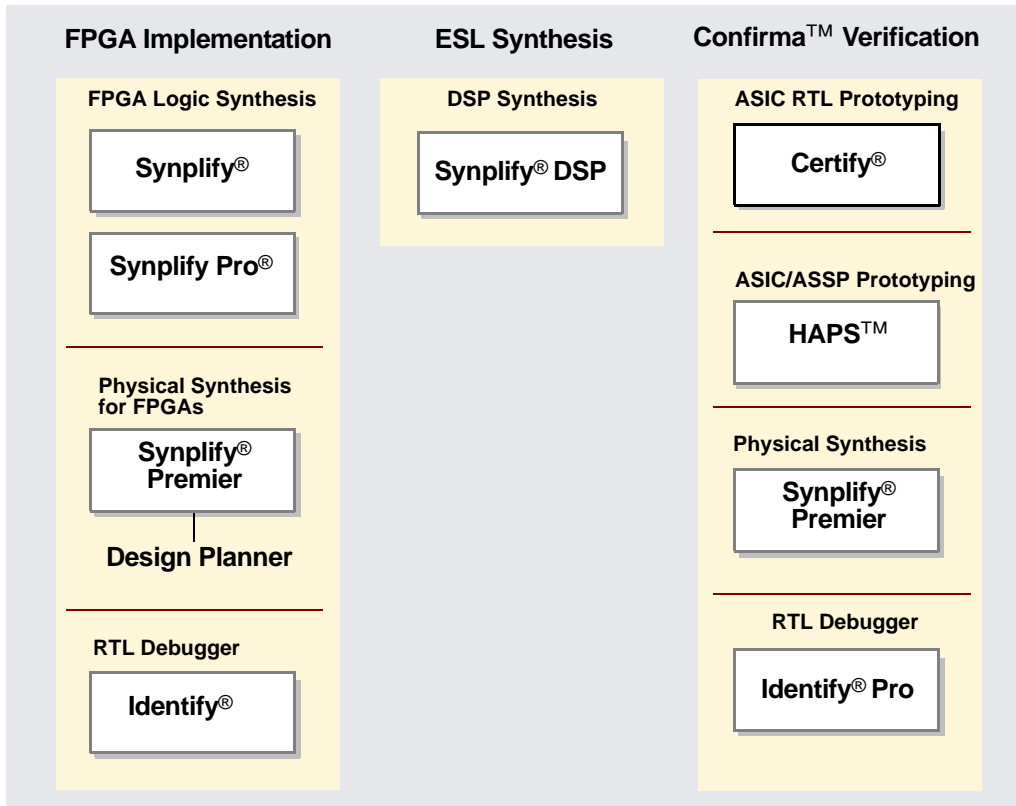
This introduction to the Synplify®, Synplify Pro®, and Synplify® Premier software describes the following:

- [The Synopsys FPGA Product Family](#), on page 22
- [Scope of the Document](#), on page 26
- [Getting Started](#), on page 27
- [User Interface Overview](#), on page 29

Throughout the documentation, features and procedures described apply to all tools, unless specifically stated otherwise.

The Synopsys FPGA Product Family

The Synopsys® family of synthesis tools is based on core logic synthesis technology, and share a common look and feel. The following figure shows the Synopsys® Synplicity® line of products.



The FPGA Synthesis Tools

This section briefly describes the FPGA synthesis tools Synplify, Synplify Pro, and Synplify Premier synthesis tools.

Synplify and Synplify Pro Software

Synplify® and Synplify Pro® are logic synthesis tools for FPGAs (Field Programmable Gate Arrays) and Complex PLDs (Programmable Logic Devices). For input, the software uses high-level designs written in Verilog and VHDL hardware description languages (HDLs). Using proprietary Behavior Extracting Synthesis Technology® (B.E.S.T.)® the tool converts the HDL into small, high-performance, design netlists that are optimized for popular technology vendors. Optionally, the software can write post-synthesis VHDL and Verilog netlists that you can use to verify functionality through simulation.

The Synplify Pro software offers a superset of the Synplify features.

Synplify Premier Software

The Synplify Premier tool offers a push-button, graph-based physical synthesis approach improving overall device performance while simultaneously delivering tight correlation between pre-route timing estimates and final post place-and-route results.

The Synplify Premier product supports three physical design flows. See [Synplify Premier Synthesis Design Flows, on page 35](#) for descriptions. You can also use it in the prototyping flow, described in [Prototyping Design Flow, on page 80](#).

Synopsys FPGA Tool Features

This table distinguishes between the Synplify Pro, Synplify, Synplify Premier, and Synplify Premier with Design Planner products.

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Performance				
Behavior Extracting Synthesis Technology® (BEST™)	x	x	x	x
Vendor-Generated Core/IP Support (certain technologies)		x	x	x
FSM Compiler	x	x	x	x

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
FSM Explorer		x	x	x
Gated Clock Conversion		x	x	x
Register Pipelining		x	x	x
Register Retiming		x	x	x
Code Analysis				
SCOPE® Spreadsheet	x	x	x	x
HDL Analyst®	Option	x	x	x
Timing Analyzer – Point-to-point		x	x	x
FSM Viewer		x	x	x
Crossprobing		x	x	x
Probe Point Creation		x	x	x
Physical Design				
Design Plan File				x
Logic Assignment to Regions				x
Area Estimation and Region Capacity				x
Pin Assignment				x
Physical Synthesis Optimizations				x
Graph-based Physical Synthesis			x	x
Island Timing Analyst			x	x
Physical Analyst			x	x
Prototyping				
Automatic translation of Synopsys® DesignWare® components			x	x

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Team Design				
Mixed Language Design		x	x	x
Modular Flow (certain technologies)		x	x	x
Compile Points		x	x	x
True Batch Mode (Floating licenses only)		x	x	x
GUI Batch Mode (Floating licenses)	x	x	x	x
Batch Mode Post-synthesis P&R Run	-	x	x	x
Back-annotation of P&R Data	-	-	-	x
Formal Verification Flow		x	x (Physical synthesis disabled)	x (Physical synthesis disabled)
Identify Integration	Limited	x	x	x
Back-annotation of P&R Data				x
Design Environment				
Technical Resource Center	x	x	x	x
Text Editor View	x	x	x	x
Log Watch Window		x	x	x
Message Window		x	x	x
Tcl Window		x	x	x
Workspaces		x	x	x
Multiple Implementations		x	x	x
Vendor Technology/Family Support	x	x	Limited	Limited

Scope of the Document

The following explain the scope of this document and the intended audience.

The Document Set

This user guide is part of a document set that includes a reference manual and a tutorial. It is intended for use with the other documents in the set. It concentrates on describing how to use the Synplify software to accomplish typical tasks. This implies the following:

- The user guide only explains the options needed to do the typical tasks described in the manual. It does not describe every available command and option. For complete descriptions of all the command options and syntax, refer to the [User Interface Overview](#) chapter in the *Synopsys FPGA Synthesis Reference Manual*.
- The user guide contains task-based information. For a breakdown of how information is organized, see [Getting Help, on page 27](#).

Audience

The Synplify, Synplify Pro, and Synplify Premier software tools are targeted towards the FPGA system developer. It is assumed that you are knowledgeable about the following:

- Design synthesis
- RTL
- FPGAs
- Verilog/VHDL
- Physical Synthesis

Getting Started

This section shows you how to get started with the Synplify software. It describes the following topics, but does not supersede the information in the installation instructions about licensing and installation:

- [Starting the Software](#), on page 27
- [Getting Help](#), on page 27
- [Requesting Technical Support](#), on page 28

Starting the Software

1. If you have not already done so, install the Synplify software according to the installation instructions.
2. Start the software.
 - If you are working on a Windows platform, select Programs->Synplicity->*product version* from the Start button.
 - If you are working on a UNIX platform, type the appropriate command at the command line:

```
synplify
synplify_pro
synplify_premier
synplify_premier_dp
```

The command starts the synthesis tool, and opens the Project window. If you have run the software before, the window displays the previous project. For more information about the interface, see the [User Interface Overview](#) chapter of the *Reference Manual*.

Getting Help

Before you call Synopsys Support, look through the documented information. You can access the information online from the Help menu, or refer to the PDF version. The following table shows you how the information is organized.

For help with...	Refer to the...
Using software features	<i>Synopsys FPGA Synthesis User Guide</i>
How to...	<i>Synopsys FPGA Synthesis User Guide</i> , application notes on the support web site
Flow information	<i>Synopsys FPGA Synthesis User Guide</i> , application notes on the support web site
Error messages	Online help (select Help->Error Messages)
Licensing	Synopsys FPGA Licensing Document (PDF)
Attributes and directives	<i>Synopsys FPGA Synthesis Reference Manual</i>
Synthesis features	<i>Synopsys FPGA Synthesis Reference Manual</i>
Language and syntax	<i>Synopsys FPGA Synthesis Reference Manual</i>
Tcl syntax	Online help (select Help->Tcl Help)
Tcl synthesis commands	<i>Synopsys FPGA Synthesis Reference Manual</i>
Product updates	<i>Synopsys FPGA Synthesis Reference Manual</i> (Web menu commands)

Requesting Technical Support

To request assistance from Technical Support for the Synopsys FPGA synthesis products, use the SolvNet Online Support utility – an online web-based interface from which you can submit your request form and attach project files. This is the preferred mechanism for contacting Technical Support and may facilitate a faster response than requesting support through email.

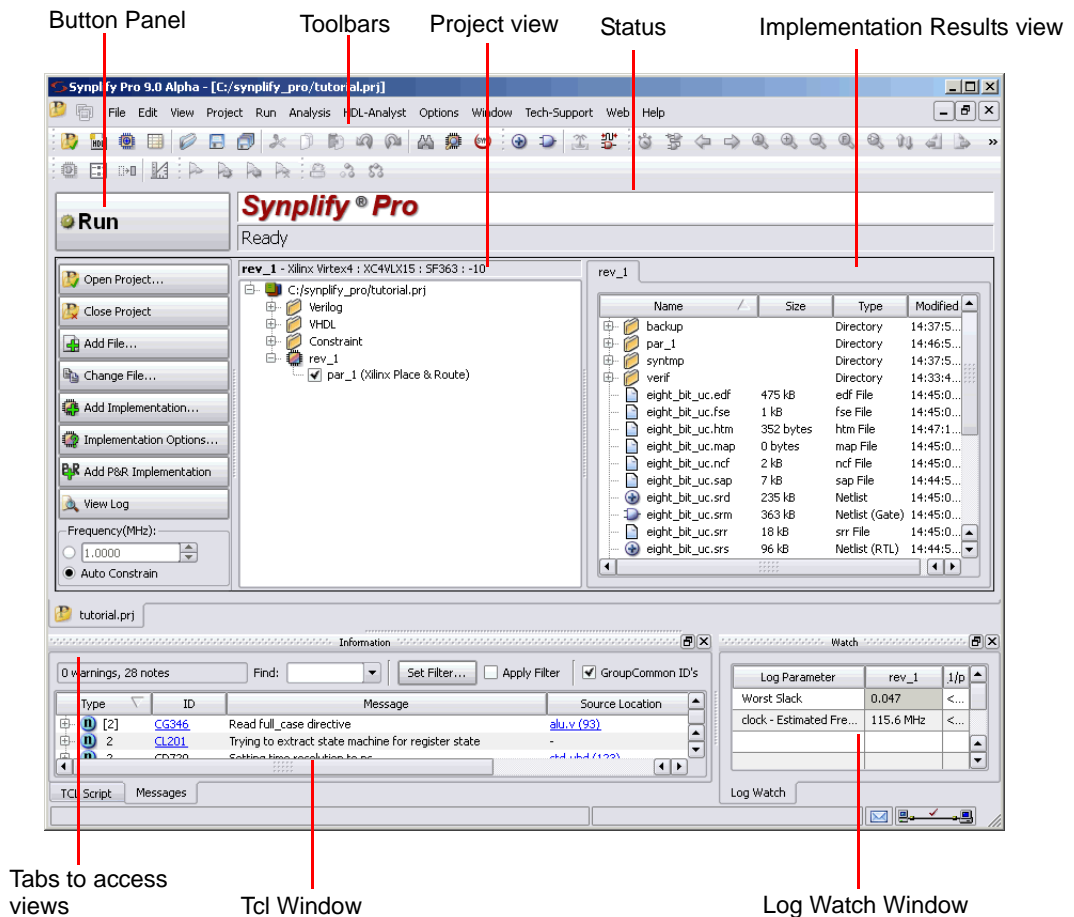
You can access SolvNet Online Support in one of these ways:

- From the tool: Tech-Support->Submit Support Request. This opens a wizard that walks you through the process of making a request and attaching related files from your project.
- From the tool: Tech-Support->Web Support.
- Through the link on the web page:
<http://solvnet.synopsys.com>.

User Interface Overview

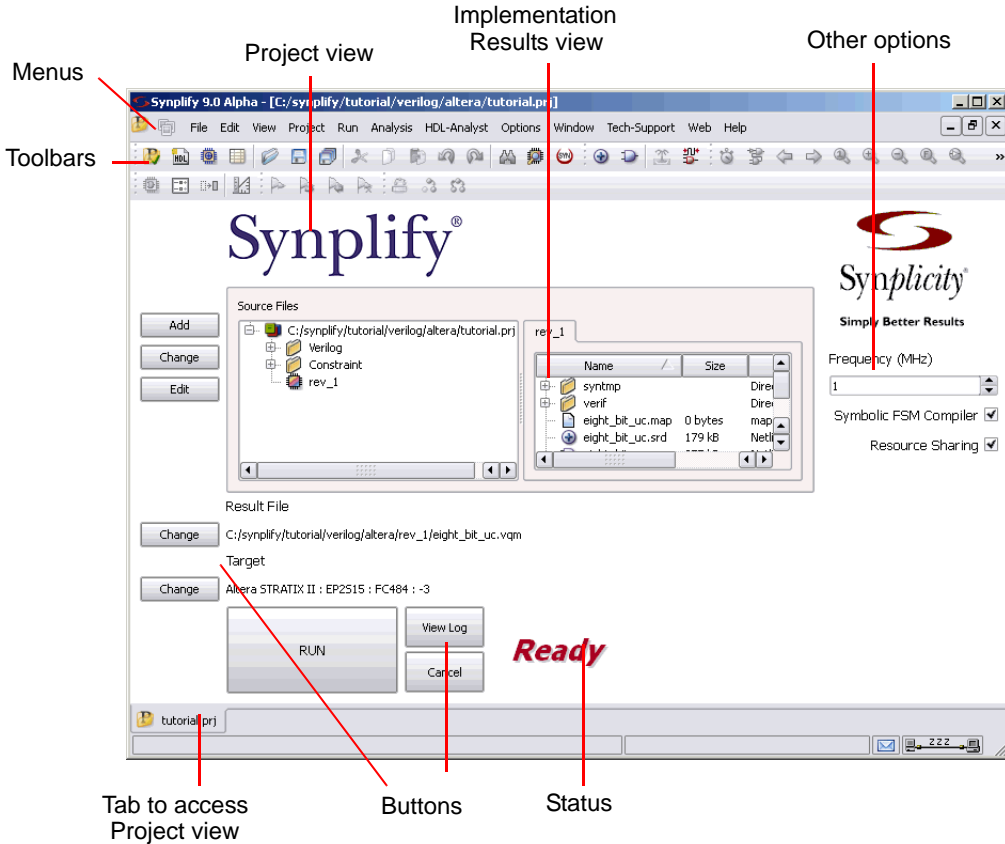
The user interface (UI) consists of a main window, called the Project view, and specialized windows or views for different tasks. For details about each of the features, see [Chapter 2, User Interface Overview](#) of the *Synopsys FPGA Synthesis Reference Manual*. The Synplify Pro and Synplify Premier tools have the same standard interface, while Synplify uses a different interface.

Synplify Pro and Synplify Premier Standard Interface



Synplify Interface

The following figure shows you the Synplify interface.



Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA
 Phone: +1 408 215-6000, Fax: +1 408 222-068
 www.solvnet.com

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

CHAPTER 2

FPGA Logic and Physical Synthesis Flows

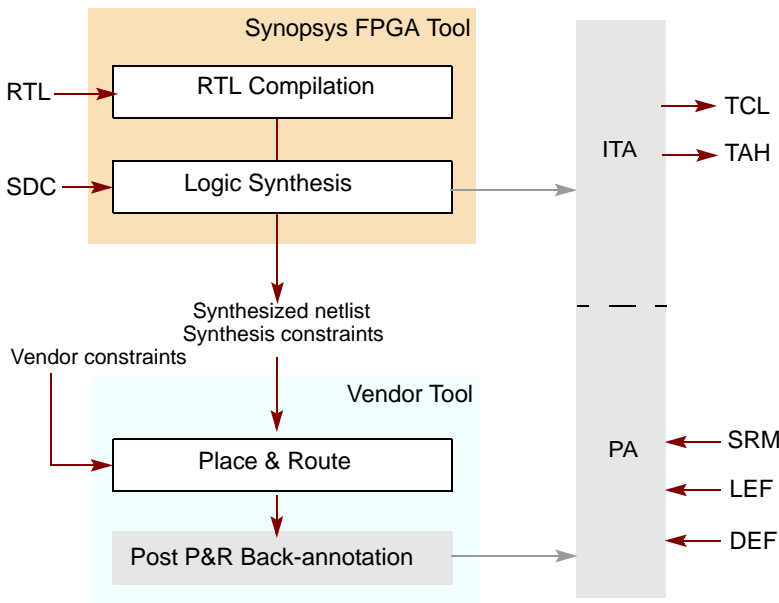
This describes the following tool flows:

- [Logic Synthesis Design Flow](#), on page 32
- [Synplify Premier Synthesis Design Flows](#), on page 35
- [Actel Physical Synthesis](#), on page 52
- [Altera Physical Synthesis](#), on page 60
- [Xilinx Physical Synthesis](#), on page 69
- [Prototyping Design Flow](#), on page 80

Logic Synthesis Design Flow

The Synopsys FPGA tools synthesize logic by first compiling the RTL source, and then doing logical mapping and optimizations. After logic synthesis, you get a vendor-specific netlist and constraint file that you 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. You can use the Synplify, Synplify Pro, or Synplify Premier synthesis software for this flow. The interactive timing analysis, physical analysis, and backannotation steps that are shown in gray are 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

For a design flow with step-by-step instructions based on specific design data, download the tutorial from the website. The following steps summarize the process, 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. If you are running Synplify Premier in logic synthesis mode, disable the Physical Synthesis option in the Project view.

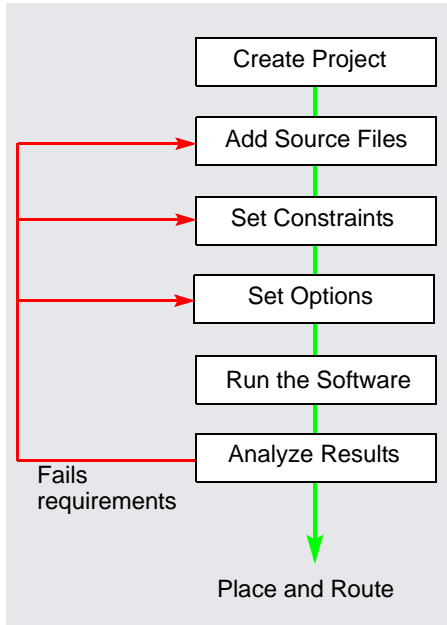
This setting directs the software to only run logic synthesis, without any physical optimizations. For other logic synthesis modes in the Synplify Premier tool, see [Fast Synthesis, on page 481](#) and [Logic Synthesis with Enhanced Optimization, on page 36](#).

6. Click Run to run logic synthesis.
7. Analyze the results, using the log file, the HDL Analyst schematic views, the Message window and the Log Watch window.

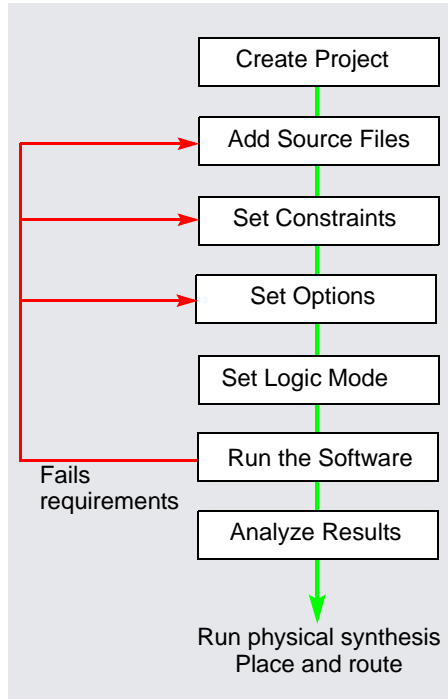
After you have completed the design, you can use the output files to run place-and-route with the vendor tool and implement the FPGA. If you are using the Synplify Premier software, you can choose to run physical synthesis before place-and-route.

The following figure lists the main steps in the flow:

SYNPLIFY & SYNPLIFY PRO



SYNPLIFY PREMIER



Synplify Premier Synthesis Design Flows

You use the Synplify Premier tool to perform logic synthesis as well as physical synthesis. The logic synthesis flows let you run logic synthesis as a separate step. The physical synthesis flows include logic synthesis.

The following table lists the Synplify Premier logical and physical synthesis flows. Some of these flows are only available in certain technologies.

Logic Synthesis Flows

Logic Synthesis Design Flow , on page 32	Same as Synplify Pro logic synthesis
Logic Synthesis with Fast Synthesis	Synplify Premier logic synthesis with fast synthesis runtimes. For details about this flow, see Fast Synthesis , on page 481.
Logic Synthesis with Enhanced Optimization , on page 36	Synplify Premier logic synthesis includes additional optimizations during logic synthesis and provides an output netlist with better QoR than when running basic logic synthesis. Enhanced Optimization is turned on by default.
Design Plan-Based Logic Synthesis , on page 38	Synplify Premier logic synthesis with placement constraints from a floorplan file (needs Design Planner option)

Physical Synthesis Flows

Graph-Based Physical Synthesis , on page 42	Automated physical synthesis. This includes the enhanced logic synthesis optimizations.
Graph-Based Physical Synthesis with Design Planner , on page 46	Automated physical synthesis that uses floorplan file constraints (needs Design Planner option)
Design Plan-based Physical Synthesis , on page 48	Physical synthesis with placement constraints from a floorplan file (needs Design Planner option)

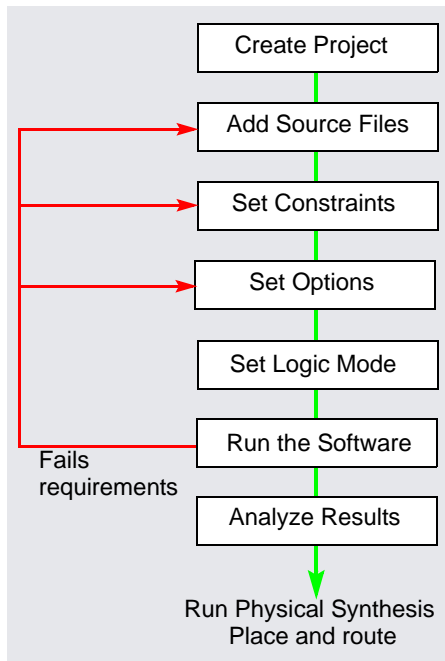
Logic Synthesis with Enhanced Optimization

Enhanced Optimization is a standard feature in the Synplify Premier tool. It includes additional optimizations during logic synthesis and provides an output netlist with better QoR (quality of results) than when running basic logic synthesis. One reason for improved QoR is that the Synplify Premier tool can take advantage of placement-aware data during logic synthesis. This flow is only available for some Altera and Xilinx technologies.

This switch is enabled by default. If your goal is to get the same results that you get from synthesis in the Synplify Pro tool, turn this switch off.

Enhanced Optimization has no effect if Physical Synthesis is enabled. When you use this option, ensure that the Auto Constrain option is disabled (Off).

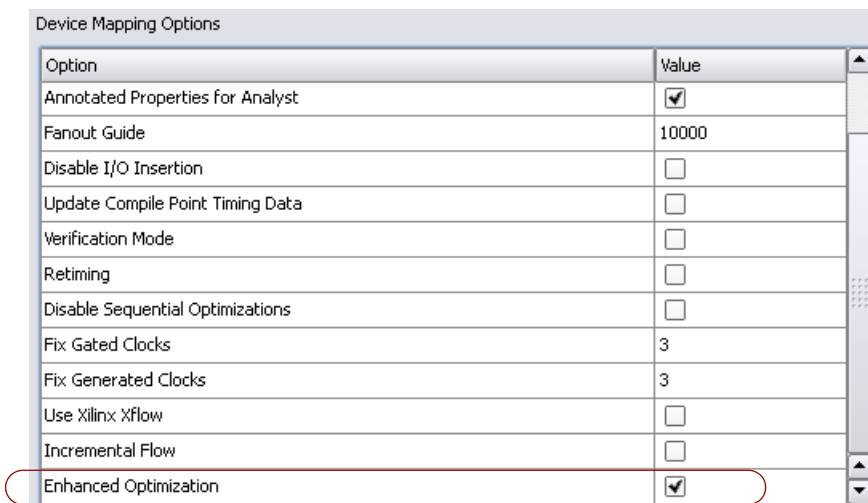
The following figure summarizes the steps in the flow. The steps are briefly described after the figure.



Running Logic Synthesis with Enhanced Optimization

Physical synthesis automatically runs various logic optimizations as part of the process. These optimizations do not run in the basic logic synthesis process. If you want to run logic synthesis only, but want to include these enhanced optimizations, use the following procedure.

1. Create a Synplify Premier 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. Specify logic synthesis with enhanced optimizations.
 - Disable the Physical Synthesis option, either in the Project window or in the Implementation Options dialog box. This directs the software to run logic synthesis only.



- Make sure that Fast Synthesis is disabled, either in the Project view or on the Options tab of the Implementation Options dialog box. The two options are contradictory and if both options are enabled, Fast Synthesis takes priority.
- Enable Enhanced Optimization in the Device tab of the Implementation Options dialog box. When you enable the Physical Synthesis option, the

software automatically uses enhanced optimizations as part of that flow, so you do not have to specifically enable it.

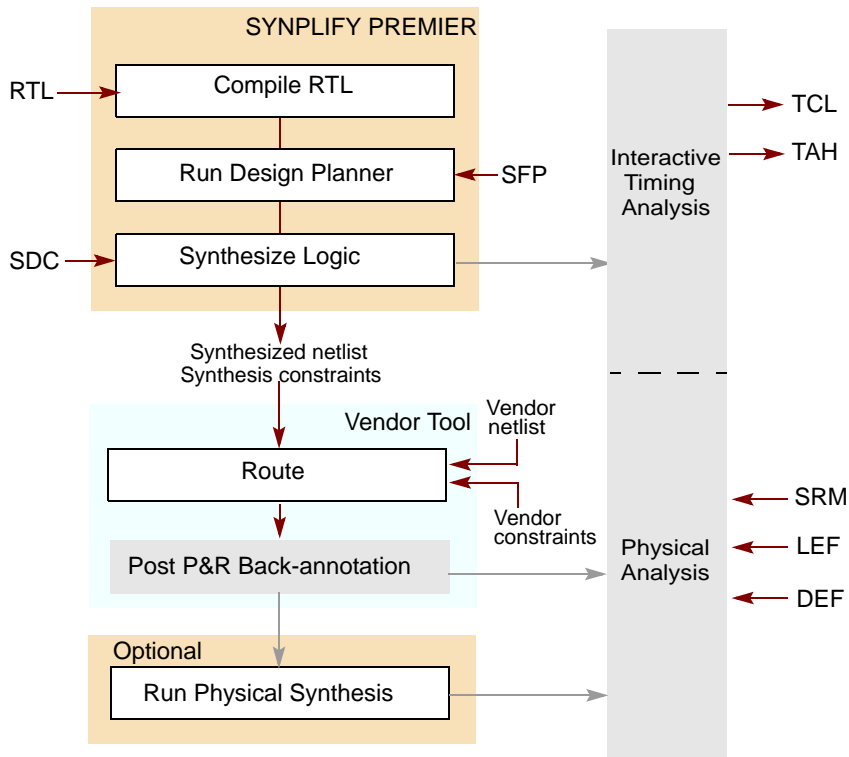
6. Click Run to run logic synthesis.
7. Analyze the results, using the log file, the HDL Analyst schematic views, the Message window and the Log Watch window.

After you have completed the design, you can use the output files to run place-and-route with the vendor tool. You could also run physical synthesis before placement and routing.

Design Plan-Based Logic Synthesis

This flow lets you use a floorplan to guide logic synthesis; you do not have to run physical synthesis. To do this, you require the Synplify Premier software with the Design Planner option (see [Chapter 11, *Floorplanning with Design Planner*](#) for information about using this tool). This flow supports more recent Altera and Xilinx technologies.

The following figure shows the phases and tools used in the flow, and some of the major inputs and outputs. The interactive timing analysis, physical synthesis and analysis, and backannotation steps that are shown in gray are optional.

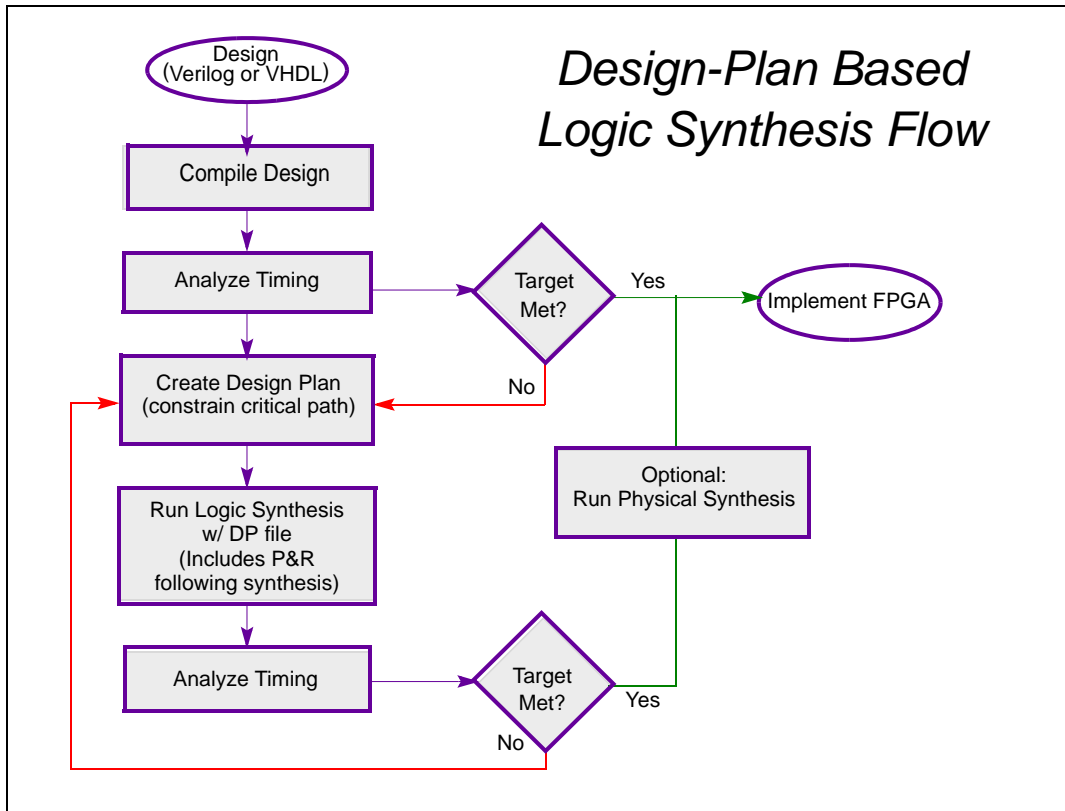


Running Logic Synthesis with a Design Plan


With this methodology, you use the Design Planner tool to manually create physical constraints that assign critical path logic to specific locations on the die to improve performance. You then use this design plan file to constrain logic synthesis.

You can only use this methodology if you have the Design Planner option and if you are targeting certain Altera and Xilinx technologies.

The figure below shows the logic synthesis flow based on a design plan.



1. Set up the project and compile the design in logic synthesis mode.
 - Set up the design as described in [Set up the Altera Physical Synthesis Project, on page 61](#), and [Set up the Xilinx Physical Synthesis Project, on page 70](#). Set up a P&R implementation.
 - Disable the Physical Synthesis option to run the tool in logic synthesis mode.
 - Compile the design.
2. Analyze timing results.
 - Analyze timing.
 - Determine which components you want to assign to regions.

3. Launch the Design Planner tool () and do the following:
- Create regions for the critical paths and interactively assign the critical paths to regions of the chip. See [Working with Regions, on page 505](#), [Working with Altera Regions, on page 519](#), [Working with Xilinx Regions, on page 523](#) and [Assigning Objects to Xilinx Regions, on page 527](#) for details.
 - Obtain a size estimation for each RTL block in the design. See [Checking Utilization, on page 517](#) for details.
 - For multiple clocks, assign critical logic associated with each clock domain (that does not meet design requirements) to a unique region to avoid resource contention.
 - If you have any black boxes in your design, assign them to a region. Designate this region as an IP block, so that the Synplify Premier software can instantiate the black box in the .vqm file. However, you must provide the content for the black box so that the place-and-route tool can run successfully.

For details about using Design Planner, see [Floorplanning with Design Planner, on page 487](#). Consult the following for more information on how to complete the Design Plan file (.sfp): [Creating and Using a Design Plan File for Physical Synthesis, on page 494](#), [Working with Regions, on page 505](#), and [Assigning Pins and Clocks, on page 495](#).

- Save the design plan file (.sfp) and add it to your project.
4. Run logic synthesis.
- Make sure the Physical Synthesis switch is disabled, but that the project includes the physical constraints file (.sfp).
 - Set up the project to automatically run place-and-route after synthesis completes. Alternatively, you can run the P&R tool in standalone mode.

The synthesis tool honors the region placement constraints in the floorplan file. It treats each region you defined in the floorplan as a hard hierarchy, and does not optimize across this boundary. When synthesis is complete, the tool generates a structural netlist for the target technology and a Tcl script that contains the information for forward-annotation, like the region assignments.

The tool then launches the P&R tool, and uses the forward-annotated constraints to direct the P&R run.

5. Analyze the timing in the Synplify Premier tool, using the log file and analysis tools. See [Checking Log Results, on page 596](#), [Analyzing Timing in Schematic Views, on page 730](#), and [Using the Stand-alone Timing Analyst, on page 736](#) for details.

If the target is met, you can continue to P&R. If not, you should re-evaluate timing and placement. Or, you can run physical synthesis.

Graph-Based Physical Synthesis

Synplify Premier graph-based physical synthesis is an automated, single-pass flow that allows you to constrain assigned logic to specific locations, and which optimizes the design based on this placement information. The essence of the graph-based approach is that preexisting wires, switches and placement sites used for routing an FPGA are represented as a detailed routing resource graph. The Synplify Premier tool can then allow for delay and wire availability, which produces more accurate results and improves timing closure. During physical synthesis, the tool performs concurrent placement and synthesis optimizations to ensure fast routes for critical paths. It generates a fully-placed and physically-optimized netlist ready for the vendor place-and-route tool.

Physical synthesis does not require a design plan or place-and-route implementation. If you want to use a design plan file with this flow, see [Graph-Based Physical Synthesis with Design Planner, on page 46](#). For graph-based physical synthesis, the tool automatically performs placement with backannotation during the physical synthesis run. It absorbs the core files into the Synplify Premier database for timing, placement and optimizations. See the following for further details:

- [Design Phases in Graph-based Physical Synthesis, on page 42](#)
- [Graph-based Physical Synthesis Flows for Different Vendors, on page 44](#)

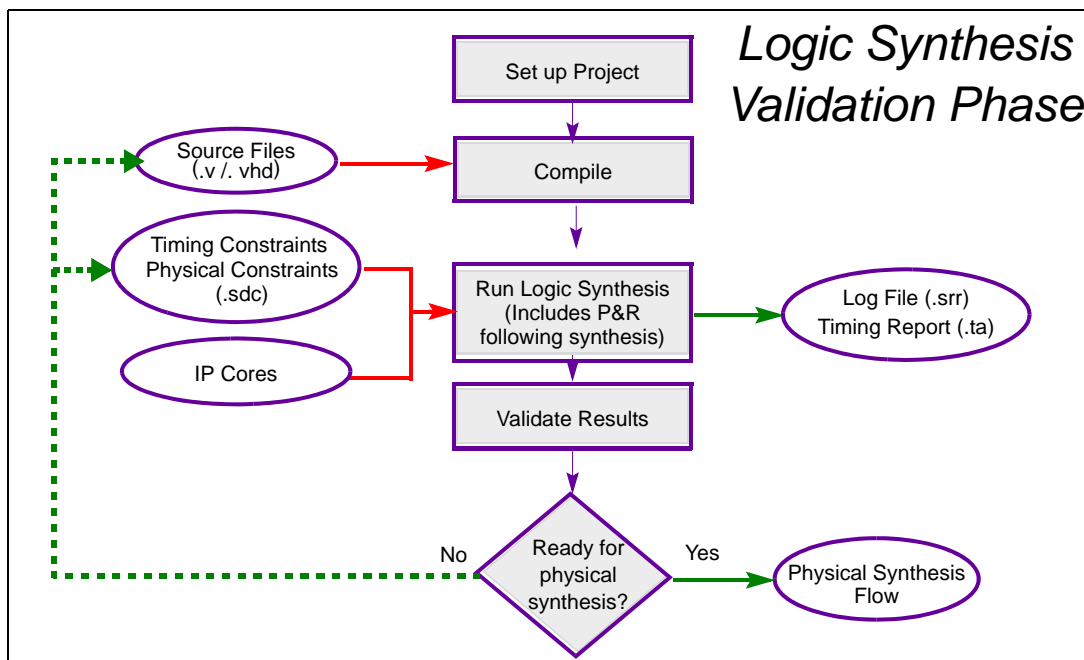
Design Phases in Graph-based Physical Synthesis

The physical synthesis design flow consists of two phases: logic synthesis validation, and physical synthesis.

Logic Synthesis Validation

You first run logic synthesis to ensure that the design can successfully complete logic synthesis and place-and-route, and that it has been assigned accurate, realistic constraints. Doing an initial logic synthesis run can save valuable time by identifying obvious problems early in the process.

The figure below shows the flow for logic synthesis validation phase.

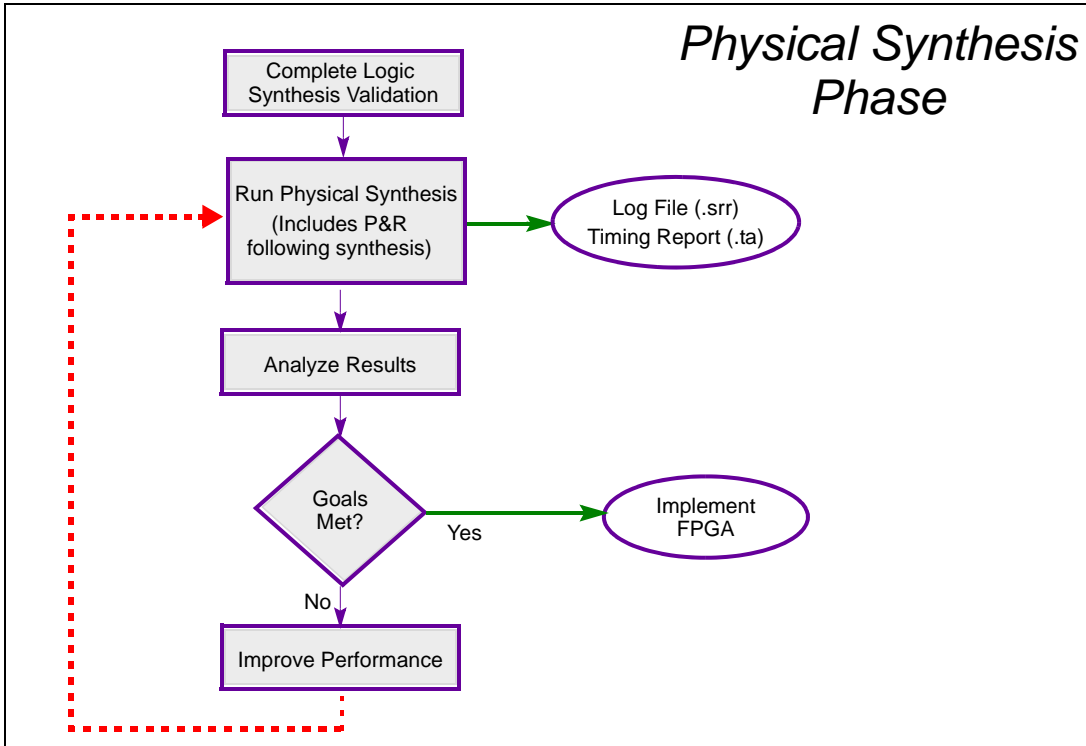


For vendor-specific explanations of the steps shown here, see [Actel Physical Synthesis, on page 52](#), [Altera Physical Synthesis, on page 60](#), and [Xilinx Physical Synthesis, on page 69](#).

Physical Synthesis

After successfully running through logic synthesis, you set up the design for physical synthesis. Physical synthesis merges design optimization and placement to generate a fully-placed, physically-optimized netlist, providing rapid timing closure and increased timing improvement. The tool performs concur-

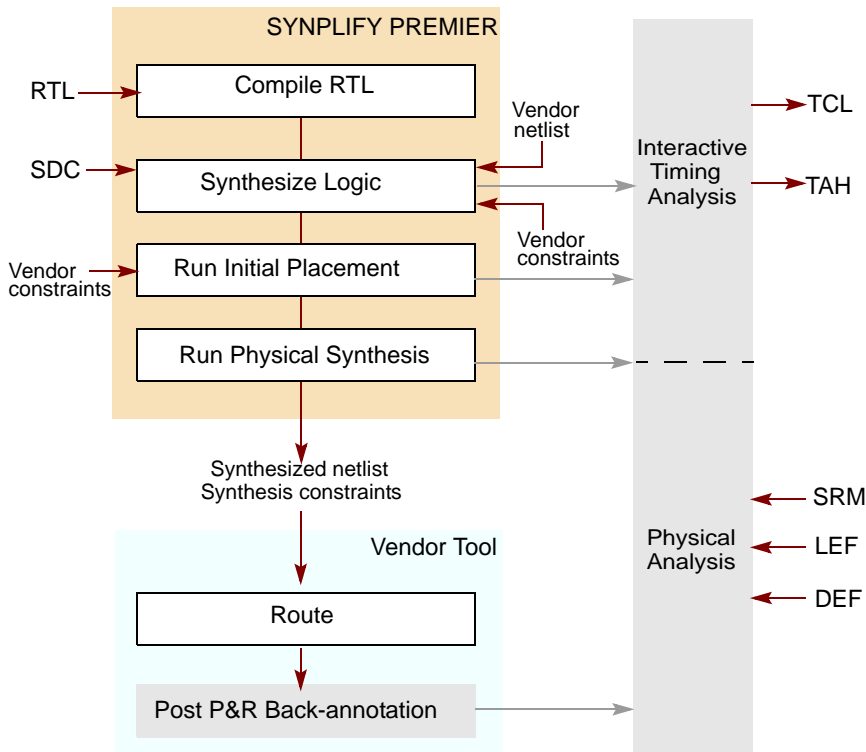
rent placement and optimizations based on timing constraints and the device technology. The output netlist contains placement information. The figure below shows the flow for the physical synthesis phase:



For vendor-specific explanations of the steps shown here, see [Actel Physical Synthesis, on page 52](#), [Altera Physical Synthesis, on page 60](#), and [Xilinx Physical Synthesis, on page 69](#).

Graph-based Physical Synthesis Flows for Different Vendors

The following figure shows how you implement the graph-based physical flow described previously. It shows the phases and tools used in the flow, and some of the major inputs and outputs. The interactive timing analysis, physical analysis, and backannotation steps that are shown in gray are optional.



This flow is currently available only for some Actel, Altera, and Xilinx families. For vendor-specific details on this flow for your technology, see the following:

- [Actel Physical Synthesis](#), on page 52
- [Altera Physical Synthesis](#), on page 60
- [Xilinx Physical Synthesis](#), on page 69

Graph-Based Physical Synthesis with Design Planner

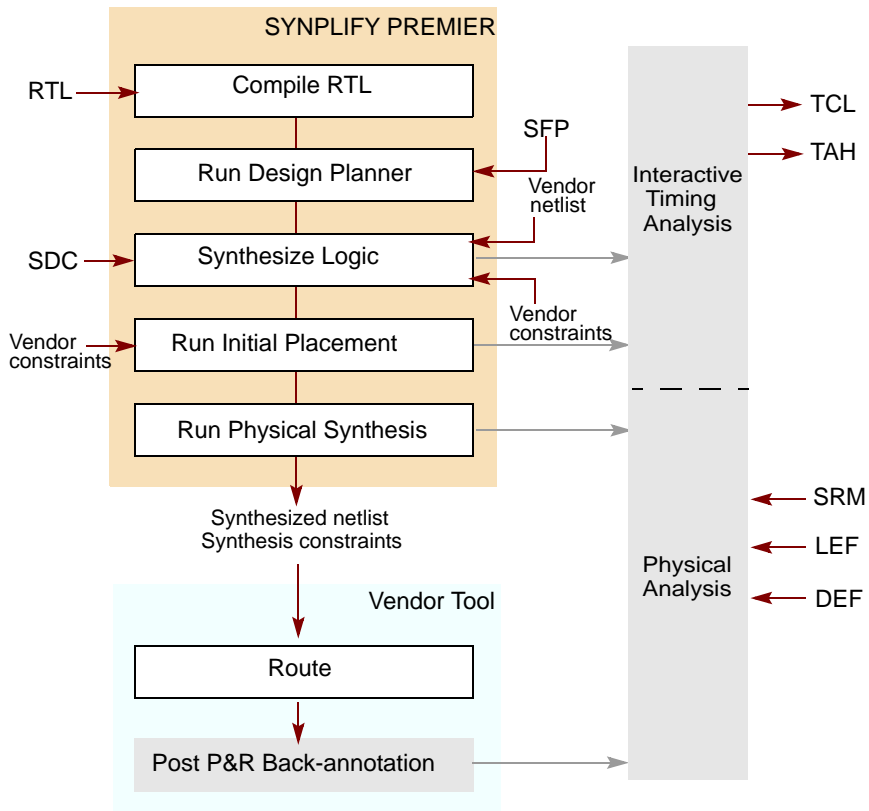
Like the graph-based flow ([Graph-Based Physical Synthesis, on page 42](#)) this is a push-button, fully automated flow that produces a synthesized design with detailed placement, but it uses a design plan file to specify physical constraints for guiding global placement. Use this flow to improve performance.

The design plan usually includes I/O settings and placement information for large blocks. The tool generates placement constraints when you assign RTL logic to ports or regions in the Design Plan view (Design Planner). These regions constrain logic to the areas you specify on the device. During optimizations, these constraints direct global placement and subsequently influence physical optimizations.

You can use this flow with supported Altera and Xilinx families. You must have the Synplify Premier product with the Design Planner option. For vendor-specific details on this flow for your technology, see the following:

- [Altera Physical Synthesis](#), on page 60
- [Xilinx Physical Synthesis](#), on page 69

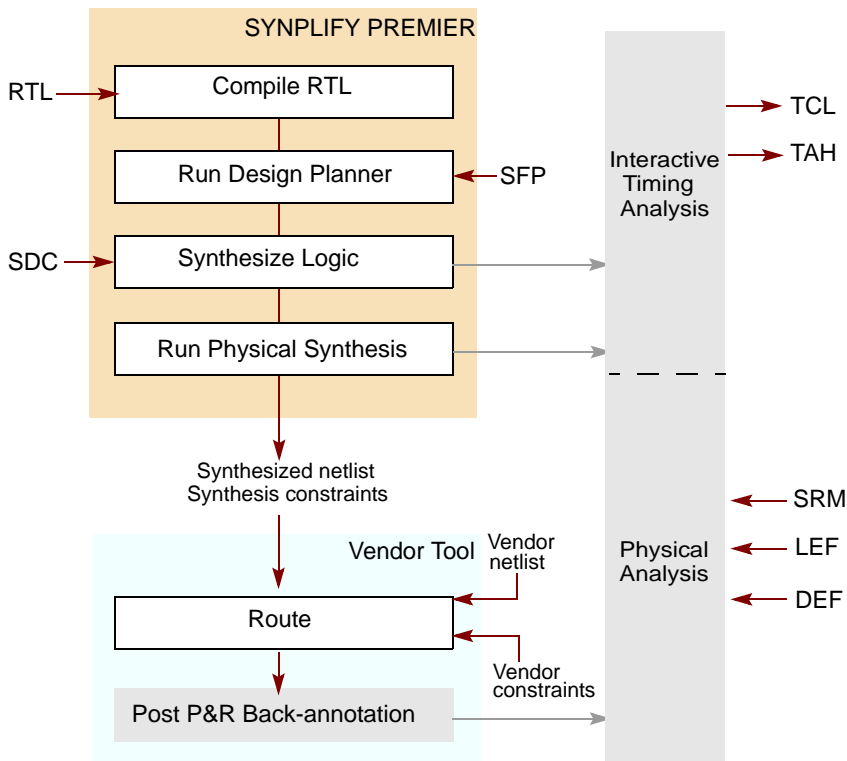
The following figure shows the phases and tools used in the flow, and some of the major inputs and outputs. The interactive timing analysis, physical analysis, and backannotation steps that are shown in gray are optional.



Design Plan-based Physical Synthesis

This is an interactive flow that where you can specify physical constraints before running physical synthesis. It requires the Design Planner option, a tool that improves performance through physical constraints. This flow supports certain Altera and Xilinx technologies. See [Running Physical Synthesis with a Design Plan, on page 49](#) for a procedure using this flow. For information about using Design Planner, see [Chapter 11, Floorplanning with Design Planner](#).

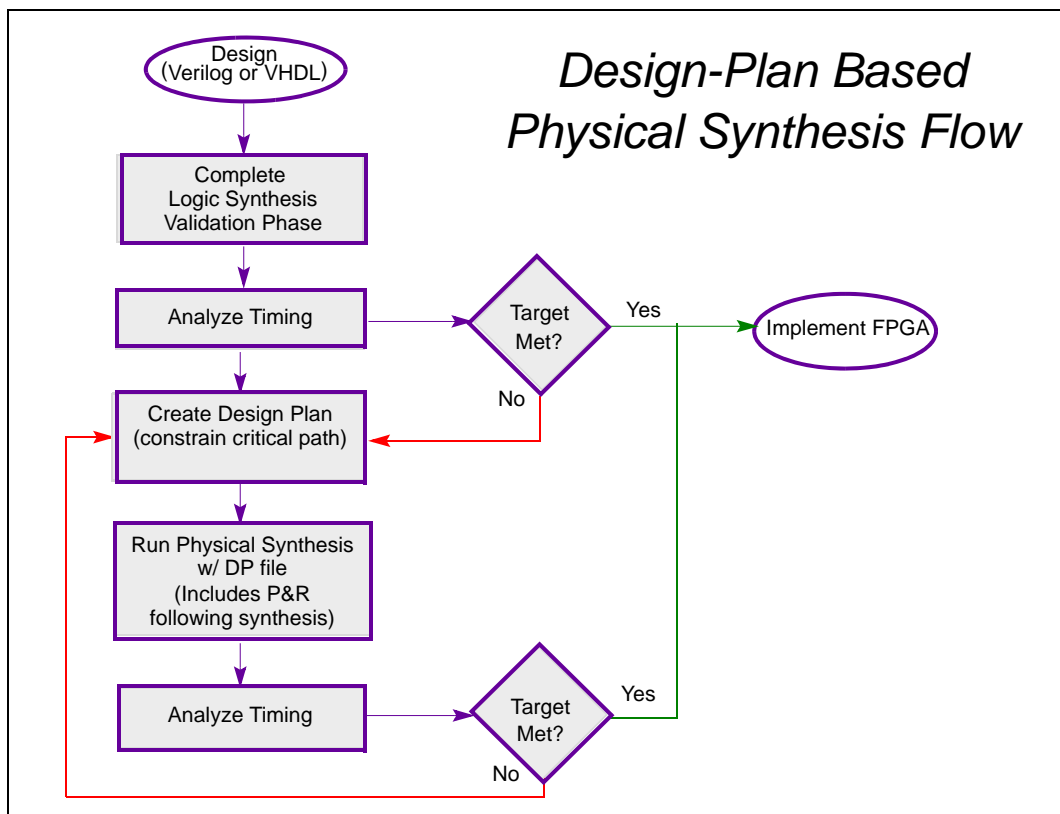
The following figure shows the phases and tools used in the flow, and some of the major inputs and outputs. The interactive timing analysis, physical analysis, and backannotation steps that are shown in gray are optional.



Running Physical Synthesis with a Design Plan

In this flow, you use the Design Planner tool to manually create physical constraints that assign critical path logic to specific locations on the die to improve performance. You then use these constraints to drive physical synthesis for the design.

The figure below shows the physical synthesis design-plan flow.




1. Run logic synthesis.

- Set up the project for your target technology. See [Set up the Altera Physical Synthesis Project, on page 61](#) and [Set up the Xilinx Physical Synthesis Project, on page 70](#) for details.
- Synthesize the design in logic synthesis mode, using timing constraints and no physical constraints.

This phase is to determine if the design can successfully complete synthesis and if timing performance enhancements are needed. The logic synthesis validation phase includes running the netlist through place-and-route after synthesis completes.

2. Analyze timing results. See [Validating Logic Synthesis for Physical Synthesis, on page 609](#) for details.

If timing goals are met, you are done. Otherwise, go to the next step.

3. Determine the critical paths from the P&R results; these are the candidates for logic assignments to regions.
4. Bring up the Design Planner () and do the following:
 - Create regions for the critical paths and interactively assign the critical paths to regions of the chip. See [Working with Regions, on page 505](#), [Working with Altera Regions, on page 519](#), [Working with Xilinx Regions, on page 523](#) and [Assigning Objects to Xilinx Regions, on page 527](#) for details.
 - Obtain a size estimation for each RTL block in the design. See [Checking Utilization, on page 517](#) for details.
 - For multiple clocks, assign critical logic associated with each clock domain (that does not meet design requirements) to a unique region to avoid resource contention.
 - If you have any black boxes in your design, assign them to a region. Designate this region as an IP block, so that the Synplify Premier software can instantiate the black box in the .vqm file. However, you must provide the content for the black box so that the place-and-route tool can run successfully.

For details about using Design Planner, see [Floorplanning with Design Planner, on page 487](#).

You can also open Physical Analyst to view the design and check critical path placement. Consult the following sections for more information on how to complete the Design Plan file (.sfp): [Creating and Using a Design Plan File for Physical Synthesis, on page 494](#), [Working with Regions, on page 505](#), and [Assigning Pins and Clocks, on page 495](#).

5. Save the design plan file (.sfp) and add it to your project.
6. Run physical synthesis. Use the same project file that you created in step 1 above. This time enable the Physical Synthesis switch and include

the physical constraints file (.sfp). This phase also includes running the netlist through place-and-route after synthesis completes.

7. Analyze the timing in the Synplify Premier tool. Use the log file and graphical analysis tools. See [Analyzing Physical Synthesis Results, on page 678](#) for details.

If the target is met, you can continue to the next design phase. If not, you should re-evaluate timing and placement. You might find there is a new critical path or the one that is already assigned to a region that needs tweaking. See [Improving Altera Physical Synthesis Performance, on page 779](#) and [Improving Xilinx Physical Synthesis Performance, on page 827](#) for more suggestions.

Actel Physical Synthesis

For Actel designs, you can run the following flows with the Synplify Premier tool:

For details, see...

Basic logic synthesis	Logic Synthesis Design Flow , on page 32
Graph-based physical synthesis	Graph-Based Physical Synthesis , on page 42 for a description of the flow The Actel-specific steps listed below

The following topics describe the graph-based physical synthesis flow for Actel technologies, and broadly outline the steps to be followed:

- [Set up the Actel Physical Synthesis Project](#), on page 52
- [Run Logic Synthesis for the Actel Physical Synthesis Flow](#), on page 56
- [Validate Logic Synthesis Results for Actel Physical Synthesis](#), on page 56
- [Set up Actel Physical Constraints](#), on page 57
- [Run Actel Physical Synthesis](#), on page 57
- [Analyze Results of Actel Physical Synthesis](#), on page 57

Set up the Actel Physical Synthesis Project

Project setup is the first phase of the physical synthesis design process. The project file (.prj) is a collection of input files and optimization switches required to synthesize your design. You must create a synthesis project, define constraints, set options for the implementation, and set up a P&R project.

The following procedure outlines the procedure you must follow to set up the physical synthesis project. Follow the links if you need further details about any of the steps.

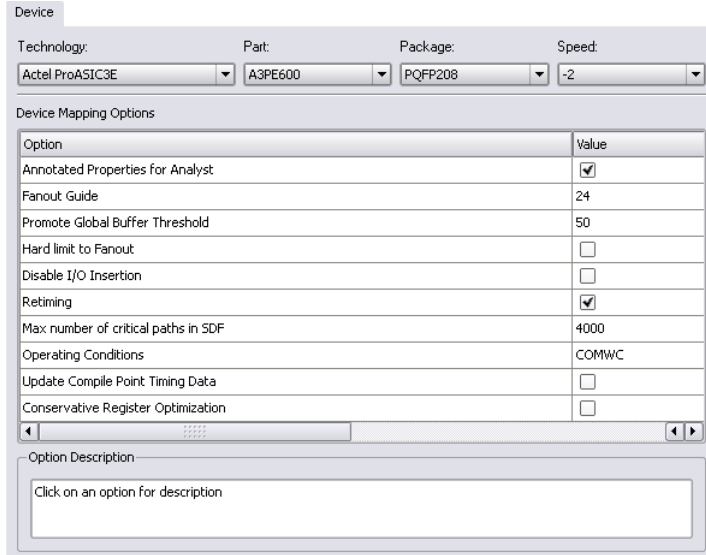
1. Make sure you follow these requirements for this flow:
 - Include the entire design; you cannot have black boxes.

- Assign realistic, accurate timing constraints. Do not over-constrain the tool. See [Improving Performance in Actel Physical Synthesis Designs](#), on page 763 for tips.
 - Use the top-down design methodology. A bottom-up flow is not supported.
 - Install the recommended version of the Actel Designer place-and-route tool before you run physical synthesis. Consult the release notes for the most current information on supported Designer versions. (From the Synplify Premier tool: Help->Online Documents->release_notes.pdf->*Third Party Tool Versions*).
2. Create the project. See [Setting Up HDL Source Files](#), on page 82 and [Setting Up Project Files](#), on page 270 for details.
 3. Set timing constraints.
 - Compile the design.
 - Open the SCOPE interface and set constraints. Timing constraints specify performance goals and describe the design environment. See [Using the SCOPE UI](#), on page 212 and [Specifying Timing Constraints](#), on page 219 for details.

The screenshot shows a window titled "C:\synplify_premier_actel\tutorial.sdc*" containing a table of timing constraints. The table has the following columns: Enabled, Clock Object, Clock Alias, Frequency (MHz), Period (ns), Clock Group, Rise At (ns), Fall At (ns), Duty Cycle (%), Route (ns), Virtual Clock, and Comment. The first row is populated with: Enabled (checkbox), Clock Object (clock), Clock Alias, Frequency, Period, Clock Group (default_clkgroup_0), Rise At, Fall At, Duty Cycle, Route, Virtual Clock (checkbox), and Comment. Below the table are several tabs: Clocks, Clock to Clock, Collections, Inputs/Outputs, Registers, Delay Paths, Attributes, I/O Standards, Compile Points, and Other.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock	Comment
1	<input type="checkbox"/>	clock				default_clkgroup_0					<input type="checkbox"/>	
2												
3												
4												
5												
6												

- Save the constraints file and save the project file.
4. Specify the implementation options for synthesis.
 - Click the Implementation Options button.

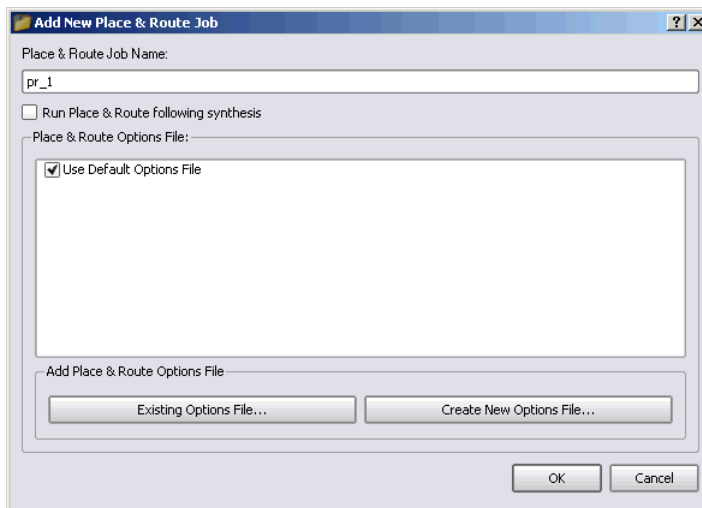


- Set implementation options on the various tabs of the dialog box as shown in the following table. For details about setting implementation options, refer to [Setting Logic Synthesis Implementation Options, on page 289](#).

Device	Set technology and device mapping options. Make sure the Disable I/O Insertion option is disabled, because Synplify Premier physical synthesis requires this setting. See Setting Device Options, on page 289 .
Options	Set optimization switches for synthesis. See Setting Optimization Options, on page 292 .
Constraints	Set an overall target frequency for the design. Select the constraint file you want to use. See Specifying Global Frequency and Constraint Files, on page 294 .
Implementation Results	Specify the output results directory and output file options. See Specifying Result Options, on page 296 for details.

Timing Report	Specify the number of critical paths and start/end points to display in the timing report. See Specifying Timing Report Output, on page 297 .
Verilog/VHDL	Specify the HDL options. See Setting Verilog and VHDL Options, on page 298 .
Netlist Restructure	Specify options for any necessary netlist optimizations, and the netlist restructure file (.nrf) for which bit slicing or zippering might have been performed. See Setting Synplify Premier Netlist Restructuring Optimizations, on page 330 for descriptions.

- Click OK to apply the implementation options.
 - Save the project file.
5. Create a place-and-route implementation to automatically run the Actel place and route tool from the synthesis interface after synthesis is complete. See [Creating a Place and Route Implementation, on page 332](#) for details.
- Make sure you have the correct version of the P&R tool installed.
 - Create a P&R implementation.



- Specify the Place & Route Job Name. Make sure the Run Place and Route following synthesis switch is enabled and click OK.

- Specify the place-and-route options file. The tool automatically uses default options located in `<install_dir>\lib\Actel\Actel_par.opt`.
- Go to Implementation Options->Place and Route and enable the place-and-route implementation that you want to use for your project.



- Save the project file.

Run Logic Synthesis for the Actel Physical Synthesis Flow

If this is the first time you are running synthesis on the design, run it in logic synthesis mode (with the Physical Synthesis option disabled, as described in [Logic Synthesis Design Flow, on page 32](#)). This initial synthesis run lets you determine if there are any problems that need to be addressed before going on to the physical synthesis stage. See the first step of [Running Physical Synthesis, on page 592](#) for details.

Validate Logic Synthesis Results for Actel Physical Synthesis

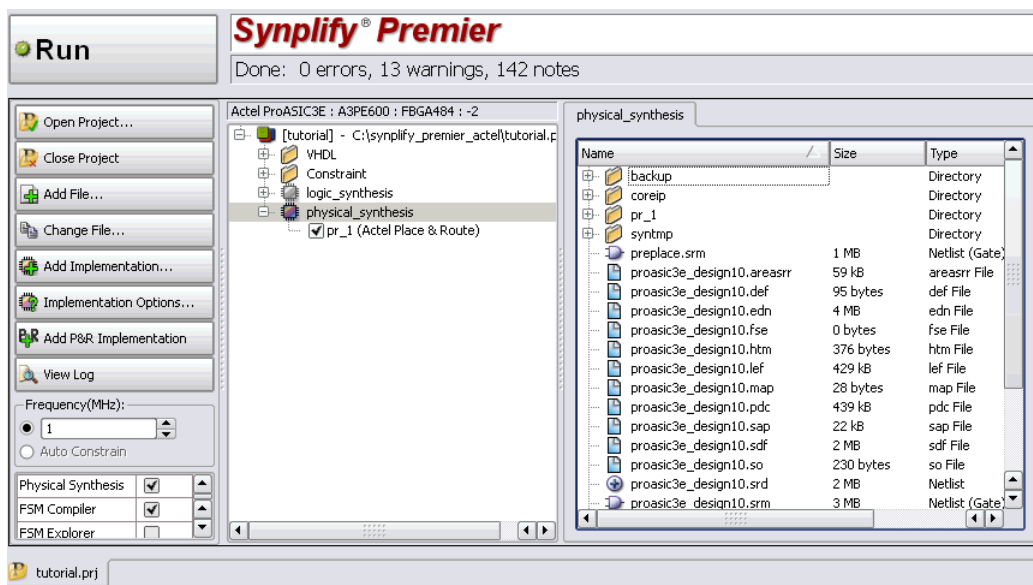
After doing an initial run of logic synthesis, check the results and fix any errors you find. See [Validating Logic Synthesis for Physical Synthesis, on page 609](#) for details.

Set up Actel Physical Constraints

You can translate I/O placement information from the `designname_ba.pdc` place-and-route file to lock down the I/Os during physical synthesis. You can either do this manually by attaching the `syn_loc` attribute to the I/Os, or use the `pdc2sdc` utility to automatically translate the constraints. For details, see [Translating Actel I/O Constraints](#), on page 348.





Run Actel Physical Synthesis

Once you have validated the logic synthesis run and set up the physical constraints, you can run physical synthesis. Make sure to enable the Physical Synthesis switch and to enable the place-and-route implementation before clicking Run. For a detailed procedure, see [Running Physical Synthesis](#), on page 592.



Analyze Results of Actel Physical Synthesis

To determine if your design has met performance goals, use the following Synplify Premier analysis tools to analyze the critical path(s) with negative slack and identify potential solutions to improve performance:

Log file that includes the default timing report (.srr or .htm)	See Checking Log Results, on page 596 .
HDL Analyst	Consists of schematic views that help you analyze the design. See Chapter 5, Specifying Constraints .
RTL View ()	Select HDL Analyst->RTL->Hierarchical View or Flattened View to display the compiled view of the design. See Chapter 15, Analyzing with HDL Analyst and FSM Viewer .
Technology View ()	Select HDL Analyst ->Technology->Hierarchical View, or ->Flattened View to display the mapped view of the design. See Chapter 15, Analyzing with HDL Analyst and FSM Viewer .
Physical Analyst ()	The Physical Analyst provides a visual display of the device, and design placement of instances and nets. Select HDL Analyst->Physical Analyst. See Chapter 16, Analyzing Designs in Physical Analyst .
Timing Analyst ()	The stand-alone timing analyzer produces timing reports (.ta) for specific reporting requirements. See Using the Stand-alone Timing Analyst, on page 736 .

Use these guidelines to analyze the results:

Check this...	Tool
Are start and end points being constrained by the proper clocks?	Timing report You can also trace the clock network using HDL Analyst Technology view.

Check this...	Tool
Is the critical path a multi-cycle path or false path?	Timing report HDL Analyst
If the path is inside a state machine, is the FSM being fully optimized?	HDL Analyst. Open the RTL view and push down into the state machine module to display the FSM viewer.
Are the net delays contributing to the highest percentage on the critical path?	Timing report Check the % breakdown of delay for each path. Search for Total path delay . Physical Analyst Use it to analyze the instance placement of the critical path.

For more details and guidelines on improving design performance, see [Improving Performance in Actel Physical Synthesis Designs, on page 763](#).

Altera Physical Synthesis

In addition to the graph-based physical design flow, you can use the following Synplify Premier flows in Altera designs.

	For details, see...
Basic logic synthesis	Logic Synthesis Design Flow , on page 32
Logic synthesis with enhanced optimization	Logic Synthesis with Enhanced Optimization , on page 36
Logic synthesis with fast synthesis	Fast Synthesis , on page 481
Logic synthesis with design plan	Design Plan-Based Logic Synthesis , on page 38
Physical synthesis with design plan	Design Plan-based Physical Synthesis , on page 48
Graph-based synthesis with a design plan	Graph-Based Physical Synthesis with Design Planner , on page 46

Altera Graph-based Physical Synthesis

The following topics provide an overview of what you need to do to run graph-based physical synthesis in your Altera designs.

- [Guidelines for Physical Synthesis in Altera Designs](#), on page 60
- [Set up the Altera Physical Synthesis Project](#), on page 61
- [Run Logic Synthesis for the Altera Physical Synthesis Flow](#), on page 65
- [Validate Logic Synthesis Results for Altera Physical Synthesis](#), on page 66
- [Run Altera Physical Synthesis](#), on page 66
- [Analyze Results of Altera Physical Synthesis](#), on page 66

Guidelines for Physical Synthesis in Altera Designs

Follow these guidelines for physical synthesis:

- Include the entire design – black boxes cannot be present. However, Altera LPMs (Library of Parameterized Modules) or Megafunctions are

supported. See [Using Altera LPMs or Megafunctions in Synthesis](#), on page 161 if more information is needed.

- Use the appropriate methodology defined for Altera IPs or Nios II cores in the design. For more information, see the following:
 - [Including Altera MegaCore IP Using an IP Package](#), on page 181
 - [Implementing Megafunctions with Grey Box Models](#), on page 175
 - [Including Altera Processor Cores Generated in SOPC Builder](#), on page 186.
- Assign realistic, accurate timing constraints. Do not over-constrain the tool. (See [Improving Altera Physical Synthesis Performance](#), on page 779 for tips.)
- Use the top-down design methodology. (A bottom-up flow is not supported.)
- Do not use compile points with graph-based physical synthesis.
- Install the recommended version of the Altera Quartus II place-and-route tool.

Set up the Altera Physical Synthesis Project

Project setup is the first phase of the physical synthesis design process. The project file (.prj) is a collection of input files and optimization switches required to synthesize your design. This section contains details on how to set up the file.

1. Make sure you follow these requirements for this flow:
 - Make sure the design is properly constrained. (See [Improving Altera Physical Synthesis Performance](#), on page 779 for tips.)
 - Make sure to specify I/O pin location constraints for all pins in the design for physical synthesis.
 - Make sure to include any I/O constraints from the Quartus settings file (.qsf) as necessary. You can translate the I/O constraints and I/O standards to .sdc format with a utility. See [Translating Altera QSF Constraints](#), on page 253.)
 - If you have the Design Planner option, you can use a design plan file (.sfp) for physical synthesis.

- If you are using one of the graph-based physical synthesis flows, make sure you select a target technology that is supported. See [Altera Graph-based Physical Synthesis, on page 60](#) or [Xilinx Graph-based Physical Synthesis, on page 69](#).
 - Keep the guidelines described in [Guidelines for Physical Synthesis in Altera Designs, on page 60](#) in mind.
2. Create the project. If you are using a design plan file, make sure to add it to the project.

See [Setting Up HDL Source Files, on page 82](#) and [Setting Up Project Files, on page 270](#) for details.

3. Set constraints.

- Compile the design. Open the SCOPE interface and set constraints. Timing constraints specify performance goals and describe the design environment. See [Using the SCOPE UI, on page 212](#) and [Specifying Timing Constraints, on page 219](#) for details.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock	Comment
1	<input type="checkbox"/>	clock				default_clkgroup_0					<input type="checkbox"/>	
2												
3												
4												
5												
6												

- Add physical constraints.
 - Translate constraints from the Quartus settings file (QSF) and combine them with the timing constraints into a single .sdc constraint file. See [Translating Altera QSF Constraints, on page 253](#) for details.
 - Check your constraints with Run->Constraint Check.
 - Save the constraints file and save the project file.
4. Specify the implementation options for synthesis.
- Click the Implementation Options button.

Device

Technology: Part: Package: Speed:

Device Mapping Options

Option	Value
Fanout Guide	30
Annotated Properties for Analyst	<input checked="" type="checkbox"/>
Verification Mode	<input type="checkbox"/>
Disable I/O Insertion	<input type="checkbox"/>
Update Compile Point Timing Data	<input type="checkbox"/>
Retiming	<input type="checkbox"/>
Disable Sequential Optimizations	<input type="checkbox"/>
Fix Gated Clocks	3
Fix Generated Clocks	3
Altera Models	on
Enhanced Optimization	<input checked="" type="checkbox"/>

Option Description

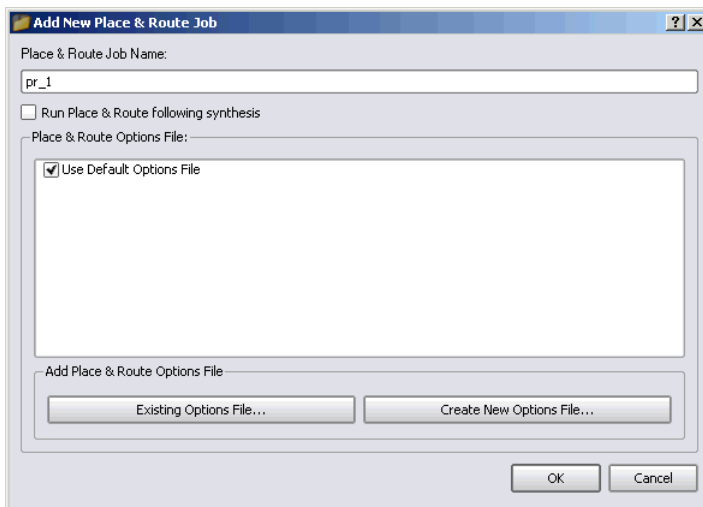
Click on an option for description

- Set implementation options on the various tabs of the dialog box as shown in the following table. For details about setting implementation options, refer to [Setting Logic Synthesis Implementation Options, on page 289](#).

Device	Set technology and device mapping options. Make sure the Disable I/O Insertion option is disabled, because Synplify Premier physical synthesis requires this setting. See Setting Device Options, on page 289 .
Options	Set optimization switches for synthesis. See Setting Optimization Options, on page 292 .
Constraints	Set an overall target frequency for the design. Select the constraint files you want to use. See Specifying Global Frequency and Constraint Files, on page 294 .
Implementation Results	Specify the output results directory and output file options. See Specifying Result Options, on page 296 for details.

Timing Report	Specify the number of critical paths and start/end points to display in the timing report. If you want an island timing report, enable the option. See Specifying Timing Report Output , on page 297.
Verilog/VHDL	Specify the HDL options. See Setting Verilog and VHDL Options , on page 298.
Netlist Restructure	Specify options for any necessary netlist optimizations, and the netlist restructure file (.nrf) for which bit slicing or zipping might have been performed. See Setting Synplify Premier Netlist Restructuring Optimizations , on page 330 for descriptions.

- Click OK to apply the implementation options. Save the project file.
5. Create a place-and-route implementation to automatically run the Altera Quartus II place and route tool from the synthesis UI after synthesis.
 - Make sure you have the correct version of the P&R tool and that you have set the environment variables for the tool.
 - Create a P&R implementation. See [Creating a Place and Route Implementation](#), on page 332 for details.



- Specify the Place & Route Job Name. Make sure the Run Place and Route following synthesis switch is enabled and click OK.
- Specify the place-and-route options file. The tool automatically uses default options located in `<install_dir>\lib\altera\altera_par.tcl`. For more information, see [Specifying Altera Place-and-Route Options, on page 337](#). Select other options for backannotation and for forward-annotation of constraints and click OK.
- Go to Implementation Options->Place and Route and enable the place-and-route implementation that you want to use for your project.



- Save the project file.

Run Logic Synthesis for the Altera Physical Synthesis Flow

If this is the first time you are running synthesis on the design, you must run logic synthesis mode. This means that you run synthesis with the Physical Synthesis switch disabled. You can choose to run logic synthesis in the Synplify Premier tool with the Enhanced Optimization mode or the standard logic synthesis mode for certain Altera devices. When the Enhanced Optimization mode is:

- Enabled (box is checked in the Project view or on the Implementation Options - Device tab) — Additional placement aware optimizations are

used to achieve QoR results that exceed the standard logic synthesis QoR. This is the default.

- Disabled (box is unchecked in the Project view or on the Implementation Options - Device tab) — The standard logic synthesis QoR can be achieved.

This initial synthesis run is to determine if there are any problems that need to be addressed before going on to the physical synthesis stage. See the first step of [Running Physical Synthesis, on page 592](#) for details of running logic synthesis as part of a physical synthesis flow.

Validate Logic Synthesis Results for Altera Physical Synthesis





After doing an initial run of logic synthesis, check the results and fix any errors you find. See [Validating Logic Synthesis for Physical Synthesis, on page 609](#) for details.

Run Altera Physical Synthesis

Once you have validated the logic synthesis run and set up the physical constraints, you can run physical synthesis. Make sure to enable the Physical Synthesis switch and to enable the place-and-route implementation before clicking Run. If you are using a design plan file, you must also enable this file. For a detailed procedure, see [Running Physical Synthesis, on page 592](#).

Analyze Results of Altera Physical Synthesis

To determine if your design has met performance goals, use the following Synplify Premier analysis tools to analyze the critical path(s) with negative slack and identify potential solutions to improve performance:

Log file that includes the default timing report (.srr or .htm)	See Checking Log Results, on page 596 .
HDL Analyst	Consists of schematic views that help you analyze the design. See Chapter 5, Specifying Constraints .
RTL View ()	Select HDL Analyst->RTL->Hierarchical View or Flattened View to display the compiled view of the design. See Chapter 15, Analyzing with HDL Analyst and FSM Viewer .
Technology View ()	Select HDL Analyst ->Technology->Hierarchical View, or ->Flattened View to display the mapped view of the design. See Chapter 15, Analyzing with HDL Analyst and FSM Viewer .
Physical Analyst ()	The Physical Analyst provides a visual display of the device, and design placement of instances and nets. Select HDL Analyst->Physical Analyst. See Chapter 16, Analyzing Designs in Physical Analyst .
Timing Analyst ()	The stand-alone timing analyzer produces timing reports (.ta) for specific reporting requirements. See Using the Stand-alone Timing Analyst, on page 736 .

Use these guidelines to analyze the results:

Check this...	Tool
Are start and end points being constrained by the proper clocks?	Timing report You can also trace the clock network using HDL Analyst Technology view.
Is the critical path a multi-cycle path or false path?	Timing report HDL Analyst
Will pipelining improve results?	HDL Analyst

Check this...	Tool
If the path is inside a state machine, is the FSM being fully optimized?	HDL Analyst. Open the RTL view and push down into the state machine module to display the FSM viewer.
Are the net delays contributing to the highest percentage on the critical path?	Timing report Check the % breakdown of delay for each path. Search for Total path delay . Physical Analyst Use it to analyze the instance placement of the critical path.
Will physical constraints improve results?	Physical Analyst Use it to analyze the design. Design Planner Use it to assign logic to regions and generate a design plan file.

Xilinx Physical Synthesis

In addition to the graph-based physical design flow, you can use the following Synplify Premier flows in Xilinx designs.

	For details, see...
Basic logic synthesis	Logic Synthesis Design Flow , on page 32
Logic synthesis with enhanced optimization	Logic Synthesis with Enhanced Optimization , on page 36
Logic synthesis with fast synthesis	Fast Synthesis , on page 481
Logic synthesis with design plan	Design Plan-Based Logic Synthesis , on page 38
Physical synthesis with design plan	Design Plan-based Physical Synthesis , on page 48
Graph-based synthesis with a design plan	Graph-Based Physical Synthesis with Design Planner , on page 46

Xilinx Graph-based Physical Synthesis

The following provide an overview of what you need to do to run graph-based physical synthesis in your Xilinx designs, and other related information.

- [Set up the Xilinx Physical Synthesis Project](#), on page 70
- [Run Logic Synthesis for the Xilinx Physical Synthesis Flow](#), on page 74
- [Validate Logic Synthesis Results for Xilinx Physical Synthesis](#), on page 75
- [Run Xilinx Physical Synthesis](#), on page 75
- [Analyze Results of Xilinx Physical Synthesis](#), on page 75
- [Guidelines for Xilinx Timing Constraints for Physical Synthesis](#), on page 77
- [Using IP Cores in Xilinx Physical Synthesis Flows](#), on page 78
- [Placement and Routing Phases in Xilinx Physical Synthesis](#), on page 78

Set up the Xilinx Physical Synthesis Project

Project setup is the first phase of the physical synthesis design process. The project file (.prj) is a collection of input files and optimization switches required to synthesize your design. This section contains details on how to set up the file.

1. Make sure you follow these requirements for this flow:
 - Make sure the design is complete including all IPs (no black boxes). See [Working with Xilinx IP, on page 195](#) for more information.
 - Make sure the design is properly constrained. Do not over-constrain, but use realistic constraints. See [Improving Xilinx Physical Synthesis Performance, on page 827](#) for tips.
 - Use the top-down design methodology. A bottom-up flow is not supported.)
 - Do not use compile points with graph-based physical synthesis.
 - Install the recommended version of the Xilinx ISE place-and-route tool. Check the release notes.
 - Depending on your target Xilinx technology, you can optionally specify a design plan file (.sfp) for physical synthesis. Using the sfp file requires that you have the separately-licensed Synplify Premier Design Planner option.
 - If you are using one of the graph-based physical synthesis flows, make sure you select a target technology that is supported.
2. Create the project.
 - Add the source files and core IP files (.edn/.ngc) to the project.
 - If you are using the optional .sfp design plan file, make sure to add it to the project.

See [Setting Up HDL Source Files, on page 82](#) and [Setting Up Project Files, on page 270](#) for details.

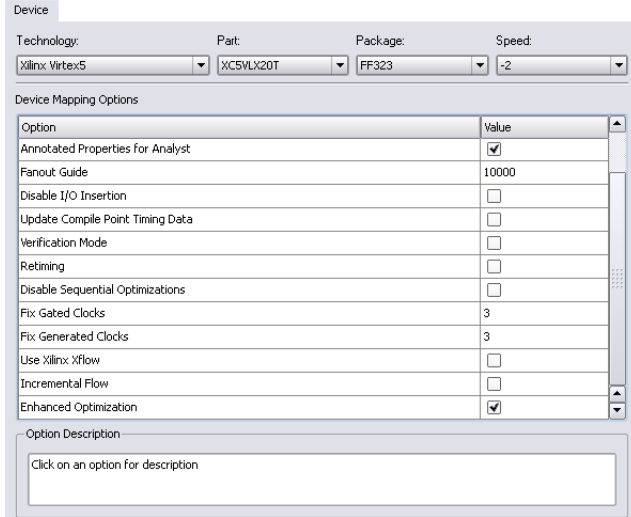
3. Set constraints.
 - Compile the design. Open the SCOPE interface and set constraints. Timing constraints specify performance goals and describe the design environment. See [Using the SCOPE UI, on page 212](#) and [Specifying Timing Constraints, on page 219](#) for details, and [Guidelines for Xilinx Timing Constraints for Physical Synthesis, on page 77](#) for guidelines.

The screenshot shows the 'Clocks' window in Synplify. The window title is 'C:\synplify_premier_actel\tutorial.sdc *'. The table below is the main content of the window.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock	Comment
1	<input type="checkbox"/>	clock				default_clkgroup_0					<input type="checkbox"/>	
2												
3												
4												
5												
6												

At the bottom of the window, there are several tabs: 'Clocks', 'Clock to Clock', 'Collections', 'Inputs/Outputs', 'Registers', 'Delay Paths', 'Attributes', 'I/O Standards', 'Compile Points', and 'Other'. The 'Clocks' tab is currently selected.

- If you are iterating through the flow, make sure that any -route constraints are turned off for logic synthesis and turned on for physical synthesis. See [Using -route for Physical Synthesis in Xilinx Designs, on page 230](#).
 - Add physical constraints. Use the xc_use_rpms attribute to manage relationally placed macros (RPMs) during physical synthesis. For details, refer to [xc_use_rpms Attribute, on page 1188](#) in the *Reference Manual*.
 - If you have Xilinx UCF constraints, translate them as described in [Converting and Using Xilinx UCF Constraints, on page 255](#).
 - Check your constraints with Run->Constraint Check.
 - Save the constraints file and save the project file.
4. Specify the implementation options for synthesis.
- Click the Implementation Options button.

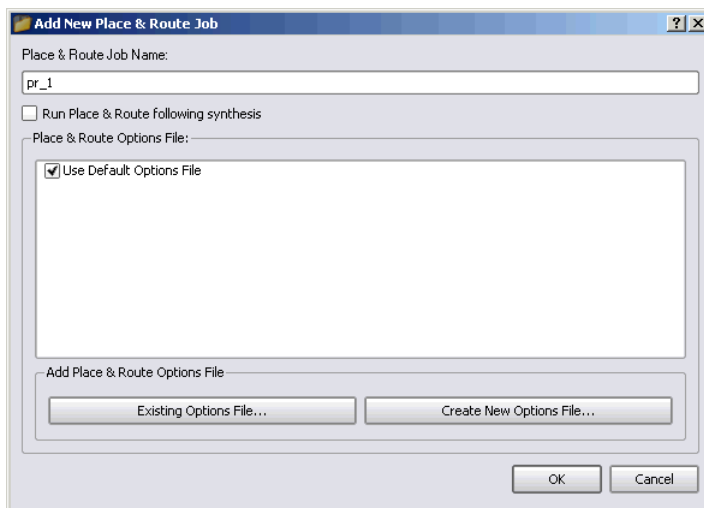


- Set implementation options on the various tabs of the dialog box as shown in the following table. For details about setting implementation options, refer to [Setting Logic Synthesis Implementation Options](#), on page 289.

Device	Set technology and device mapping options. Make sure the Disable I/O Insertion option is disabled, because Synplify Premier physical synthesis requires this setting. See Setting Device Options , on page 289.
Options	Set optimization switches for synthesis. See Setting Optimization Options , on page 292.
Constraints	Set an overall target frequency for the design. Select the constraint file you want to use. See Specifying Global Frequency and Constraint Files , on page 294.
Implementation Results	Specify the output results directory and output file options. See Specifying Result Options , on page 296 for details.

Timing Report	Specify the number of critical paths and start/end points to display in the timing report. If you want an automatic island timing report, enable the option. See Specifying Timing Report Output , on page 297.
Verilog/VHDL	Specify the HDL options. See Setting Verilog and VHDL Options , on page 298.
Netlist Restructure	Specify options for any necessary netlist optimizations, and the netlist restructure file (.nrF) for which bit slicing or zippering might have been performed. See Setting Synplify Premier Netlist Restructuring Optimizations , on page 330 for descriptions.

- Click OK to apply the implementation options. Save the project file.
5. Create a place-and-route implementation to automatically run the Xilinx ISE place and route tool from the synthesis UI after synthesis.
- Make sure you have the correct version of the P&R tool and that you have set the environment variables for the tool. Check the release notes. Select Help->Online Documents->release_notes.pdf->*Third Party Tool Versions*.
 - Create a P&R implementation. See [Creating a Place and Route Implementation](#), on page 332 for details.



- Specify the Place & Route Job Name. Make sure the Run Place and Route following synthesis switch is enabled and click OK.
- Specify the place-and-route options file. The tool automatically uses default options in the .tcl file from <install_dir>\lib\Xilinx. This file is used by the Xilinx xtclsh executable to run the P&R tool. For more information, see [Specifying Xilinx Place-and-Route Options in a Tcl File, on page 340](#). Select other options for backannotation and for forward-annotation of constraints and click OK.
- Go to Implementation Options->Place and Route and enable the place-and-route implementation that you want to use for your project.



- Save the project file.
6. If you want to override the global placement options, use the SYN_XILINX_GLOBAL_PLACE_OPT environment variable. See [Specifying Xilinx Global Placement Options, on page 346](#).

Run Logic Synthesis for the Xilinx Physical Synthesis Flow

If this is the first time you are running synthesis on the design, you must run logic synthesis mode. This means that you run synthesis with the Physical Synthesis switch disabled. This initial synthesis run is to determine if there are

any problems that need to be addressed before going on to the physical synthesis stage. See the first step of [Running Physical Synthesis, on page 592](#) for details of running logic synthesis as part of a physical synthesis flow.

Validate Logic Synthesis Results for Xilinx Physical Synthesis

After doing an initial run of logic synthesis, check the results and fix any errors you find. See [Validating Logic Synthesis for Physical Synthesis, on page 609](#) for details.

Run Xilinx Physical Synthesis


Once you have validated the logic synthesis run and set up the physical constraints, you can run physical synthesis. Make sure to enable the Physical Synthesis switch and to enable the place-and-route implementation before clicking Run. If you are using a design plan file, you must also enable this file. For a detailed procedure, see [Running Physical Synthesis, on page 592](#). The tool runs physical synthesis, automatically including enhanced optimizations as part of the run.

Analyze Results of Xilinx Physical Synthesis

To determine if your design has met performance goals, use the following Synplify Premier analysis tools to analyze the critical path(s) with negative slack and identify potential solutions to improve performance:


Log file that includes the default timing report (.srr or .htm) See [Checking Log Results, on page 596](#).

HDL Analyst Consists of schematic views that help you analyze the design. See [Chapter 5, Specifying Constraints](#).


RTL View () Select HDL Analyst->RTL->Hierarchical View or Flattened View to display the compiled view of the design. See [Chapter 15, Analyzing with HDL Analyst and FSM Viewer](#).

Technology View ()

Select HDL Analyst ->Technology->Hierarchical View, or ->Flattened View to display the mapped view of the design. See [Chapter 15, Analyzing with HDL Analyst and FSM Viewer](#).

Physical Analyst ()

The Physical Analyst provides a visual display of the device, and design placement of instances and nets. Select HDL Analyst->Physical Analyst. See [Chapter 16, Analyzing Designs in Physical Analyst](#).

Timing Analyst ()

The stand-alone timing analyzer produces timing reports (.ta) for specific reporting requirements. See [Using the Stand-alone Timing Analyst, on page 736](#).

Use these guidelines to analyze the results:

Check this...**Tool**

Are start and end points being constrained by the proper clocks?

Timing report
You can also trace the clock network using HDL Analyst Technology view.

Is the critical path a multi-cycle path or false path?

Timing report
HDL Analyst

Will pipelining improve results?

HDL Analyst

If the path is inside a state machine, is the FSM being fully optimized?

HDL Analyst. Open the RTL view and push down into the state machine module to display the FSM viewer.

Are the net delays contributing to the highest percentage on the critical path?

Timing report
Check the % breakdown of delay for each path. Search for **Total path delay**.
Physical Analyst
Use it to analyze the instance placement of the critical path.

Will physical constraints improve results?

Physical Analyst
Use it to analyze the design.
Design Planner
Use it to assign logic to regions and generate a design plan file.

Guidelines for Xilinx Timing Constraints for Physical Synthesis

The Synplify Premier tool requires that you provide accurate and complete timing constraints to run physical synthesis effectively. The Synplify Premier software outputs a placed design, for which the place-and-route tool cannot correct constraints used inaccurately during physical synthesis. For example, a false path constraint provided only to the place-and-route tool allows the Synplify Premier software to move components affected by the false path far apart to resolve critical path issues.

It is extremely important that you verify timing constraints. Use the following guidelines:

- Define all clocks.
- Assign realistic, accurate timing constraints. Do not over-constrain the tool.
- Assign clocks to the correct clock group. Clocks assigned to separate groups are cross-clock paths, which are treated as false paths.
- Specify all multicycle paths.
- Specify all false paths.
- Specify all input and output delays.
- Ensure that all I/Os have I/O standards and drive strengths specified.
- Top-level clocks that do not use DCM/PLL should have the constraint specified on the port to ensure insertion delay is modeled correctly.
- Clocks derived through the DCM/PLL should have the constraint specified on the port driving the DCM/PLL, if possible.
- Make sure constraints are valid. Check for the following types of messages in the log file:
 - Cannot find object `<clock>` to apply `define_clock`.
 - Timing constraint `<x> <y>` never applies in design and was not found.

You can translate UCF constraints as described in [Converting and Using Xilinx UCF Constraints](#), on page 255, but you might need to manually review it and make sure this information is complete.

Using IP Cores in Xilinx Physical Synthesis Flows

The following is a summary of the information on using IP cores in a physical synthesis flow:

- You cannot have black box IP, as the flows do not support this.
- You can use secure and non-secure IP edn, ngc, and ngo cores. See [Including Xilinx Cores for Logic and Physical Synthesis, on page 196](#) and [Working with EDK Cores, on page 204](#).

Placement and Routing Phases in Xilinx Physical Synthesis

The following Synplify Premier processes are described here:

- [Global Placement](#)
- [Detail Placement](#)
- [Routing](#)

Global Placement

The 9.0.1 and later versions of the Synplify Premier tool use the Xilinx placer to generate locations for I/Os and block components. Global placement spreads out the design evenly on the chip. This is especially important for large block RAM and DSP48 components because physical synthesis does not move them. To avoid block component placement problems, you need to lock placement with a coreloc file. See [Generating a Xilinx Coreloc Placement File, on page 354](#).

Detail Placement

The detail placer performs complete legality checks for placed components to ensure critical paths are routed optimally. It also performs some local routing and optimizations.

The Xilinx Virtex-5 CLB placement and packing rules are complex. Despite extensive testing, the Synplify Premier flow might still encounter placement violation rules for this device. If you encounter a problem, report it to technical support and:

- Include the Xilinx map log file, Synplify Premier *.aux files, and .srm netlist file to view the problem slice.

- You can also report the problem from the Technology view using HDL-Analyst->Technology->Flattened View. For example, specify the following commands to select components placed in the problem slice:

```
set x [find -hier -inst * -filter @location==SLICE_X23_Y54  
select $x  
filter
```

Or you can use the Filter Schematic icon to create a schematic with just the instances for this slice. Send this schematic.

Routing

In the Synplify Premier flow final routing is implemented by the Xilinx place-and-route tool. The Synplify Premier tool only performs local routing during detail placement and optimization. In Synplify Premier 9.0.1 and later versions, local routing is forward-annotated as initial pin assignments on LUTs. Placement and routing can change pin assignments to accommodate longer distance routing, but generally result in equivalent path delays.

The default routing effort level is recommended for most designs to route successfully with good timing closure. However, if high density routing causes routing detours, increase the effort level to enable extra effort using the `-sc c` switch. You can detect routing detours in the following ways:

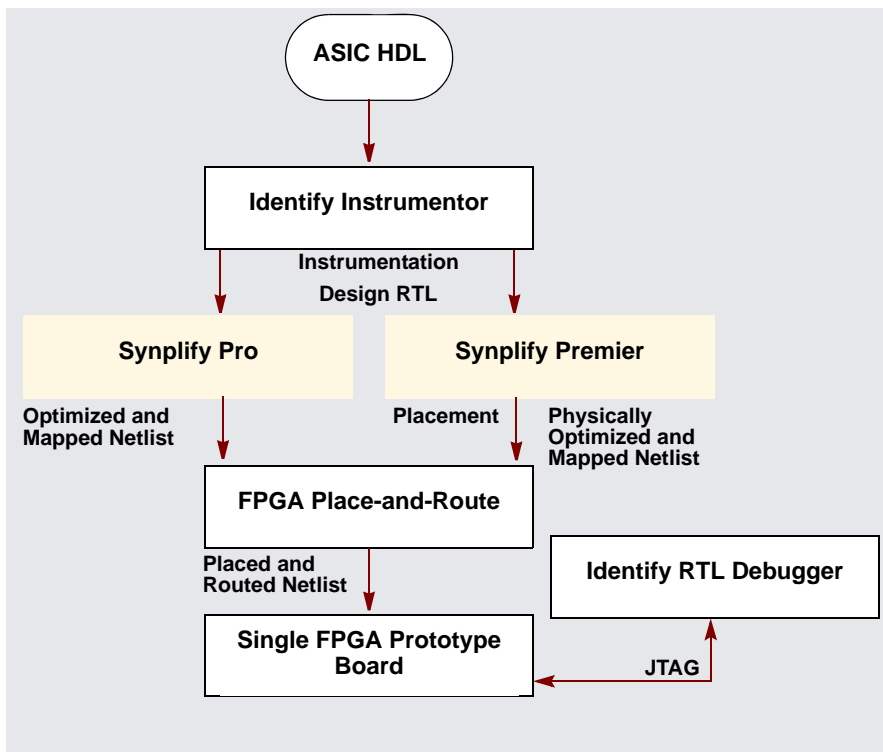
- Look for messages in the log file about congested designs and switching to a non-timing-driven mode.
- Open the Xilinx FPGA Editor and select the long delay nets. When wires show an indirect path to the load, then the design probably has detours.

If detours happen on high fanout nets, set the `MAXSKEW` option on the net to a fairly loose value, such as 1.0 ns. This setting uses a different router algorithm with higher priority on the applied net. If the design is still congested, then contact technical support.

Prototyping Design Flow

The Synplify Pro and Synplify Premier tools support a complete design and verification environment that features the Identify RTL Debugger product and automated HDL code translation. You can run the Identify tool from the synthesis interface.

You can use the flow shown in the following figure for single FPGA prototypes. For partitioning and timing optimizations in multi-FPGA designs, use the Certify product.



Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA
 Phone: +1 408 215-6000, Fax: +1 408 222-068
www.solvnet.com

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Conforma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

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 82
- [Using Mixed Language Source Files](#), on page 95
- [Working with Constraint Files](#), on page 98
- [Using Input from Related Tools](#), on page 106
- [Converting Synopsys DesignWare Components](#), on page 107

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 270](#). Source files can be in Verilog or VHDL. For information about structuring the files for synthesis, refer to the *Reference Manual*. This section discusses the following topics:

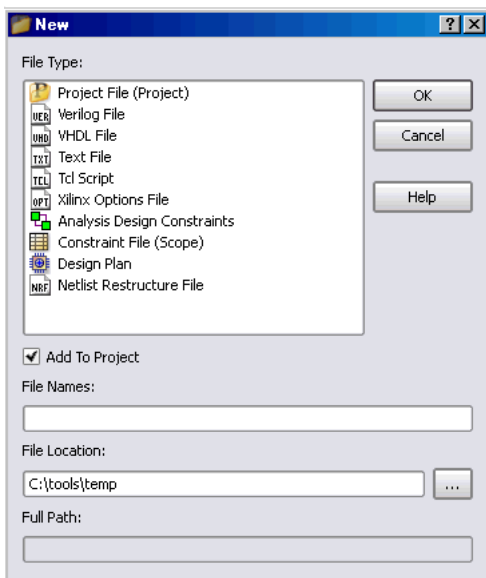
- [Creating HDL Source Files](#), on page 82
- [Checking HDL Source Files](#), on page 83
- [Editing HDL Source Files with the Built-in Text Editor](#), on page 85
- [Using an External Text Editor](#), on page 90
- [Setting Editing Window Preferences](#), on page 88
- [Using Hyper Source](#), on page 91

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. For details of what you can and cannot include, as well as vendor-specific information, see the *Reference Manual*. 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 83](#) and [Editing HDL Source Files with the Built-in Text Editor, on page 85](#)).

You can use Verilog or VHDL for your source files. The files have `.v` (Verilog) or `.vhd` (VHDL) file extensions, respectively. With the Synplify Premier and Synplify Pro products, 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 95](#).

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).



- Type a name and location for the file and Click OK. A blank editing window opens with line numbers on the left.
- 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 85](#) for more information on working in the Editing window.

For the best synthesis results, check the *Reference Manual* and ensure that you are using the available 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 83](#).

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. An exclamation mark next to a file in the project list indicates that it has errors or warnings. The number of warnings is listed after the file name. If there are no errors, the following message is displayed at the bottom of the log file:

```
Syntax check successful!
```

3. To run a synthesis check, select Run->Synthesis Check or press Shift+F8.

The software detects hardware-related errors such as incorrectly coded flip-flops. It puts an exclamation mark next to files in the project list that have errors or warnings, and lists the number of errors, warnings or notes found in each file. If there are no errors, the following message is displayed at the bottom of the log file:

```
Synthesis check successful!
```

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.

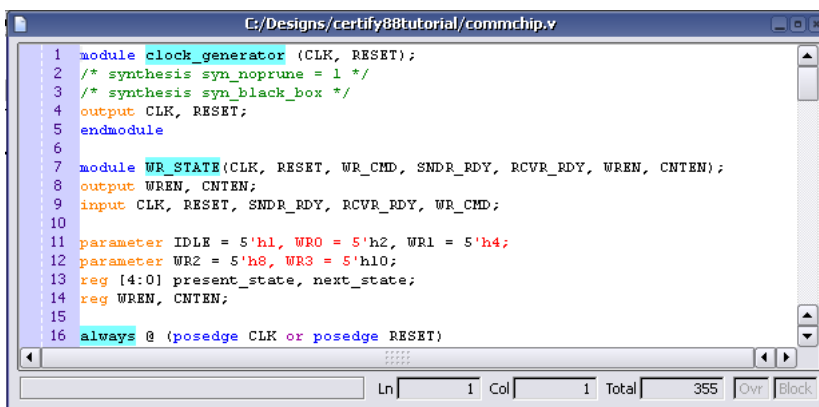
Messages can be categorized as errors, warnings, or notes. Review all messages and resolve any errors. Warnings are less serious than errors, but you must read through and understand them even if you do not resolve all of them. Notes are informative and do not need to be resolved.

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 90](#).

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 88](#).



```
C:/Designs/certify88tutorial/commchip.v
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)
```

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.

- 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.
- 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->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->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->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 (not available with the Synplify product) and click on the 5-character error code.
 - In the Tcl window (not available with the Synplify product), click the Messages tab and click on the 5-character error code in the ID column.
7. To crossprobe 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 649](#) 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.

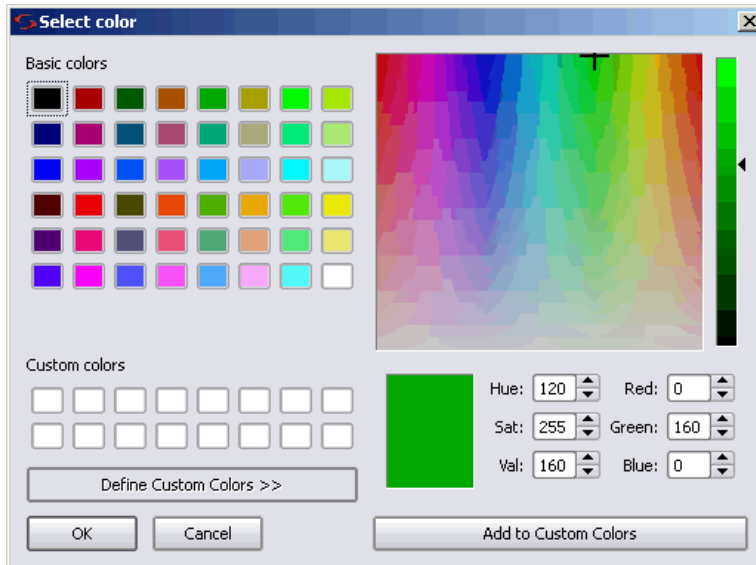
1. Select Options->Editor Options and either Synplicity Editor or External Editor. For more information about the external editor, see [Using an External Text Editor, on page 90](#).
2. Then depending on the type of file you open, you can to set the background, syntax coloring, and font preferences to 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.

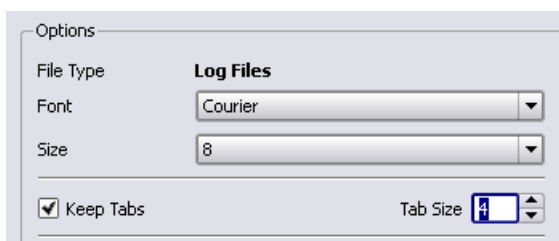
3. You can set syntax colors for some common syntax options, such as keywords, strings, and comments. For example in the log file, warnings and errors can be color-coded for easy recognition.

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.

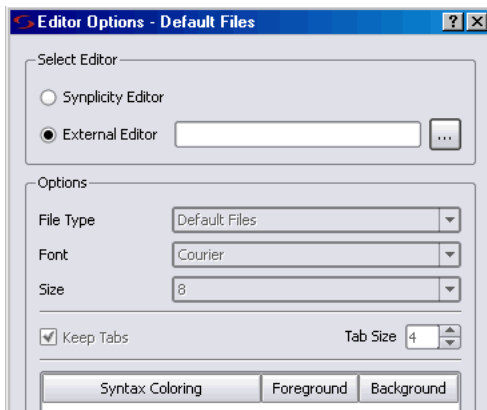


6. Click OK on the Editor Options form.

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 85](#).

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 Hyper Source

Use this mechanism to thread a signal through the design hierarchy of a user IP. This signal can be threaded to a top-level port or signal. This works even if the Verilog or VHDL is compiled separately. Ports and signals will automatically be added between the source and the connection. Otherwise, these connections must be manually added to the RTL code.

Refer to the HDL [Hyper Source Example, on page 92](#) below. The following procedure describes a method for using hyper source.

1. Define how to connect to the signal source. In this case, the:
 - Signal `syn_hyper_source (in1)` module is defined for the source with a width of 1.
 - Label name of "tag_name" is the global name for the hyper source.
2. Define how to access the hyper source which drives the local signal or port. In this case, the:
 - Signal `syn_hyper_connect (out1)` module is defined for the connection. The signal width of 1 must match the source.
 - Label name can be the global name or the instance path to the hyper source.
3. In this hierarchical design, the hyper source:
 - Applies to the module `lower_module`.
 - Signal `syn_hyper_source my_source(din)` module is defined for the source with a width of 8.
 - Label name of "probe_sig" must match the name used in the hyper connect block to thread the signal properly.
4. In this hierarchical design, the hyper connect:
 - Applies to the top-level module `top`, but can be any level of hierarchy.
 - Signal `syn_hyper_connect connect_block(probe)` module is defined for the connection with a width of 8.
 - Label name of "probe_sig" must match the name used in the hyper source block to thread the signal properly.

5. After you run synthesis, the following message appears in the log file:

```
Available hyper_sources - for debug and ip models
HyperSrc label sub2_module.sub1_module.lower_module.probe_sig

Making connections to hyper_source modules
@N: : hyper\_example.v\(54\) | Connected syn_hyper_connect hstcm_training_done_connect, label probe_sig
Finished RTL optimizations (Time elapsed 0h:00m:01s; Memory used current: 122MB peak: 129MB)
```

Hyper Source Example

```
/* connect to a signal you want to export example : in1*/
module syn_hyper_source(in1) /*synthesis syn_black_box=1 syn_noprune=1 */;
parameter w = 1;
parameter label = "tag_name"; /* global name of hyper_source */
input [w-1:0] in1;
endmodule

/* use to access hyper_source and drive a local signal or port example
:out1 */
module syn_hyper_connect(out1) /* synthesis syn_black_box=1 syn_noprune=1
*/;
parameter w = 1; /* width must match source */
parameter label = "tag_name"; /* global name or instance path to
hyper_source */
parameter dflt = 0;
parameter mustconnect = 1'b1;
output [w-1:0] out1;
endmodule

/* Example hierarchical design which uses hyper_source */
module lower_module (clk, dout, din1, din2, we);
output reg [7:0] dout;
input clk, we;
input [7:0] din1, din2;
wire [7:0] din;

syn_hyper_source my_source(din);
defparam my_source.label = "probe_sig"; /* to thread the signal this
tag_name must match to name used in the hyper connect block */
defparam my_source.w = 8;

always @(posedge clk)
if (we)
dout <= din;
assign din = din1 & din2;
```



```
endmodule

module sub1_module (clk, dout, din1, din2, we);
output[7:0] dout;
input clk, we;
input [7:0] din1, din2;
lower_module lower_module (clk, dout, din1, din2, we);
endmodule

module sub2_module (clk, dout, din1, din2, we);
output [7:0] dout;
input clk, we;
input [7:0] din1, din2;
sub1_module sub1_module (clk, dout, din1, din2, we);
endmodule

module top (clk, dout, din1, din2, we, probe);
output[7:0] dout;
output [7:0] probe;
input clk, we;
input [7:0] din1, din2;

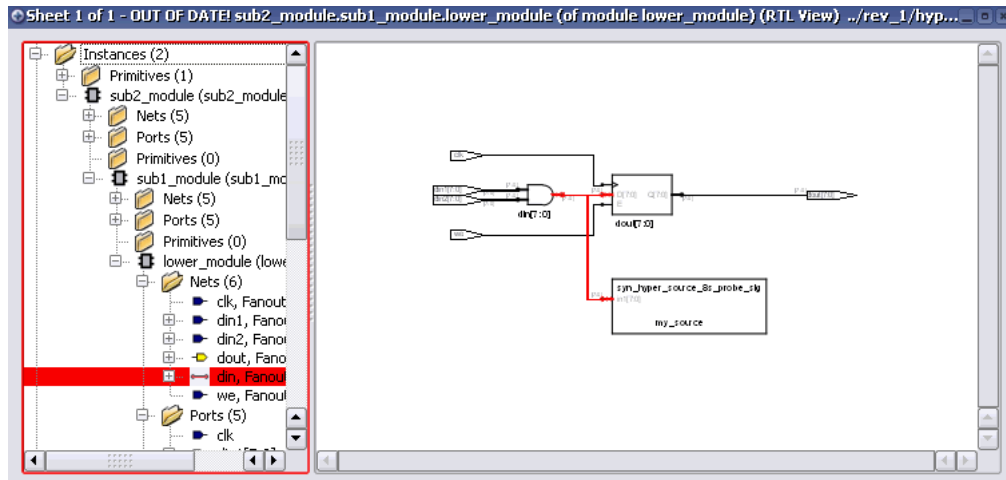
syn_hyper_connect connect_block(probe);
defparam connect_block.label = "probe_sig"; /* to thread the signal this
tag_name must match to name used in the hyper connect block */
defparam connect_block.w = 8;

sub2_module sub2_module (clk, dout, din1, din2, we);

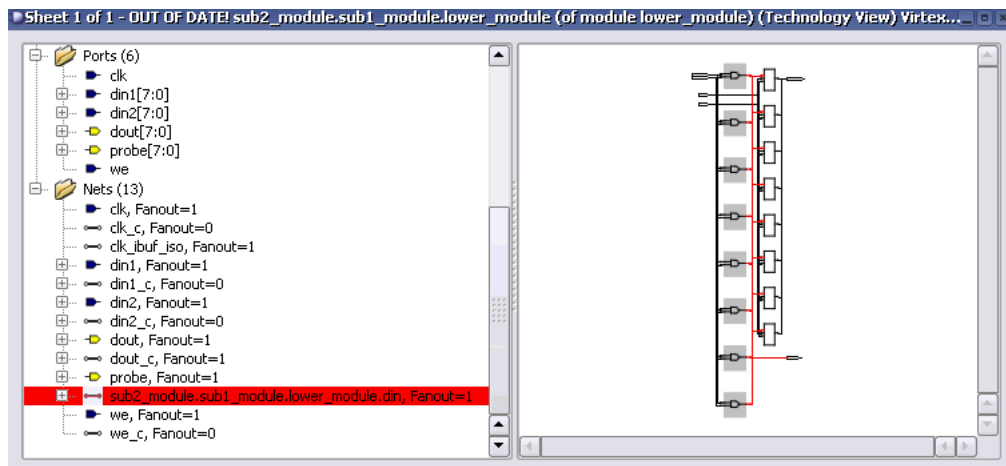
endmodule
```

The following figures show how the hyper source signal automatically gets connected through the hierarchy of the IP in the HDL Analyst views.

RTL View



Technology View



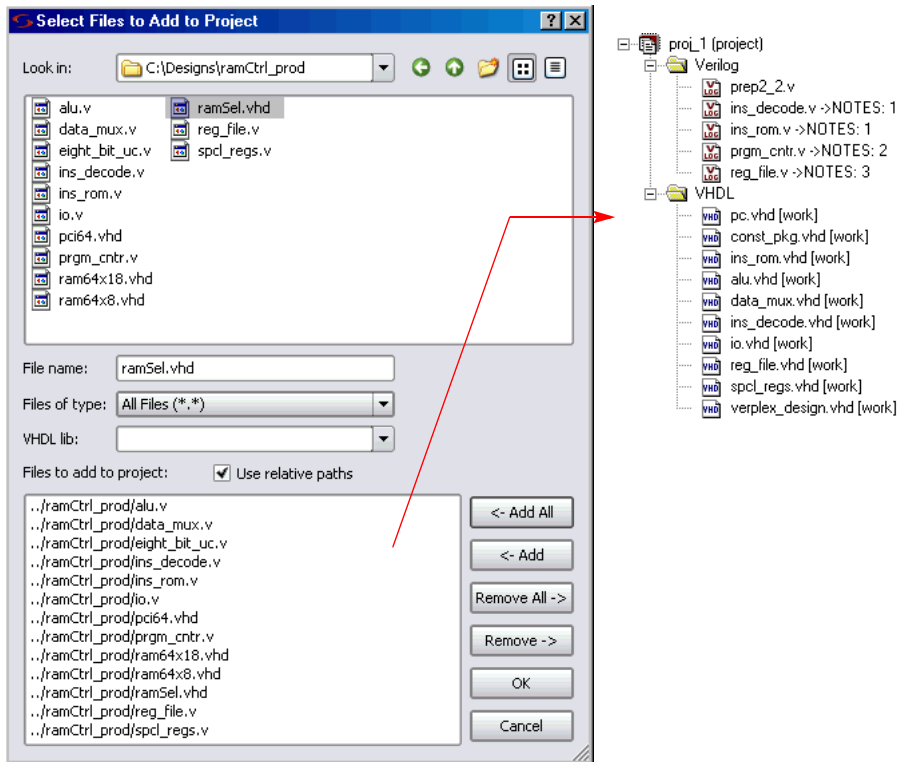
Using Mixed Language Source Files

With the Synplify Pro and Synplify Premier 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 these restrictions and set up the mixed language design files accordingly:
 - You can not use defparams across languages.
 - Verilog does not support unconstrained VHDL ports
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 274](#).



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 289](#).
 - 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 148](#).

The image shows two screenshots of a configuration window. The top screenshot is for the Verilog tab. It has a 'Top Level Module' field containing 'dna_system_test'. Below it, under 'Verilog Language', there are two checked checkboxes: 'Verilog 2001' and 'System Verilog'. To the right, under 'Compiler Directives and Parameters', there is a table with two columns: 'Parameter Name' and 'Value'. The table is currently empty. Below the table is an 'Extract Parameters' button. The bottom screenshot is for the VHDL tab. It has a 'Top Level Entity' field containing 'dna_system_test'. To its right, 'Default Enum Encoding' is set to 'sequential'. Below these are two checked checkboxes: 'Push Tristates' and 'Synthesis On/Off Implemented as Translate On/Off'.

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 289](#).
 - 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.

Working with Constraint Files

Constraint files are text files that are automatically generated by the SCOPE interface (see [Specifying Timing Constraints, on page 219](#)), 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 it is not the preferred method.

This section contains information about

- [When to Use Constraint Files over Source Code](#), on page 98
- [Tcl Syntax Guidelines for Constraint Files](#), on page 99
- [Using a Text Editor for Constraint Files](#), on page 100
- [Using Synopsys Design Compiler Constraints](#), on page 102
- [Checking Constraint Files](#), on page 104
- [Generating Constraint Files for Forward Annotation](#), on page 105

When to Use Constraint Files over Source Code

You can add constraints in constraint files (generated by 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.

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 [Entering Attributes and Directives, on page 304](#) 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*:]*object_name*

Where *cell* is the name of the design entity, *prefix* is a prefix to identify objects with the same name, *object_name* 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*:]*object_name*

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 *object_name* 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[*]
```

Using a Text Editor for Constraint Files

This section shows you how to manually create a Tcl constraint file. The software automatically creates this file if you use the SCOPE interface to enter the constraints. The Tcl constraint file only contains general timing constraints. Black box constraints must be entered in the source code. For details of the Tcl commands, refer to the *Reference Manual*. For additional information, see [When to Use Constraint Files over Source Code, on page 98](#).

1. Open a file for editing.
 - Make sure you have closed the SCOPE window, or you could overwrite previous constraints.
 - To create a new file, select File->New, and select the Constraint File (SCOPE) option. Type a name for the file and click OK.
 - To edit an existing file, select File->Open, set the Files of Type filter to Constraint Files (.sdc) and open the file you want.
2. Follow the syntax guidelines in [Tcl Syntax Guidelines for Constraint Files, on page 99](#).

3. Enter the timing constraints you need. For the syntax, see the *Reference Manual*. If you have black box timing constraints, you must enter them in the source code.

To define...	Use...
Clock frequencies	define_clock. See Defining Clocks, on page 222 for additional information.
Clock frequency other than the one implied by the signal on the clock pin	syn_reference_clock (attribute). See Defining Clocks, on page 222 for additional information
Clock domains with asymmetric duty cycles	define_clock. See Defining Clocks, on page 222 for additional information
Edge-to-edge clock delays	define_clock_delay. See Defining Clocks, on page 222 for additional information
Speed up paths feeding into a register	define_reg_input_delay.
Speed up paths coming from a register	define_reg_output_delay.
Input delays from outside the FPGA	define_input_delay. See Defining Input and Output Constraints, on page 226 for additional information
Output delays from your FPGA	define_output_delay. See Defining Input and Output Constraints, on page 226 for additional information
Paths with multiple clock cycles	define_multicycle_path. See Defining Multi-cycle Paths, on page 234 for additional information
False paths (certain technologies)	define_false_path. See Defining False Paths, on page 235 for additional information.
Path delays	define_path_delay. See Defining From/To/Through Points for Timing Exceptions, on page 231 for additional information

The following code excerpt shows some typical Tcl constraints:

```
# Override the default frequency for clk_fast and set it to run
# at 66.0 MHz.
define_clock {clk_fast} -freq 66.0
```

```
# Set a default input delay of 4 ns
define_input_delay -default 4.0

# Except for the "sel" signal, which has an input delay of 8 ns
define_input_delay {sel} 8.0

# The outputs have an off-chip delay of 3.0 ns
define_output_delay -default 3.0
```

4. You can also add vendor-specific attributes in the constraint file using `define_attribute`. See [Specifying Attributes in the Constraints File \(.sdc\)](#), on page 310 for more information.
5. Save the file.
6. Add the file to the project as described in [Making Changes to a Project](#), on page 274, and run synthesis.

Using Synopsys Design Compiler Constraints

The FPGA synthesis tools can read some native Design Compiler (SDC) constraint files for a supported set of clock definition, I/O delay, and timing exception constraints. To use these constraints for FPGA synthesis, do the following:

1. If you are importing constraints in the Design Compiler SDC format, do not use other constraints in the FPGA synthesis format.
2. Add the Design Compiler file to your project.
3. Run the constraint checker, as described in [Checking Constraint Files](#), on page 104 and edit the constraint files as needed.
4. Edit your Design Compiler file if needed.
 - Make sure the object identifiers map as expected:

Synopsys Design Compiler Identifiers	FPGA Synthesis Identifiers
get_clocks (Wildcards are not supported)	c:
get_registers	r:
get_nets	n:

Synopsys Design Compiler Identifiers	FPGA Synthesis Identifiers
get_ports	p:
get_cells	i:
get_pins	t:

- If you use different naming conventions than the default FPGA assumptions, add the appropriate command to the beginning of your file. You must add it to the beginning of the file, so that it is read first. The following table lists the default FPGA naming convention and the corresponding Design Compiler rule.

	FPGA	Design Compiler Rule
Hierarchy separator	.	set_hierarchy_separator { . }
Register names	_reg	set_rtl_ff_names { _reg }
Bus names	[]	bus_naming_style { %s[%d] }
Bus array separator	[] []	bus_dimension_separator_style { [] [] }

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 { _ }

The FPGA synthesis tool accepts the following native Design Compiler constraints, and uses them to run synthesis:

all_clocks	set_false_path
all_inputs	set_input_delay
all_outputs	set_max_delay
all_registers	set_multicycle_path
create_clock	set_output_delay
create_generated_clock	

Checking Constraint Files

For certain technologies, you can check your constraints and their syntax. Do the following:

1. Make sure your target is a technology that supports this feature.
2. Generate a constraint file.
3. Select Run->Constraint Check.

This command generate a report that checks the syntax and applicability of the timing constraints in the `.sdc` file(s) for your project. The report is written to the `project_name_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 a description of this file, see [Constraint Checking Report, on page 335](#) of the *Reference Manual*.

Generating Constraint Files for Forward Annotation

The tool automatically generates vendor-specific constraint files that you can use for forward-annotation. The synthesis constraints are mapped to the appropriate vendor constraints. You can control this process with some attributes as described in the following procedure.

1. Set attributes to control forward annotation.
 - To forward-annotate timing constraints for Actel Axcelerator, Fusion, ProASIC (500K), ProASIC Plus (PA), and ProASIC3/3E /3L families, set the clock period, max delay, input delay, output delay, multiple-cycle paths, and false paths in the SCOPE interface.
 - To forward-annotate I/O constraints (`define_input_delay` and `define_output_delay`) to the `.tcl` file for APEX designs or the `synplicity.ucf` file for Xilinx designs, set `syn_forward_io_constraints` with a value of 1 on the top level of the design or as a global attribute.
 - To forward-annotate clocks for Xilinx DCMs and DLLs, define the clock at the primary inputs and any Xilinx phase shift and frequency multiplication properties you need. See [Defining Other Clock Requirements, on page 225](#) for details. The synthesis software forward-annotates the DLL/DCM inputs.
 - To forward-annotate clocks for Altera PLLs, define the input frequency value. See [Defining Other Clock Requirements, on page 225](#) for details. The synthesis software forward-annotates the PLL inputs.
 - For some Lattice designs, set the `-from` and `-to` false path and multi-cycle constraints on the Others tab of the SCOPE interface.

For details about these attributes, see the *Reference Manual*.

2. Select Project->Implementation Options, and check Write Vendor Constraints in the Implementation Results tab.

Currently you can forward-annotate constraints for some vendors only.

3. Click OK and run synthesis.

The software converts the synthesis `define_input_delay`, `define_output_delay`, `define_clock` (including the `define_clock` constraints generated by auto constraining), `define_multicycle_path`, `define_false_path`, `define_max_delay`, and global frequency constraints into corresponding commands in the following files:

- *.acf file for Altera
- *filename_sdc.sdc* file for Actel
- *synplicity.ucf* file for Xilinx
- \$DESIGN_synplify.lpf file for Lattice

Open the Lattice ispLEVER place-and-route tool, then import the \$DESIGN_synplify.lpf file before running PAR. If a user-created *.lpf file already exists, ispLEVER backs it up into *.lpf.bak and copies the contents of \$DESIGN_synplify.lpf into the \$DESIGN.lpf file which is then used for all computations.

See the *Reference Manual* for details about forward annotation.

Using Input from Related Tools

The following show you how to incorporate input from other tools like System Designer and Synplify DSP.

Using Input from System Designer

You use System Designer to import and stitch together IP. To incorporate these components in your design, include the following files generated by System Designer in your synthesis project:

- Synplify sub-library file *synthesis.slib*
- HDL files
VHDL files in *vhdl* folder and Verilog files in the *verilog* folder in the project directory.

Using Input from Synplify DSP

The Synplify DSP tool generates IP for DSP designs. Include this in your synthesis project by adding the source files generated by the Synplify DSP tool.

Converting Synopsys DesignWare Components

The Synplify Premier and Synplify Premier with Design Planner tools can convert supported Synopsys DesignWare components that have been instantiated in your VHDL or Verilog source code by using DW-compatible models. For details on supported models and syntax, see [Translating DesignWare Components](#), on page 546 of the *Reference Manual*. For information on how to instantiate the supported models, see

- [Converting Verilog Library Components](#), on page 107
- [Converting VHDL Library Components](#), on page 109

Converting Verilog Library Components

This section describes the following:

- [Instantiating and Compiling Verilog Components](#), on page 107
- [Inferring Verilog Functions](#), on page 108

Instantiating and Compiling Verilog Components

To instantiate the components, enable access to the Verilog DW-compatible module library (`dw_verilog.v`) by :

1. Opening the Implementation Options dialog box in the Project view.
2. Going to the Options tab and enabling the Compile with Designware Library (`dw_verilog.v`) switch to automatically use the `dw_verilog.v` file during compilation.
3. To replace an existing DW-compatible module with your own module in the `dw_verilog.v` library, add the file to your project file. For example:

```
add_file -verilog "my_DW_component"
```

This command tells the Synplify Premier tool to use your version of the module. The module name in `my_DW_component` must be the same name as the component you want to replace. For a list of compatible models and their names, see [Supported Components – by Function](#), on page 546 of the *Reference Manual*.

4. Before you compile your project, enable the V2001 switch by doing one of the following:
 - Check the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box.
 - Adding the following command in the #add_file options section of your project file: `set_option -vlog_std v2001`.

The `dw_verilog.v` file uses Verilog 2001 constructs, so you must enable this option before compiling.

5. Compile and synthesize the design as usual.

The models extract the functionality of the component, but not its implementation. The mappers synthesize the model to the most appropriate implementation. If there is no DW-compatible model defined for a DesignWare component in your source code, you get an error message. For a list of compatible models and their names, see [Supported Components – by Function, on page 546](#) of the *Reference Manual*.

Inferring Verilog Functions

You can infer functions for a subset of the DW-compatible models. For a list of the supported functions, see [Supported Components – by Function, on page 546](#) of the *Reference Manual*. To use this method, add the directory containing the DW-compatible functions (`dw_functions`) to your include path, as described in the following procedure. You can either use the GUI method described in step 1 or the command line method described in step 2.

1. To set up function inferencing from the GUI, do the following:
 - Open the Implementation Options dialog box in the Project view.
 - Go to the Verilog tab and type the following path in the Include Path Order field:

```
install_dir/lib/designware/dw_functions
```

2. To set up function inferencing from the command line, add the following line to your project file:

```
set_option -include_path  
"install_dir/lib/designware/dw_functions"
```

3. Synthesize the design.

Converting VHDL Library Components

For VHDL designs, this section provides setup information so that the Synplify Premier or Synplify Premier with Design Planner tool can convert DesignWare components to DW-compatible models.

- [Instantiating VHDL Components](#), on page 109
- [Creating and Editing VHDL Component Libraries](#), on page 109

Instantiating VHDL Components

The VHDL DesignWare components require the corresponding Verilog components. To include the Verilog DW-compatible module library in your project:

1. Open the Implementation Options dialog box in the Project view, go to the Options tab, and enable the Compile with Designware Library (dw_verilog.v) switch.
2. Specify the top module:

```
set_option -top_module "module_name"
```

You must specify the top module for mixed HDL designs.

3. Add the VHDL component libraries. For information about creating and editing the VHDL component libraries, see [Creating and Editing VHDL Component Libraries](#), on page 109.

Creating and Editing VHDL Component Libraries

The following procedure shows you how to create and edit a VHDL library of DesignWare-compatible models. For a list of supported DesignWare components, see [Supported Components – by Function](#), on page 546 of the *Reference Manual*.

1. To create and add a new library, use the following command syntax:

```
add_file -library newlib  
add_file -vhdl -lib newlib dw_name.vhd
```

For example, the following defines a new library my_lib and compiles the component dw02_comp.vhd into this library:

```
add_file -library my_lib
add_file -vhdl -lib my_lib dw02_comp.vhd
```

2. To add a new set of DesignWare objects, such as entities and or architectures to an existing library, use this command syntax:

add_file -vhdl -lib *existing_lib* "*new_object*"

The following command appends information for the new DW_square component to the DW01 library:

```
add_file -vhdl -lib DW01 "DW_square.vhd"
```

3. To add a missing DesignWare component declaration to an existing package, do the following:
 - First add the missing component declaration to the library using this command syntax:

add_file -vhdl -lib *library_name* "*component_pkg.vhd*"

The following example adds the missing DW_square component to the dw01_comp.vhd library:

```
add_file -vhdl -lib DW01 "DW_square.vhd"
```

- Add the component declaration to the vhd package by adding a line like the following to the project file. This example adds the component declaration for DW_square in dw01_comp.vhd package. The DW01_components package is declared in the file by name: dw01_comp_add.vhd

```
add_file -vhdl -lib DW01 "DW01_comp_add.vhd"
```

4. To replace a component declaration in the existing package with a new declaration, add a line like the following:

```
add_file -vhdl -lib DW01 "DW01_comp_replace.vhd"
```

The example shows a new declaration for a component, DW01_comp_replace.vhd, in the DW01 library. For a list of supported DesignWare components, see [Supported Components – by Function, on page 546](#) of the *Reference Manual*.

5. To replace an existing entity or architecture component in a library with a component you define, do the following:

- Write your code for the component, and give the module the same name as the module you want to replace. For example, if you want to replace an existing component called DW_square, name your custom design DW_square.vhd.
- Add a statement like the following to the project file to override the existing module:

```
add_file -vhd1 -lib DW02 DW_square.vhd
```

Your DW_square module overwrites the existing module in the DW02 library.

6. If your design references libraries or packages that are not included, create a dummy package and add it to your project file.

This ensures that the compiler ignores the package and you do not get a compiler error because the tool cannot find a library.



Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA
Phone: +1 408 215-6000, Fax: +1 408 222-068
www.solvnet.com

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

CHAPTER 4

Working with IP Input

This chapter describes how to work with IP from different sources. It describes the following:

- [Generating IP with SYNCORE](#), on page 114
- [The ReadyIP Encryption Flow](#), on page 143
- [Working with Encrypted IP](#), on page 148
- [Working with Altera IP](#), on page 161
- [Working with Lattice IP](#), on page 194
- [Working with Xilinx IP](#), on page 195
- [Including Xilinx EDK Cores](#), on page 199


Generating IP with SYNCORE

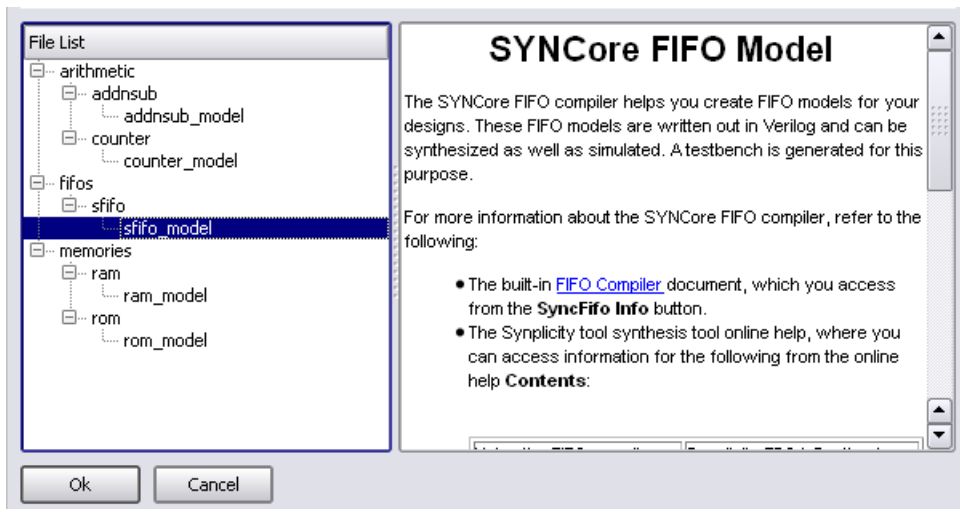
You can use the SYNCORE IP wizard to generate FIFO, RAM, ROM, adder/subtractor, and counter implementations. See the following for more information.

- [Specifying FIFOs with SYNCORE](#), on page 114
- [Specifying RAMs with SYNCORE](#), on page 119
- [Specifying ROMs with SYNCORE](#), on page 125
- [Specifying Adder/Subtractors with SYNCORE](#), on page 130
- [Specifying Counters with SYNCORE](#), on page 137

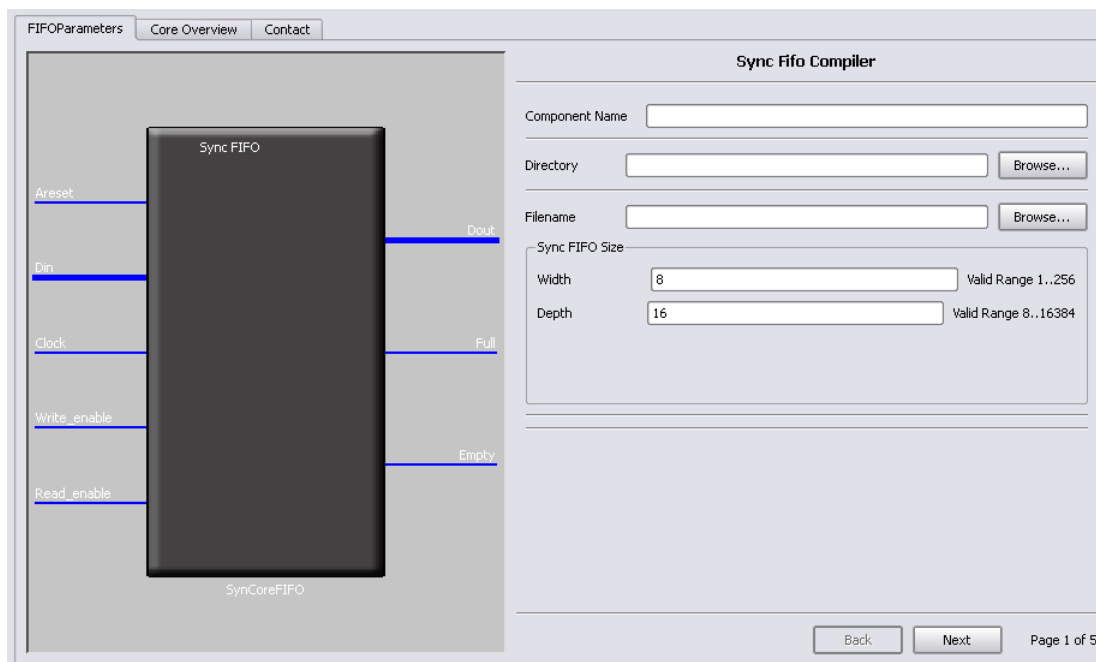
Specifying FIFOs with SYNCORE

The SYNCORE IP Wizard helps you generate Verilog code for your FIFO implementations. The following procedure shows you how to generate Verilog code for a FIFO you specify, using the SYNCORE IP wizard.

1. Start the wizard.
 - From the synthesis tool GUI, select Run->Launch SYNCORE or click the Launch SYNCORE icon  to start the SYNCORE IP wizard.



- In the window that opens, select `sffifo_model` and click Ok. This opens the first screen of the wizard.



2. Specify the parameters you need in the five pages of the wizard. For details, refer to [Specifying SYNCore FIFO Parameters, on page 117](#).

The FIFO symbol on the left reflects the parameters you set.

3. After you have specified all the parameters you need, click the Generate button (lower left).

The tool displays a confirmation message (TCL execution successful!) and writes the required files to the directory you specified in the parameters. The HDL code is in Verilog.

The FIFO generated is a synchronous FIFO with symmetric ports and with the same clock controlling both the read and write operations. Data is written or read on the rising edge of the clock. All resets are synchronous with the clock. All edges (clock, enable, and reset) are considered positive.

SYNCORE also generates a testbench for the FIFO that you can use for simulation. The testbench covers a limited set of vectors for testing.

You can now close the SYNCORE wizard.

4. Add the FIFO you generated to your design.
 - Use the Add File command to add the Verilog design file that was generated and the `syncore_sfifo.v` file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
 - Use a text editor to open the `instantiation_file.vin` template file, which is located in the same directory. Copy the lines that define the memory, and paste them into your top-level module. The following shows a template file (in red text) inserted into a top-level module.

```
module top (
input Clk,
input [15:0] DataIn,
input WrEn,
input RdEn,
output Full,
output Empty,
output [15:0] DataOut
);
```

```
fifo_a32 <instanceName>(
.Clock(Clock)
, .Din(Din)
, .Write_enable(Write_enable)
, .Read_enable(Read_enable)
, .Dout(Dout)
, .Full(Full)
, .Empty(Empty)
)
```

template

- ```
endmodule
```
- Edit the template port connections so that they agree with the port definitions in the top-level module as shown in the example below. You can also assign a unique name to each instantiation.



```
module top (
 input Clk,
 input [15:0] DataIn,
 input WrEn,
 input RdEn,

 output Full,
 output Empty,
 output [15:0] DataOut
);

 fifo_a32 busfifo(
 .Clock(Clk)
 , .Din(DataIn)
 , .Write_enable(WrEn)
 , .Read_enable(RdEn)
 , .Dout(DataOut)
 , .Full(Full)
 , .Empty(Empty)
)
endmodule
```

Note that currently the FIFO models will not be implemented with the dedicated FIFO blocks available in certain technologies.

## Specifying SYNCore FIFO Parameters

The following elaborates on the parameter settings for SYNCore FIFOs. The status, handshaking, and programmable flags are optional. For descriptions of the parameters, see [SYNCore FIFO Wizard, on page 190](#) in the *Reference Manual*. For timing diagrams, see [SYNCore FIFO Compiler, on page 561](#) in the *Reference Manual*.

1. Start the SYNCore wizard, as described in [Specifying FIFOs with SYNCore, on page 114](#).
2. Do the following on page 1 of the FIFO wizard:
  - In Component Name, specify a name for the FIFO. Do not use spaces.
  - In Directory, specify a directory where you want the output files to be written. Do not use spaces.

- In Filename, specify a name for the Verilog output file with the FIFO specifications. Do not use spaces.
  - Click Next. The wizard opens another page where you can set parameters.
3. For a FIFO with no status, handshaking, or programmable flags, use the default settings. You can generate the FIFO, as described in [Specifying FIFOs with SYNCORE, on page 114](#).
  4. To set an almost full status flag, do the following on page 2 of the FIFO wizard:
    - Enable Almost Full.
    - Set associated handshaking flags for the signal as desired, with the Overflow Flag and Write Acknowledge options.
    - Click Next when you are done.
  5. To set an almost empty status flag, do the following on page 3:
    - Enable Almost Empty.
    - Set associated handshaking flags for the signal as desired, with the Underflow Flag and Read Acknowledge options.
    - Click Next when you are done.
  6. To set a programmable full flag, do the following:
    - Make sure you have enabled Full on page 2 of the wizard and set any handshaking flags you require.
    - Go to page 4 and enable Programmable Full.
    - Select one of the four mutually exclusive configurations for Programmable Full on page 4. See [Programmable Full, on page 572](#) in the *Reference Manual* for details.
    - Click Next when you are done.
  7. To set a programmable empty flag, do the following:
    - Make sure you have enabled Empty on page 3 of the wizard and set any handshaking flags you require.
    - Go to page 5 and enable Programmable Empty.

- Select one of the four mutually exclusive configurations for Programmable Empty on page 5. See [Programmable Empty, on page 575](#) in the *Reference Manual* for details.


You can now generate the FIFO and add it to the design, as described in [Specifying FIFOs with SYNCORE, on page 114](#).

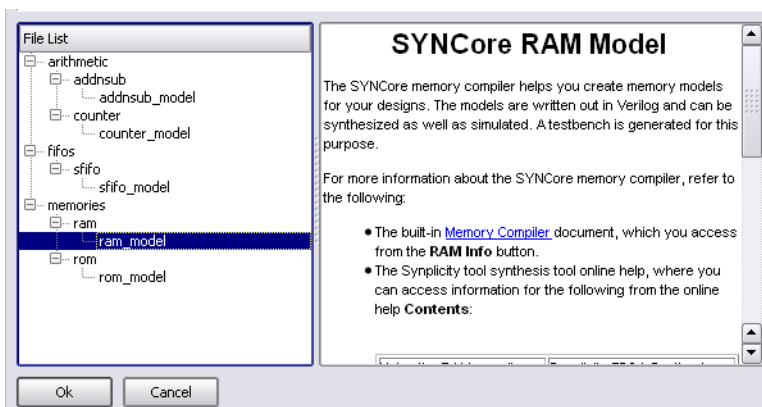
## Specifying RAMs with SYNCORE

The SYNCORE IP wizard helps you generate Verilog code for your RAM implementation requirements.

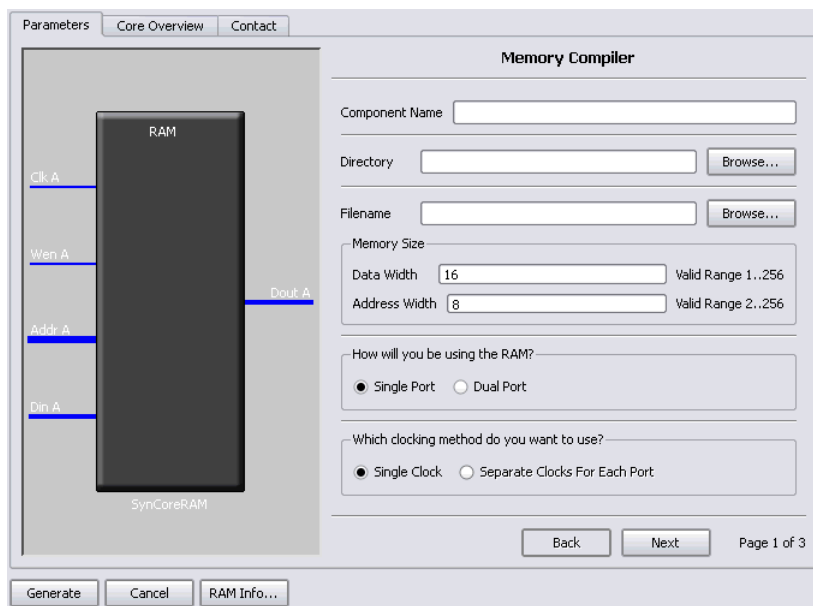
The following procedure shows you how to generate Verilog code for a RAM you specify, using the SYNCORE IP wizard.

### 1. Start the wizard.

- From the synthesis tool GUI, select Run->Launch SYNCORE or click the Launch SYNCORE icon  to start the SYNCORE IP wizard.



- In the window that opens, select `ram_model` and click Ok. This opens the first screen of the wizard.



2. Specify the parameters you need in the wizard.
  - For details about the parameters for a single-port RAM, see [Specifying Parameters for Single-Port RAM, on page 122](#).
  - For details about the parameters for a dual-port RAM, see [Specifying Parameters for Dual-Port RAM, on page 123](#). Note that dual-port implementations are only supported for some technologies.

The RAM symbol on the left reflects the parameters you set.

The default settings for the tool implement a block RAM with synchronous resets, and where all edges (clock, enable, and reset) are considered positive.

3. After you have specified all the parameters you need, click the **Generate** button in the lower left corner.

The tool displays a confirmation message is displayed (TCL execution successful!) and writes the required files to the directory you specified in the parameters. The HDL code is in Verilog.

SYNCore also generates a testbench for the RAM. The testbench covers a limited set of vectors.

You can now close the SYNCORE Memory Compiler.

4. Edit the RAM files if necessary.

- The default RAM has a `no_rw_check` attribute enabled. If you do not want this, edit `syncore_ram.v` and comment out the ``define SYN_MULTI_PORT_RAM` statement, or use ``undef SYN_MULTI_PORT_RAM`.
- If you want to use the synchronous RAMs available in the target technology, make sure to register either the read address or the outputs.

5. Add the RAM you generated to your design.

- Use the Add File command to add the Verilog design file that was generated and the `syncore_ram.v` file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
- Use a text editor to open the `instantiation_file.vin` template file, which is located in the same directory. Copy the lines that define the memory, and paste them into your top-level module. The following figure shows a template file (in red text) inserted into a top-level module.

```

module top (

input ClkA,
input [7:0] AddrA,
input [15:0] DataInA,
input WrEnA,

output [15:0] DataOutA

);

myram2 <InstanceName> (
 .PortAClk(PortAClk)
 , .PortAAddr(PortAAddr)
 , .PortADataIn(PortADataIn)
 , .PortAWriteEnable(PortAWriteEnable)
 , .PortADataOut(PortADataOut)
);

endmodule

```

**template**

- Edit the template port connections so that they agree with the port definitions in the top-level module as shown in the example below. You can also assign a unique name to each instantiation.

```
module top (

 input ClkA,
 input [7:0] AddrA,
 input [15:0] DataInA,
 input WrEnA,

 output [15:0] DataOutA

);

myram2 decoderram(
 .PortAClk(ClkA)
 , .PortAAddr(AddrA)
 , .PortADataIn(DataInA)
 , .PortAWriteEnable(WrEnA)
 , .PortADataOut(DataOutA)
);

endmodule
```

## Specifying Parameters for Single-Port RAM

To create a single-port RAM with the SYNCORE Memory Compiler, you need to specify a single read/write address (single port) and a single clock. You only need to configure Port A. The following procedure lists what you need to specify. For descriptions of each parameter, refer to [SYNCore RAM Wizard, on page 199](#) in the *Reference Manual*.

1. Start the SYNCORE RAM wizard, as described in [Specifying RAMs with SYNCORE, on page 119](#).
2. Do the following on page 1 of the RAM wizard:
  - In Component Name, specify a name for the memory. Do not use spaces.
  - In Directory, specify a directory where you want the output files to be written. Do not use spaces.

- In Filename, specify a name for the Verilog file that will be generated with the RAM specifications. Do not use spaces.
- Enter data and address widths.
- Enable Single Port, to specify that you want to generate a single-port RAM. This automatically enables Single Clock.
- Click Next. The wizard opens another page where you can set parameters for Port A.

The RAM symbol dynamically updates to reflect the parameters you set.

3. Do the following on page 2 of the RAM wizard:

- Set Use Write Enable to the setting you want.
- Set Register Read Address to the setting you want.
- Set Synchronous Reset to the setting you want. Register Outputs is always enabled
- Specify the read access you require for the RAM.

You can now generate the RAM by clicking Generate, as described in [Specifying RAMs with SYNCore, on page 119](#). You do not need to specify any parameters on page 3, as this is a single-port RAM and you do not need to specify Port B. All output files are in the directory you specified on the first page of the wizard.

For details about setting dual-port RAM parameters, see [Specifying Parameters for Dual-Port RAM, on page 123](#). For read/write timing diagrams, see [Read/Write Timing Sequences, on page 585](#) of the *Reference Manual*.

## Specifying Parameters for Dual-Port RAM

The following procedure shows you how to set parameters for dual-port memory in the SYNCore wizard. Dual-port RAMs are only supported for some technologies. For information about generating single-port RAMs, see [Specifying Parameters for Single-Port RAM, on page 122](#). It shows you how to generate these common RAM configurations:

- One read access and one write access
- Two read accesses and one write access
- Two read accesses and two write accesses

For the corresponding read/write timing diagrams, see [Read/Write Timing Sequences](#), on page 585 of the *Reference Manual*.

1. Start the SYNCore RAM wizard, as described in [Generating IP with SYNCore](#), on page 114.
2. Do the following on page 1 of the RAM wizard:
  - In Component Name, specify a name for the memory. Do not use spaces.
  - In Directory, specify a directory where you want the output files to be written. Do not use spaces.
  - In Filename, specify a name for the Verilog file that will be generated with the RAM specifications. Do not use spaces.
  - Enter data and address widths.
  - Enable Dual Port, to specify that you want to generate a dual-port RAM.
  - Specify the clocks.

|                       |                      |
|-----------------------|----------------------|
| For a single clock... | Enable Single Clock. |
|-----------------------|----------------------|

|                                              |                                       |
|----------------------------------------------|---------------------------------------|
| For separate clocks for each of the ports... | Enable Separate Clocks For Each Port. |
|----------------------------------------------|---------------------------------------|

- Click Next. The wizard opens another page where you can set parameters for Port A.
3. Do the following on page 2 of the RAM wizard to specify settings for Port A:
    - Set parameters according to the kind of memory you want to generate:

|                      |                          |
|----------------------|--------------------------|
| One read & one write | Enable Read Only Access. |
|----------------------|--------------------------|

|                       |                                                                          |
|-----------------------|--------------------------------------------------------------------------|
| Two reads & one write | Enable Read and Write Access.<br>Specify a setting for Use Write Enable. |
|-----------------------|--------------------------------------------------------------------------|

|                        |                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------|
| Two reads & two writes | Enable Read and Write Access.<br>Specify a setting for Use Write Enable.<br>Specify a read access option for Port A. |
|------------------------|----------------------------------------------------------------------------------------------------------------------|



- Specify a setting for Register Read Address.
  - Set Synchronous Reset to the setting you want. Register Outputs is always enabled.
  - Click Next. The wizard opens another page where you can set parameters for Port B. The page and the parameters are identical to the previous page, except that the settings are for Port B instead of Port A.
4. Specify the settings for Port B on page 3 of the wizard according to the kind of memory you want to generate:

|                        |                                                                                                                                                                                                                                                                        |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| One read & one write   | Enable Write Only Access.<br>Set Use Write Enable to the setting you want.                                                                                                                                                                                             |
| Two reads & one write  | Enable Read Only Access.<br>Specify a setting for Register Read Address.                                                                                                                                                                                               |
| Two reads & two writes | Enable Read and Write Access.<br>Specify a setting for Use Write Enable.<br>Specify a setting for Register Read Address.<br>Set Synchronous Reset to the setting you want.<br>Note that Register Outputs is always enabled.<br>Select a read access option for Port B. |

The RAM symbol on the left reflects the parameters you set. All output files are written to the directory you specified on the first page of the wizard.

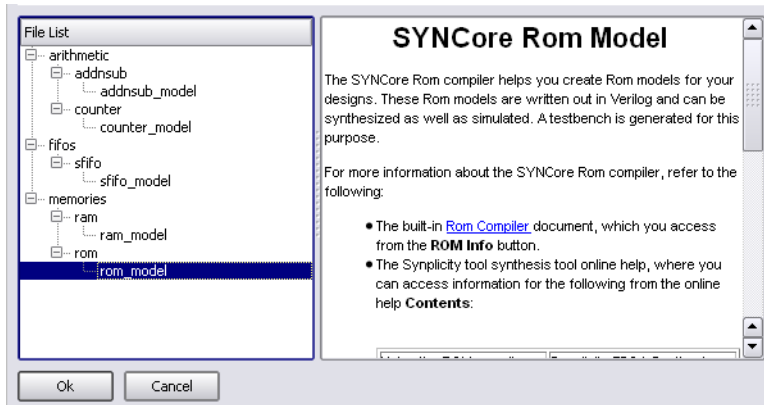
You can now generate the RAM by clicking Generate, as described in [Generating IP with SYNCORE, on page 114](#), and add it to your design.

## Specifying ROMs with SYNCORE

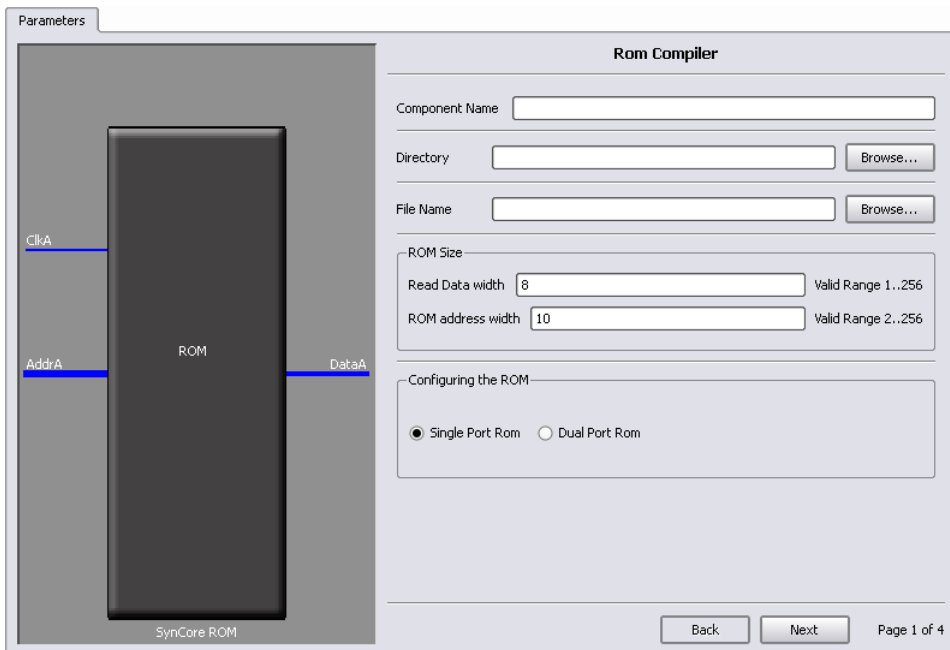
The SYNCORE IP wizard helps you generate Verilog code for your ROM implementation requirements.

The following procedure shows you how to generate Verilog code for a ROM you define using the SYNCORE IP wizard.

1. Start the wizard.
  - From the FPGA synthesis tool GUI, select Run->Launch SYNCORE or click the Launch SYNCORE icon  to start the SYNCORE IP wizard.



- In the window that opens, select rom\_model and click Ok to open page 1 of the wizard.



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying ROM Parameters, on page 129](#). The ROM symbol on the left reflects any parameters you set.

3. After you have specified all the parameters you need, click the **Generate** button in the lower left corner. The tool displays a confirmation message (TCL execution successful!) and writes the required files to the directory you specified on page 1 of the wizard. The HDL code is in Verilog.

SYNCORE also generates a testbench for the ROM. The testbench covers a limited set of vectors.

You can now close the SYNCORE ROM Compiler.

4. Edit the ROM files if necessary. If you want to use the synchronous ROMs available in the target technology, make sure to register either the read address or the outputs.
5. Add the ROM you generated to your design.
  - Use the Add File command to add the Verilog design file that was generated and the `syncore_rom.v` file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
  - Use a text editor to open the `instantiation_file.vin` template file. This file is located in the same output files directory. Copy the lines that define the ROM, and paste them into your top-level module. The following figure shows a template file (in red text) inserted into a top-level module.

```
module test_rom_style(z,a,clk,en,rst);
input clk,en,rst;
output reg [3:0] z;
input [6:0] a;
```

```
my1stROM <InstanceName> (
 // Output Ports
 .DataA(DataA),

 // Input Ports
 .ClkA(ClkA),
 .EnA(EnA),
 .ResetA(ResetA),
 .AddrA(AddrA)
);
```

template

- Edit the template port connections so that they agree with the port definitions in the top-level module as shown in the example below. You can also assign a unique name to each instantiation.

```

module test_rom_style(z,a,clk,en,rst);
 input clk,en,rst;
 output reg [3:0] z;
 input [6:0] a;

 my1stROM decode_rom(
 // Output Ports
 .DataA(z),

 // Input Ports
 .ClkA(clk),
 .EnA(en),
 .ResetA(rst),
 .AddrA(a)
);

```

## Port List

PortA interface signals are applicable for both single-port and dual-port configurations; PortB signals are applicable for dual-port configuration only.

| Name   | Type   | Description                               |
|--------|--------|-------------------------------------------|
| ClkA   | Input  | Clock input for Port A                    |
| EnA    | Input  | Enable input for Port A                   |
| AddrA  | Input  | Read address for Port A                   |
| ResetA | Input  | Reset or interface disable pin for Port A |
| DataA  | Output | Read data output for Port A               |
| ClkB   | Input  | Clock input for Port B                    |
| EnB    | Input  | Enable input for Port B                   |

|        |        |                                           |
|--------|--------|-------------------------------------------|
| AddrB  | Input  | Read address for Port B                   |
| ResetB | Input  | Reset or interface disable pin for Port B |
| DataB  | Output | Read data output for Port B               |

## Specifying ROM Parameters

If you are creating a single-port ROM with the SYNCore IP wizard, you need to specify a single read address and a single clock, and you only need to configure the Port A parameters on page 2 . If you are creating a dual-port ROM, you must additionally configure the Port B parameters on page 3. The following procedure lists what you need to specify. For descriptions of each parameter, refer to [SYNCore RAM Wizard, on page 199](#) in the *Reference Manual*.

1. Start the SYNCore ROM wizard, as described in [Specifying ROMs with SYNCore, on page 125](#).
2. Do the following on page 1 of the ROM wizard:
  - In Component Name, specify a name for the memory. Do not use spaces.
  - In Directory, specify a directory where you want the output files to be written. Do not use spaces.
  - In Filename, specify a name for the Verilog file that will be generated with the ROM specifications. Do not use spaces.
  - Enter values for Read Data width and ROM address width (minimum depth value is 2; maximum depth of the memory is limited to  $2^{256}$ ).
  - Select Single Port Rom to indicate that you want to generate a single-port ROM or select Dual Port Rom to generate a dual-port ROM.
  - Click Next. The wizard opens page 2 where you set parameters for Port A.

The ROM symbol dynamically updates to reflect any parameters you set.

3. Do the following on page 2 (Configuring Port A) of the RAM wizard:
  - For synchronous ROMs, select Register address bus AddrA and/or Register output data bus DataA to register the read address and/or the outputs. Selecting either checkbox enables the Enable for Port A checkbox which is used to select the Enable level.

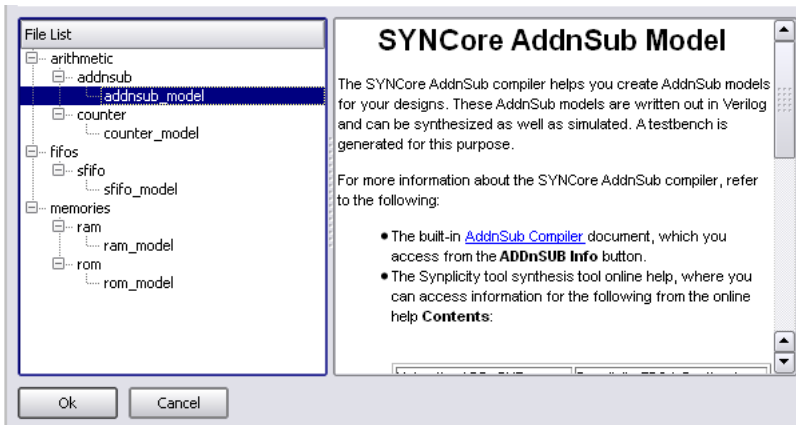
- Set the Configure Reset Options. Enabling the checkbox enables the type of reset (asynchronous or synchronous) and allows an output data pattern (all 1's or a specified pattern) to be defined on page 4.
4. If you are generating a dual-port ROM, set the port B parameters on page 3 (the page 3 parameters are only enabled when Dual Port Rom is selected on page 1).
  5. On page 4, specify the location of the ROM initialization file and the data format (Hexadecimal or Binary). ROM initialization is supported using memory-coefficient files. The data format is either binary or hexadecimal with each data entry on a new line in the memory-coefficient file (specified by parameter INIT\_FILE). Supported file types are .txt, .mem, .dat, and .init (recommended).
  6. Generate the ROM by clicking Generate, as described in [Specifying ROMs with SYNCORE, on page 125](#) and add it to your design. All output files are in the directory you specified on page 1 of the wizard.

For read/write timing diagrams, see [Read/Write Timing Sequences, on page 585](#) of the *Reference Manual*.

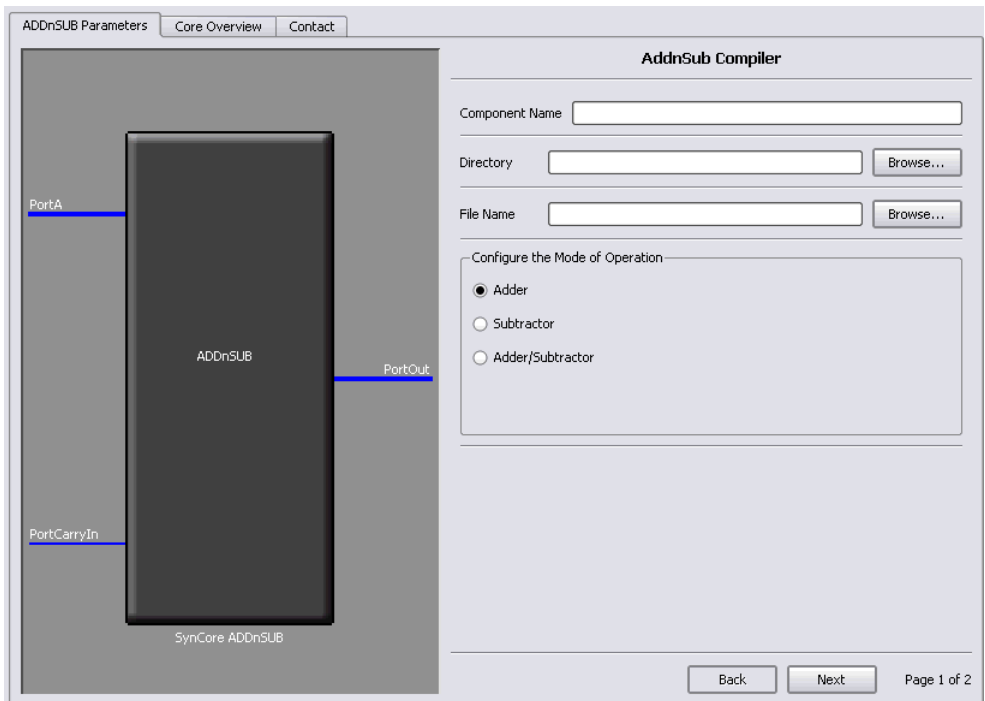
## Specifying Adder/Subtractors with SYNCORE

The SYNCORE IP wizard helps you generate Verilog code for your adder/subtractor implementation requirements. The following procedure shows you how to generate Verilog code for an adder/subtractor that you define using the SYNCORE IP wizard.

1. Start the wizard.
  - From the FPGA synthesis tool GUI, select Run->Launch SYNCORE or click the Launch SYNCORE icon  to start the SYNCORE IP wizard.



- In the window that opens, select addnsub\_model and click Ok to open page 1 of the wizard.



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying Adder/Subtractor Parameters, on page 135](#). The ADDnSUB symbol on the left reflects any parameters you set.
3. After you have specified all the parameters you need, click the Generate button in the lower left corner.

The tool displays a confirmation message (TCL execution successful!) and writes the required files to the directory you specified on page 1 of the wizard. The HDL code is in Verilog.

The SYNCORE wizard also generates a testbench for your adder/subtractor. The testbench covers a limited set of vectors. You can now close the wizard.

4. Add the adder/subtractor you generated to your design.
  - Edit the adder/subtractor files if necessary.
  - Use the Add File command to add the Verilog design file that was generated and the `syncore_addnsub.v` file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
  - Use a text editor to open the `instantiation_file.v` template file. This file is located in the same output files directory. Copy the lines that define the adder/subtractor and paste them into your top-level module. The following figure shows a template file (in red text) inserted into a top-level module.



```

.....
module top (
 output [15 : 0] Out,
 input Clk,
 input [15 : 0] A,
 input CEA,
 input RSTA,
 input [15 : 0] B,
 input CEB,
 input RSTB,
 input CEOut,
 input RSTOut,
 input ADDnSUB,
 input CarryIn);

My_ADDnSUB <InstanceName> (
// Output Ports
 .PortOut(PortOut),
// Input Ports
 .PortClk(PortClk),
 .PortA(PortA),
 .PortCEA(PortCEA),
 .PortRSTA(PortRSTA),
 .PortB(PortB),
 .PortCEB(PortCEB),
 .PortRSTB(PortRSTB),
 .PortCEOut(PortCEOut),
 .PortRSTOut(PortRSTOut),
 .PortADDnSUB(PortADDnSUB),
 .PortCarryIn(PortCarryIn));
endmodule

```

template

- Edit the template port connections so that they agree with the port definitions in the top-level module as shown in the example below. You can also assign a unique name to each instantiation.

```

module top (
 output [15 : 0] Out,
 input Clk,
 input [15 : 0] A,
 input CEA,
 input RSTA,
 input [15 : 0] B,
 input CEB,

```

```

 input RSTB,
 input CEOut,
 input RSTOut,
 input ADDnSUB,
 input CarryIn);

My_ADDnSUB ADDnSUB_inst (
// Output Ports
 .PortOut(Out),
// Input Ports
 .PortClk(Clk),
 .PortA(A),
 .PortCEA(CEA),
 .PortRSTA(RSTA),
 .PortB(B),
 .PortCEB(CEB),
 .PortRSTB(RSTB),
 .PortCEOut(CEOut),
 .PortRSTOut(RSTOut),
 .PortADDnSUB(ADDnSUB),
 .PortCarryIn(CarryIn));
endmodule

```

## Port List

The following table lists the port assignments for all possible configurations; the third column specifies the conditions under which the port is available.

| Port Name | Description                                                                | Required/Optional                                                            |
|-----------|----------------------------------------------------------------------------|------------------------------------------------------------------------------|
| PortA     | Data input for adder/subtractor<br>Parameterized width and pipeline stages | Always present                                                               |
| PortB     | Data input for adder/subtractor<br>Parameterized width and pipeline stages | Not present if adder/subtractor is configured as a constant adder/subtractor |
| PortClk   | Primary clock input; clocks all registers in the unit                      | Always present                                                               |
| PortRstA  | Reset input for port A pipeline registers (active high)                    | Not present if pipeline stage for port A is 0                                |

| Port Name   | Description                                              | Required/Optional                                                              |
|-------------|----------------------------------------------------------|--------------------------------------------------------------------------------|
| PortRstB    | Reset input for port B pipeline registers (active high)  | Not present if pipeline stage for port B is 0 or for constant adder/subtractor |
| PortADDnSUB | Selection port for dynamic operation                     | Not present if adder/subtractor configured as standalone adder or subtractor   |
| PortRstOut  | Reset input for output register (active high)            | Not present if output pipeline stage is 0                                      |
| PortCEA     | Clock enable for port A pipeline registers (active high) | Not present if pipeline stage for port A is 0                                  |
| PortCEB     | Clock enable for port B pipeline registers (active high) | Not present if pipeline stage for port B is 0 or for constant adder/subtractor |
| PortCarryIn | Carry input for adder/subtractor                         | Always present                                                                 |
| PortCEO     | Clock enable for output register (active high)           | Not present if output pipeline stage is 0                                      |
| PortOut     | Data output                                              | Always present                                                                 |

## Specifying Adder/Subtractor Parameters

The SYNCORE adder/subtractor can be configured as any of the following:

- Adder
- Subtractor
- Dynamic Adder/Subtractor

If you are creating a constant input adder, subtractor, or a dynamic adder/subtractor with the SYNCORE IP wizard, you must select Constant Value Input and specify a value for port B in the Constant Value/Port B Width field on page 2 of the parameters. The following procedure lists the parameters you need to define when generating an adder/subtractor. For descriptions of each parameter, see [SYNCORE Adder/Subtractor Wizard, on page 206](#) in the *Reference Manual*.

1. Start the SYNCORE adder/subtractor wizard as described in [Specifying Adder/Subtractors with SYNCORE, on page 130](#).

2. Enter the following on page 1 of the wizard:
  - In the Component Name field, specify a name for your adder/subtractor. Do not use spaces.
  - In the Directory field, specify a directory where you want the output files to be written. Do not use spaces.
  - In the Filename field, specify a name for the Verilog file that will be generated with the adder/subtractor definitions. Do not use spaces.
  - Select the appropriate configuration in Configure the Mode of Operation.
3. Click Next. The wizard opens page 2 where you set parameters for port A and port B.
4. Configure Port A and B.
  - In the Configure Port A section, enter a value in the Port A Width field.
  - If you are defining a synchronous adder/subtractor, check Register Input A and then check Clock Enable for Register A and/or Reset for Register A.
  - To configure port B as a constant port, go to the Configure Port B section and check Constant Value Input. Enter the constant value in the Constant Value/Port B Width field.
  - To configure port B as a dynamic port, go to the Configure Port B section and check Enable Port B and enter the port width in the Constant Value/Port B Width field.
  - To define a synchronous adder/subtractor, check Register Input B and then check Clock Enable for Register B and/or Reset for Register B.
5. In the Configure Output Port section:
  - Enter a value in the Output port Width field.
  - If you are registering the output port, check Register output Port.
  - If you are defining a synchronous adder/subtractor check Clock Enable for Register PortOut and/or Reset for Register PortOut.
6. In the Configure Reset type for all Reset Signal section, click Synchronous Reset or Asynchronous Reset as appropriate.

As you enter the page 2 parameters, the ADDnSUB symbol dynamically updates to reflect the parameters you set.

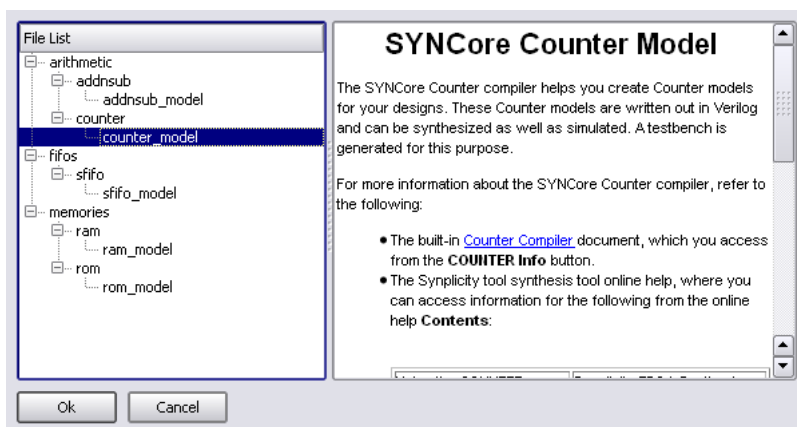
7. Generate the adder/subtractor by clicking the Generate button as described in [Specifying Adder/Subtractors with SYNCORE, on page 130](#) and add it to your design. All output files are in the directory you specified on page 1 of the wizard.

## Specifying Counters with SYNCORE

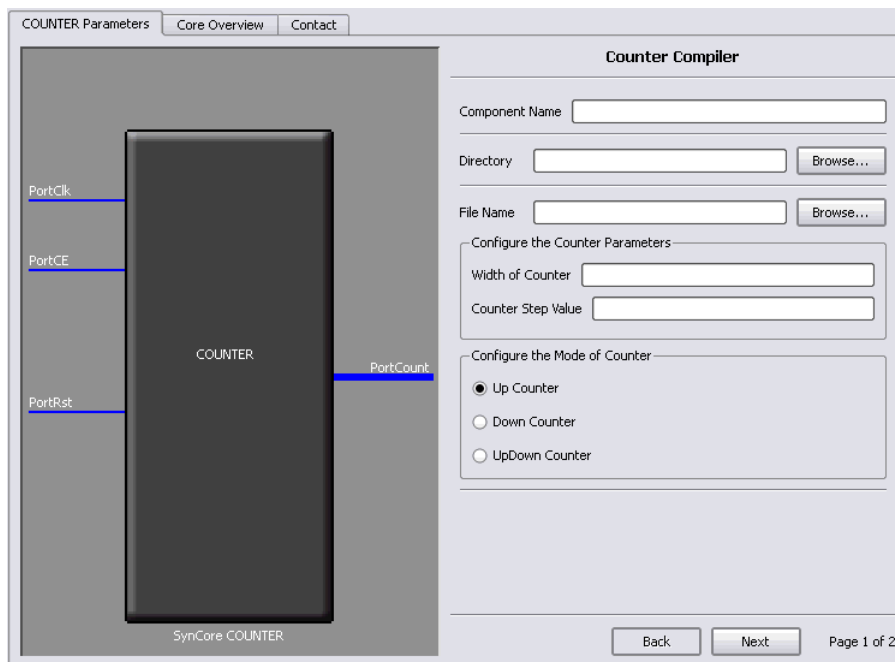
The SYNCORE IP wizard helps you generate Verilog code for your counter implementation requirements.

The following procedure shows you how to generate Verilog code for a counter that you define using the SYNCORE IP wizard.

1. Start the wizard.
  - From the FPGA synthesis tool GUI, select Run->Launch SYNCORE or click the Launch SYNCORE icon  to start the SYNCORE IP wizard.



- In the window that opens, select counter\_model and click Ok to open page 1 of the wizard.



2. Specify the parameters you need in the wizard. For details about the parameters, see [Specifying Counter Parameters, on page 141](#). The COUNTER symbol on the left reflects any parameters you set.
3. After you have specified all the parameters you need, click the Generate button in the lower left corner.

The tool displays a confirmation message (TCL execution successful!) and writes the required files to the directory you specified on page 1 of the wizard. The HDL code is in Verilog.

The SYNCore wizard also generates a testbench for your counter. The testbench covers a limited set of vectors. You can now close the wizard.

4. Add the counter you generated to your design.
  - Edit the counter files if necessary.
  - Use the Add File command to add the Verilog design file that was generated and the `syncore_addnsb.v` file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.

- Use a text editor to open the instantiation\_file.v template file. This file is located in the same output files directory. Copy the lines that define the counter and paste them into your top-level module. The following figure shows a template file (in red text) inserted into a top-level module.

```

module counter #(
 parameter COUNT_WIDTH = 5,
 parameter STEP = 2,
 parameter RESET_TYPE = 0,
 parameter LOAD = 2,
 parameter MODE = "Dynamic")
(
// Output Ports
 output wire [WIDTH-1:0] Count,
// Input Ports
 input wire Clock,
 input wire Reset,
 input wire Up_Down,
 input wire Load,
 input wire [WIDTH-1:0] LoadValue,
 input wire Enable);

SynCoreCounter #(
 .COUNT_WIDTH(COUNT_WIDTH),
 .STEP(STEP),
 .RESET_TYPE(RESET_TYPE),
 .LOAD(LOAD),
 .MODE(MODE))

SynCoreCounter_inst1 (
 .PortCount(PortCount),
 .PortClk(PortClk),
 .PortRST(PortRST),
 .PortUp_nDown(PortUp_nDown),
 .PortLoad(PortLoad),
 .PortLoadValue(PortLoadValue),
 .PortCE(PortCE));

endmodule

```

template

Edit the template port connections so that they agree with the port definitions in the top-level module as shown in the example below. You can also assign a unique name to each instantiation.

```

module counter #(
 parameter COUNT_WIDTH = 5,
 parameter STEP = 2,
 parameter RESET_TYPE = 0,
 parameter LOAD = 2,
 parameter MODE = "Dynamic")

(
// Output Ports
 output wire [WIDTH-1:0] Count,
// Input Ports
 input wire Clock,
 input wire Reset,
 input wire Up_Down,
 input wire Load,
 input wire [WIDTH-1:0] LoadValue,
 input wire Enable);

SynCoreCounter #(
 .COUNT_WIDTH(COUNT_WIDTH),
 .STEP(STEP),
 .RESET_TYPE(RESET_TYPE),
 .LOAD(LOAD),
 .MODE(MODE))

SynCoreCounter_inst1 (
 .PortCount(PortCount),
 .PortClk(Clock),
 .PortRST(Reset),
 .PortUp_nDown(Up_Down),
 .PortLoad(Load),
 .PortLoadValue(LoadValue),
 .PortCE(Enable));

endmodule

```

## Port List

The following table lists the port assignments for all possible configurations; the third column specifies the conditions under which the port is available.



| Port Name     | Description                                                                            | Required/Optional                           |
|---------------|----------------------------------------------------------------------------------------|---------------------------------------------|
| PortCE        | Count Enable input pin with size one (active high)                                     | Always present                              |
| PortClk       | Primary clock input                                                                    | Always present                              |
| PortLoad      | Load Enable input which loads the counter (active high).                               | Not present for parameter LOAD=0            |
| PortLoadValue | Load value primary input (active high)                                                 | Not present for parameter LOAD=0 and LOAD=1 |
| PortRST       | Reset input which resets the counter (active high)                                     | Always present                              |
| PortUp_nDown  | Primary input which determines the counter mode.<br>0 = Up counter<br>1 = Down counter | Present only for MODE="Dynamic"             |
| PortCount     | Counter primary output                                                                 | Always present                              |

## Specifying Counter Parameters

The SYNCORE counter can be configured for any of the following functions:

- Up Counter
- Down Counter
- Dynamic Up/Down Counter

The counter core can have a constant or variable input load or no load value. If you are creating a constant-load counter, you will need to select Enable Load and Load Constant Value on page 2 of the wizard. If you are creating a variable-load counter, you will need to select Enable Load and Use Variable Port Load on page 2. The following procedure lists the parameters you need to define when generating a counter. For descriptions of each parameter, see [SYNCore Counter Wizard, on page 210](#) of the *Reference Manual*.

1. Start the SYNCORE counter wizard, as described in [Specifying Counters with SYNCORE, on page 137](#).
2. Enter the following on page 1 of the wizard:

- In the Component Name field, specify a name for your counter. Do not use spaces.
  - In the Directory field, specify a directory where you want the output files to be written. Do not use spaces.
  - In the Filename field, specify a name for the Verilog file that will be generated with the counter definitions. Do not use spaces.
  - Enter the width and depth of the counter in the Configure the Counter Parameters section.
  - Select the appropriate configuration in the Configure the Mode of Counter section.
3. Click Next. The wizard opens page 2 where you set parameters for PortLoad and PortLoadValue.
- Select Enable Load option and the required load option in Configure Load Value section.
  - Select the required reset type in the Configure Reset type section.

The COUNTER symbol dynamically updates to reflect the parameters you set.

4. Generate the counter core by clicking Generate button. All output files are written to the directory you specified on page 1 of the wizard.

## The ReadyIP Encryption Flow

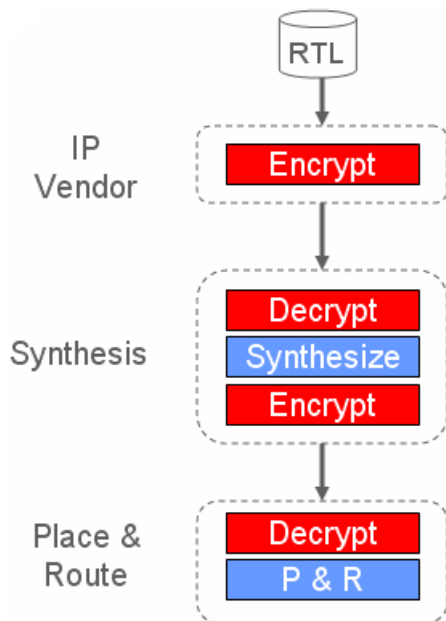
The ReadyIP™ encryption flow is a design flow that encourages interoperability while protecting IP implementations using encryption/decryption technology licensed from RSA Securities. This flow offers the following advantages: interoperability, protection of IP, reuse of IP, and a standard flow for IP encryption. Synopsys has donated this scheme to VSIA (OpenIP) and it is in the process of being made into a standard.

See the following:

- [Overview of the Synopsys ReadyIP Flow](#), on page 143
- [Encryption and Decryption](#), on page 144

### Overview of the Synopsys ReadyIP Flow

The complete flow for protecting IP requires a partnership between the IP vendor, Synopsys, and the silicon vendor, as illustrated in the following figure. However, depending on the level of agreement between Synopsys and the silicon vendor downstream, the re-encryption of IP after synthesis can vary from the ideal flow shown in the figure.

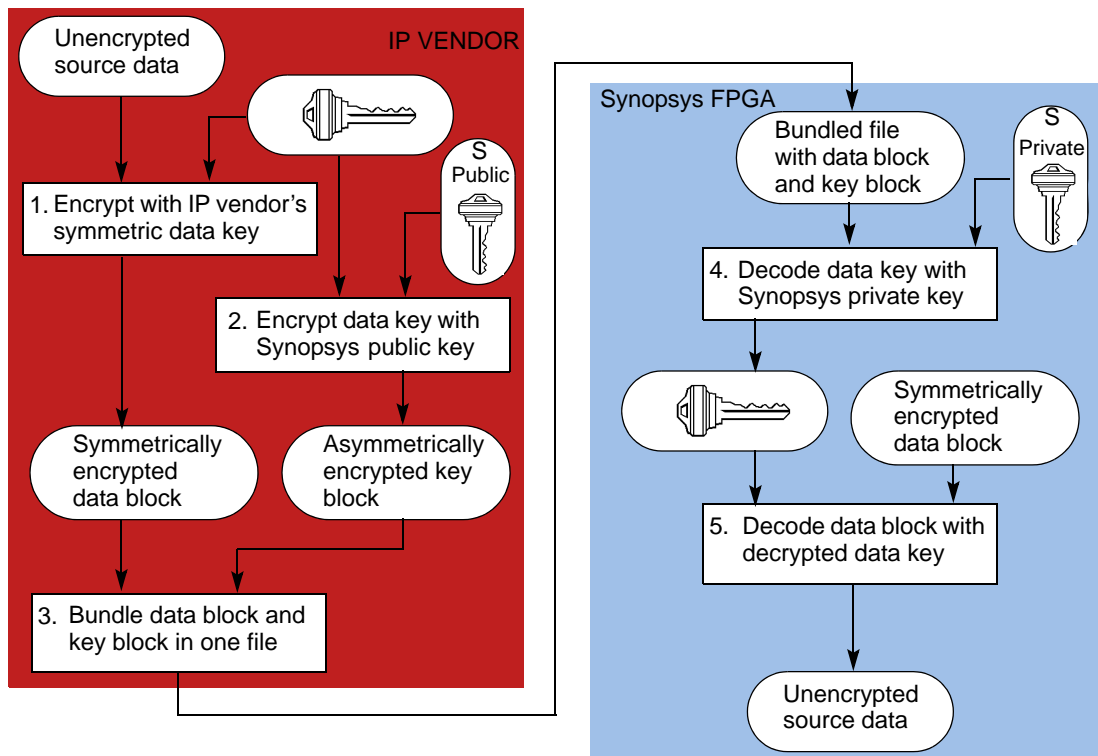


For further details of the hand-offs between vendors and how encryption and decryption are handled, see [Encryption and Decryption, on page 144](#).

## Encryption and Decryption

There are two major classes of encryption/decryption algorithms: symmetric, and asymmetric (see [Encryption and Decryption Methodologies, on page 613](#) in the *Reference Manual* for details). Each has its own advantages and disadvantages. The Synopsys approach in the ReadyIP flow is a hybrid scheme that uses both asymmetric and symmetric encryption to leverage the strengths of each scheme. The methodology described here can also be used for other design handoffs. For example, for a handoff from synthesis to place-and-route, the synthesis tool would be in the upstream position occupied by the IP vendor in this flow, and the FPGA vendor would be in the downstream position occupied by the synthesis tool.

The following figure illustrates the steps in this encryption/decryption methodology, showing the handoff from an IP vendor to a Synopsys FPGA synthesis tool.

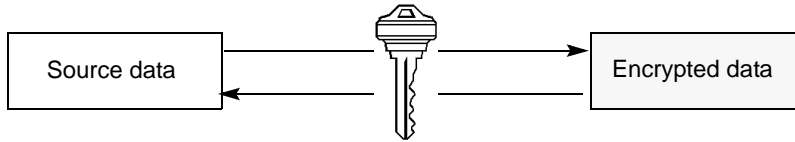


The following describes each of the phases shown in the figure. Note that Synopsys provides the `encryptIP` script ([The encryptIP Script, on page 611](#) in the *Reference Manual*) to simplify and automate the process of encrypting data for the IP vendor.

### Data Encryption - Step 1

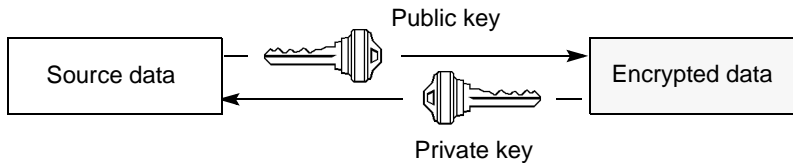
The IP vendor encrypts the IP data using their own symmetric key. This key is called the *data key*. The result of encoding is a *data block*. Using symmetric encryption offers two advantages to the IP vendor: fast data encryption because it is symmetric encryption, and freedom to use any symmetric scheme they choose.

## Symmetric Encryption/Decryption with One Key

**Data Key Encryption - Step 2**

Next, the IP vendor encrypts the data key used to encode the IP block, and generates a *key block*. For this operation, the vendor uses RSA asymmetric encryption and the public key provided by Synopsys.

## Asymmetric Encryption/Decryption with Public and Private Keys



Asymmetric encryption offers the following advantages:

- Although asymmetric encryption is compute-intensive, the data key itself is small, so this will not be time-intensive.
- The IP vendor can use public keys from different vendors to encrypt the same block for different EDA vendors. This ensures that IP consistency is maintained, because there is no need for multiple copies.
- Only the public key from the downstream vendor needs to be passed to the IP vendor.

**Bundling of Encrypted Data Block and Data Key - Step 3**

The IP vendor bundles the encrypted data block with the key block into one file for handoff to the EDA vendor. Note that this methodology allows the IP vendor to create just one version of the IP which includes the key blocks for all the downstream vendors it supports; for example, a synthesis tool and a simulation tool. Also, this approach eliminates the need to securely transmit the symmetric key, because this is included in the file. Security is maintained because the key and the data are encrypted.

In the figure, this is the point at which the IP vendor hands off the IP to the synthesis tool.

### **Data Key Decryption - Step 4**

Decryption is a two-stage process. The first step is to decrypt the symmetric data key from the IP vendor, which was encrypted using the asymmetric public key provided. To decode this, you use the private key counterpart to the public key and extract the data key.

### **Data Decryption - Step 5**

The second step is to use the extracted data key to access the IP data. As the data key is the original symmetric key used to encode the IP, the process is quick. The synthesis tools can now synthesize the unencrypted IP.

After synthesis, the IP can be re-encrypted if the vendor has adopted the Synopsys methodology. See [Output Methods for encryptIP, on page 614](#) in the *Reference Manual* for a description of the choices available.

### **Re-Encryption in the ReadyIP Flow**

Re-encryption of the synthesized IP for FPGA vendors downstream requires that the FPGA vendor supply Synopsys with a public key. If such an agreement is not in place, the IP is treated as a black box. Accordingly, you can either have an IP flow that outputs regular netlists with black boxes, or one that outputs encrypted netlists.

## Working with Encrypted IP

The encryption approach used by Synplify Pro and Premier follows the OpenIP scheme that Synopsys donated to VSIA and which has now been submitted to the IEEE (*The ReadyIP Encryption Flow, on page 143*). With this approach, the IP vendor can encrypt and control distribution of the IP from their own website. The synthesis user will have access from the synthesis tool to IP that the vendor makes available for download and evaluation within a synthesis design.

The following describe how to encrypt and package your IP for evaluation if you are an IP vendor, and how to access and evaluate available IP, if you are an end-user.

- [Encrypting Your IP](#), on page 148
- [Preparing the IP Package](#), on page 153
- [Evaluating Vendor IP](#), on page 158

### Encrypting Your IP

IP vendors can use the ReadyIP scheme to provide IP for synthesis users to evaluate and use. You can encrypt your IP. The ReadyIP scheme uses a two-stage encryption process:

- First, you encrypt your IP files using a symmetric encryption algorithm and your own session or data key. This creates an encrypted data block. Initially, this session key may be dictated by the FPGA vendor. In the future and in the ReadyIP standard, any key may be used.
- Next, you encrypt the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. Synplify currently supports RSA encryption.

Synopsys provides a script that simplifies this process. See the following procedure for details about using it.

### Preparing and Encrypting Your IP

To prepare and encrypt your IP, do the following:

1. Gather your RTL files.



You only encrypt the RTL. You can encrypt any number of Verilog and VHDL (or mixed) RTL files to form your encrypted IP, and each file may be encrypted in whole or in part.

2. Determine your file setup for each IP.
  - Create a single set of files for the IP (for use with all supported FPGAs), if your IP has no vendor-specific or vendor-optimized content, and the output method is supported by all intended consumers (blackbox or plaintext).
  - Create multiple versions of your protected IP if you have specific FPGA vendors or specific FPGA vendor families; if you are using FPGA device family specific RTL like architecture-specific instantiations; or if you optimized your RTL or constraints for use with a specific FPGA vendor device family or FPGA vendor.
3. Encrypt the files with the encryptIP script, as described in [Encrypting IP with the encryptIP Script, on page 149](#).
4. Package your IP, as described in [Preparing the IP Package, on page 153](#).
5. Verify that your IP works with the synthesis tools by going through the procedure the user would use.
  - For system-level IP, run it through System Designer™ and ensure bus-model compatibility between your IP and any other IP to which it interfaces. See the System Designer documentation for details on using this tool.
  - Start the synthesis tool and load the IP with the Import IP->Import IP Package command. You may load your IP into an existing Synplify project.
  - Run synthesis.

## Encrypting IP with the encryptIP Script

The following procedure shows you how to encrypt your data with the encryptIP script. The encryptIP script automates the two-stage encryption process proposed in the ReadyIP scheme ([The ReadyIP Encryption Flow, on page 143](#)).

- First, it encrypts your IP files using a symmetric encryption algorithm and your own session or data key. This creates an encrypted data block.

- Next, it encrypts the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. Synplify currently supports RSA encryption.

1. Install the encryptIP Perl script.

- Contact Synopsys and obtain the encryptIP Perl script.
- Install perl on your machine. You cannot run the script if you do not have perl installed.

2. Make sure you have the right key length for the encryption algorithm you are using.

For example, TEST1234 becomes a 64-bit key, so it matches the des-cbc algorithm.

3. Run the encryptIP script on each RTL file you want to encrypt.

The following example encrypts the Verilog `plain_ip.v` file into an encrypted file called `protected_ip.v`, using AES128-cbc encryption. The session key is MY\_AES\_SAMPLEKEY. See [Syntax for Running encryptIP, on page 611](#) in the *Reference Manual* for details about the syntax and required parameters.

```
perl encryptIP -in plain_ip.v -out protected_ip.v -c aes128-cbc
-k MY_AES_SAMPLEKEY -bd 16OCT2007 -om plaintext -v
```

4. Make sure you specify the appropriate output method (`-om`) when you run the script.

This is important because the output method (`-om`) determines what is encrypted to the user. When the example above is synthesized, the user can view the output netlist because the output method specified is `plaintext`, which means that the synthesis output netlist includes the IP netlist in an unencrypted and readable form. See [Specifying the Output Method for encryptIP, on page 151](#) for more information.

The script encrypts the IP with the standard symmetric encryption algorithm you specified, and produces a `data_block`. The data key used for encrypting the HDL is then encrypted with an asymmetric algorithm and the Synopsys public key, and produces a `key_block`. The `data_block` and the `key_block` are put together with the appropriate pragmas for the flow being used, and the script creates an encrypted HDL file. For a detailed figure, see [Encryption and Decryption, on page 144](#).

All other output files from synthesis (like `.srm`, `.srd`, and `.srs` files) are encrypted using the same encryption method specified for the input to synthesis. Output constraints are not encrypted.

5. Check the encrypted RTL file to make sure that there is only one key block present.

## Specifying the Output Method for encryptIP

You can control access to the IP by setting the appropriate output method. You specify the output method using the `-om` parameter, as described in [Syntax for Running encryptIP, on page 611](#) in the *Reference Manual*.

The output method mainly affects the output netlist. The following are guidelines for setting the output method for the `encryptIP` script, and detail the effects of different settings:

1. Set `-om` to `persistent_key` in the following cases:
  - If you are working with a Lattice non-CPLD technology
  - If you have an agreement in place with Synopsys and want the output netlist to be encrypted
2. Set `-om` to `plaintext` in the following cases:
  - If you want to allow the IP to be incorporated in a physical synthesis or logic synthesis design

For physical synthesis, the Synplify Premier tool runs global placement and logic synthesis simultaneously. To place the IP and its contents, the tool must be allowed to access and optimize it. Setting the output method to `plaintext` allows the tool to synthesize, run gate-level simulations, place and route, and implement an FPGA (that includes the IP) on a board.

- If you want the IP to be freely optimized by the synthesis tools

Although IP cores are already optimized, the synthesis tools can effect additional optimizations based on the design context in which it will be used. When the synthesis tool is allowed to optimize the IP, it can prune away IP logic that is unused or unnecessary in the current design context. Or take the case where the output of an instantiated IP core is timing-critical because it drives hundreds of user loads. If the synthesis tool can freely optimize, it can replicate sources within the core and fix the problem.

- To let the IP be incorporated in a logic synthesis design, set `-om` to `plaintext` or `blackbox`.

Setting the output method to `plaintext` allows the tool to synthesize, run gate-level simulations, place and route, and implement an FPGA (that includes the IP) on a board. Setting the output method to `blackbox` does not allow the tool to run gate-level simulations or place and route the IP, because it only uses the port and connectivity information.

- If you have set `-om` to `plaintext` and you want to specify individual cores as white boxes, set the `syn_macro` directive to 1 on the view for the IP.

Note that you must set this on the view, not the instance. When this is set, the tool treats the IP as a white box and only uses the timing and connection information from the IP. The synthesis tool maintains the IP boundary and only trims unused logic inside the IP.

- During synthesis, the IP contents appear as a black box in RTL, Technology, and Physical Analyst views, regardless of the output method selected.

Even if you specify `-om plaintext`, you cannot push down into an IP in an RTL, Technology, or Physical Analyst view.

- After synthesis, the output method affects the results in the following ways:
  - Output constraints for an IP are in the standard Synopsys format and are not encrypted.
  - The output method affects the contents of the output netlist and its format. This table summarizes the `encryptIP` behavior with different output methods.

| Method (-om) | Output Netlist After Synthesis                                                                                                                                                                                                                                                                                                                        |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blackbox     | The output netlist contains the IP interface only and no IP contents. It only includes IP ports and connections. The IPs are treated as black boxes, and there are no nets or instances shown inside the IP. This applies to all the netlist formats generated for different vendors, whether it is HDL (.vnm or .vhm), EDIF (.edf or .edn), or .vqm. |
| plaintext    | The output netlist contains your unencrypted IP, which is completely readable. Nothing is encrypted.                                                                                                                                                                                                                                                  |

---

| Method (-om)   | Output Netlist After Synthesis                                                                                                                                                                                                                            |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| persistent_key | The output netlist includes encrypted versions of the IP. For Lattice designs, the tool generates one output netlist (EDIF or HDL) that includes the encrypted IP blocks. The netlist is readable, except for the IP block sections, which are encrypted. |

---

## Preparing the IP Package

Do the following to package your IP and make it accessible from the synthesis tools:

1. Collect the files for the package.
  - Encrypt the files you need, as described in [Encrypting Your IP, on page 148](#).
  - Make sure your package includes the files listed in [IP Package File List, on page 155](#).
  - Structure the files as described in [Suggested Directory Structure, on page 155](#).
2. For IP-XACT models for use with System Designer (see [Providing System-Level Models for the IP, on page 157](#)), make sure of the following:
  - At a minimum, include a top-level .xml file that specifies the complete component, vendor, library, name, and component version (VLNV). This file references library components that are described in other files in the directory tree using relative paths.
  - If you want to allow System Designer to generate HDL files for the IP for later synthesis, include the Verilog/VHDL files for the IP in the package. This allows the core to be evaluated using a synthesis flow.
  - If you do not want to allow System Designer to generate HDL files for synthesis, do not include the Verilog/VHDL files in the package. The System Designer tool creates a top-level netlist and corresponding wrappers and generates an error message for the missing files. This method allows the core to be tested for compatibility with the rest of the system, but it will not be an evaluation with complete synthesis.

The IP-XACT models consist of a library of your system-level components, including bus definitions. When System Designer reads this library, the various components like specific timers, buses, and CPUs,

appear in the library window for the user to drag and drop and instantiate, as they assemble a design from the components. See the System Designer documentation for details.

3. If your IP package is intended for synthesis only, without subsystem assembly, create a compressed package for download, using one of these methods:

- Create a compressed tarball (.tar.gz), which is a tar archive compressed with the gzip utility, using one of these commands:

```
tar cf -file-list | gzip -c > compressed-tarball
gtar -cf compressed-tarball file-list
```

Preserve the directory structure when you run gzip.

- Create a zip file (.zip) by running WinZip. WinZip archives and preserves your directory hierarchy.
4. If your IP package is intended to be a subsystem that will be assembled by System Designer, create a compressed tarball (.tar.gz) using one of these commands:

```
tar cf -file-list | gzip -c > compressed-tarball
gtar -cf compressed-tarball file-list
```

Preserve the directory structure when you run gzip.

5. Post the packaged IP on your website for downloading.

The user generally untars or unzips the IP package into a top-level directory after downloading it. The synthesis tools can then read the contents of the directory.

6. Supply Synopsys with the following:

- The URL for the download package.
- Vendor and advertising information you wish to display on the Synopsys website. See [Supplying Vendor Information, on page 157](#) for details.

## IP Package File List

Your IP package should contain the following files:

| Files                           | Description                                                                                                                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ipinfo. txt                     | Text file that lists the name of the IP, the version, restrictions for use, support contact information, and an email alias to request a licence for the full RTL for your IP.                                     |
| Documentation, preferably a PDF | Documents the IP, and includes detailed information about usage restrictions like vendor, device family, etc.                                                                                                      |
| Readme                          | An optional text file that contains instructions on use of the IP for assembly and/or synthesis, and hints on how to use it correctly.                                                                             |
| Encrypted HDL or EDIF           | Protected RTL for the IP, created using the Synopsys encryptIP script. See the documentation for details.                                                                                                          |
| SDC constraints                 | Unencrypted design constraints for the IP. You need only maintain a single file for both the Synopsys synthesis tools, as the Synplify Pro software ignores any constraints that are specific to Synplify Premier. |
| SPIRIT IP-XACT v1.2 models      | System-level models for your IP. This allows the synthesis tools to include your IP in a system-level design by stitching the IP together using bus architectures.                                                 |

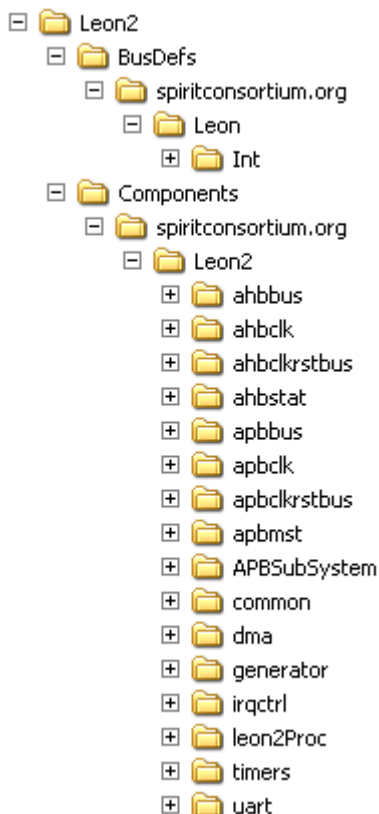
## Suggested Directory Structure

Follow these recommendations when you structure the IP package:

- If you include a Synopsys .prj project file or an xml file for use in System Designer, make sure to use relative paths from the .prj or .xml directory to refer to other files like the Verilog or VHDL files.
- Always use relative paths to reference a file.
- Always preserve directory structure when you run gzip.
- You can place IP-XACT xml files in the top-level directory or in a common subdirectory. You can have multiple files or a single file for the same component or variants of a component. However, it is preferred that you keep all IP-XACT components that are in one library at the same directory level, even if it is many levels deep in the directory hierarchy.
- For packaging, System Designer treats IP-XACT bus definitions just like IP core components. So, place each bus definition in its own separate

sub-hierarchy, parallel to the sub-hierarchies of your other system-level components. This makes it easy for the user to see if the component library includes the necessary bus definitions, and to load just the bus definition files into System Designer.

The following example shows the structure of a Leon2 processor, which is included with the System Designer installation. Note that although components are placed deep in the hierarchy, they are all at the same depth. Common files are in the common subdirectory, at the same level as the components. Bus definitions are at the same depth, in a parallel directory.





## Providing System-Level Models for the IP

If you have system-level IP like microprocessors and peripherals, you can additionally provide system-level models for your protected IP. This allows users to assemble your IP as part of a subsystem, using the Synplify Pro and Synplify Premier system-level assembly tool, System Designer.

This tool reads IP-XACT models (an XML schema) as defined by the SPIRIT Consortium ([www.spiritconsortium.org](http://www.spiritconsortium.org)). The initial release of System Designer will be Spirit v1.2 compliant with an expectation that version 1.4 will be supported in 2008.

## Supplying Vendor Information

To make your IP accessible for downloads and evaluation from the Synopsys synthesis tools, you must supply Synopsys with some vendor information as well as information for each of the cores or IPs to be used.

1. Supply Synopsys with the following general information to advertise your company and IP on the Synopsys website:

|                         |                                                                                                                                                                                                                                                              |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IP vendor name and logo | Your vendor name and logo for display.                                                                                                                                                                                                                       |
| Optional IP description | Short paragraph describing the IP and key features.                                                                                                                                                                                                          |
| Email alias             | Synopsys sends leads to this alias when evaluation cores are requested on the Synopsys IP website.                                                                                                                                                           |
| Website URL             | Unique URL for accessing IP. After the user has filled out lead information on the website, the Synopsys tool directs the user to this URL to download the IP. The lead form on your website can be pre-filled by prior arrangement with Synopsys Marketing. |

2. Supply Synopsys with the following information about each core or IP to be used:

---

|                                    |                                                                                                                                                                                                      |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IP name                            | Name of the IP.                                                                                                                                                                                      |
| IP short description               | Sentence describing the IP, which is displayed in the summary view on the Synopsys website.                                                                                                          |
| IP paragraph description           | More detailed description of the IP, covering functional description and compatibility with other cores or peripherals.                                                                              |
| Notes about usage                  | Any other information, like licensing requirements                                                                                                                                                   |
| Core datasheet (HTML or PDF)       | Information about the characteristics, features, functions, and interfaces.                                                                                                                          |
| Supported FPGA vendors and devices | List of the targeted vendors and devices that the core supports.                                                                                                                                     |
| IP-XACT compatibility information  | List of the IP-XACT version number supported, the IP-XACT VLNV, and the IP-XACT VLNVs of all the bus definitions required for the core, along with a link to download each of these bus definitions. |

---

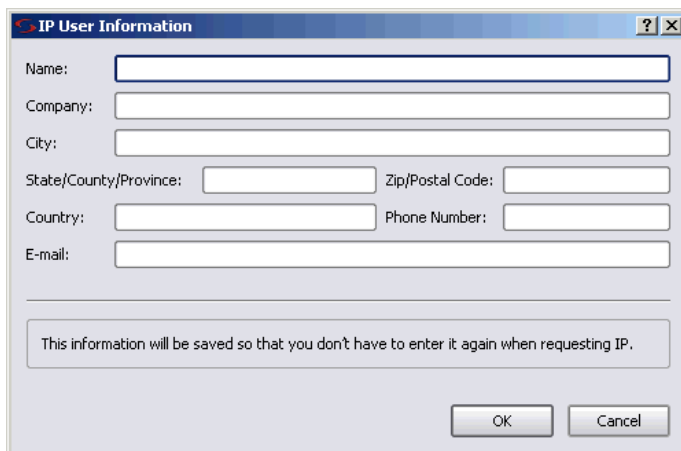
## Evaluating Vendor IP

The Synplify Pro and Synplify Premier synthesis tools facilitate the connection between the IP vendor and the IP user by offering a mechanism through which you can evaluate and use available vendor IP in a synthesis project.

To evaluate vendor IP, use this procedure:

1. In the synthesis tool, select Import IP->Browse and Download ReadyIP.

The IP User Information dialog box opens.

A screenshot of a dialog box titled "IP User Information". The dialog box has a blue header bar with the title and standard window controls (minimize, maximize, close). Below the header, there are several text input fields: "Name:", "Company:", "City:", "State/Country/Province:", "Zip/Postal Code:", "Country:", "Phone Number:", and "E-mail:". Each field is represented by a white rectangular box with a thin border. Below the input fields, there is a message box with a light gray background and a thin border, containing the text: "This information will be saved so that you don't have to enter it again when requesting IP." At the bottom right of the dialog box, there are two buttons: "OK" and "Cancel".

2. Fill out your information, and click OK.

You are directed to a website with offerings from IP vendors.

3. For RTL-based IP, do the following:

- Download the IP package from the vendor website. Usually, the download is a compressed tar archive (.tar.gz) or a zip (.zip) file. Each IP core is a separate package.
- Unzip each package into a unique directory. The tools can read a .zip or .sar file directly, but you must unzip a tar archive first. Unzipping is recommended, because if you read in a .zip or .sar package directly, everything in that package is imported into the project.
- Include the IP in a new or current synthesis project.

4. For IP-XACT-based IP, do the following:

- Create or open an existing synthesis project.
- Select Import IP->Launch System Designer, and open the System Designer tool.
- Use the System Designer tool to read in the .xml files and assemble the IP components into a subsystem. See the System Designer documentation for details.

5. Synthesize your design.

- If you are working with IP-XACT models, assemble the components into a subsystem as described in the previous step. Then synthesize the design.
- If you are not working with subsystem components, run synthesis using the encrypted RTL files and unencrypted constraints from the unzipped IP directory.

The content of the evaluation IP is solely determined by the vendor. Depending on the amount of information in the model IP, you can run complete synthesis of the IP, or only do a more limited evaluation of its compatibility, because it is treated as a black box.

6. After you have completed your evaluation, contact the vendor to license the IP.

# Working with Altera IP

You can incorporate and synthesize different kinds of Altera IP in your synthesis design. See the following for details:

- [Using Altera LPMs or Megafunctions in Synthesis](#), on page 161
- [Implementing Megafunctions with Clearbox Models](#), on page 165
- [Implementing Megafunctions with Grey Box Models](#), on page 175
- [Including Altera MegaCore IP Using an IP Package](#), on page 181
- [Including Altera Processor Cores Generated in SOPC Builder](#), on page 186
- [Working with SOPC Builder Components](#), on page 191

## Using Altera LPMs or Megafunctions in Synthesis

You can include Altera LPMs or Megafunctions in your Synplify Pro and Synplify Premier design in the following ways:

- Generate structural Verilog/VHDL for the IP and include it in your design, as described in [Recommended Method for Including Altera LPMs](#), on page 162.
- For newer Altera technologies, the synthesis tool can infer the Clearbox or greybox megafunctions as described in [Automatically Inferring Megafunctions with Clearbox Information](#), on page 166 and [Using Clearbox Information for Instantiated Megafunctions](#), on page 170
- For older Altera technologies, if you have a Clearbox netlist generated by the Quartus tool for the IP, include the Clearbox netlist in the project, and synthesize the design. See [Instantiating Clearbox Netlists for Megafunctions](#), on page 173.
- Infer Altera LPMs or megafunctions included in the `<install_dir>/lib/altera` directory. This lets the Synplify Pro and Synplify Premier tools access supported LPMs and Megafunctions as required. Note that if you are using a VHDL component from a non-default Quartus library, you must set the Quartus version and add the library file you want to use to the `.prj` file with an `-add file` command.

Currently, physical synthesis only supports LPMs and Megafunctions for Stratix II, Stratix II GX, and Stratix III devices. Note, at this time the Synplify Premier software cannot handle the megafunction `alt_pll` component. This megafunction is treated as a black box.

## Recommended Method for Including Altera LPMs

The following is the recommended methodology for incorporating the LPM or megafunction, where you generate structural Verilog or VHDL files for the megafunction.

1. Use the Altera MegaWizard Plug-in Manager to generate structural Verilog or VHDL files for the LPMs or megafunctions. See [LPM / Megafunction Example, on page 163](#) for an example.
2. Add the structural Verilog or VHDL files in the project. For the example, you would add `my_ram.v` file to the project.
3. Instantiate the LPMs or megafunctions with a wrapper in your top-level HDL source code.

For the example in [LPM / Megafunction Example, on page 163](#), instantiate the megafunction wrapper, `my_ram`, in the top-level HDL source code.

```
module top (
 ...
);

my_ram my_ram_instantiation (
 .address(),
 .clock(),
 .data(),
 .wren(),
 .q()
);

endmodule // top
```

4. Select the correct Quartus version.

This ensures that the synthesis tool accesses the appropriate port and parameter definitions for these LPMs or megafunctions. If you need to, ensure that existing megafunction wrappers comply with the latest applicable version of the Quartus II place-and-route tool, by updating the wrappers with this Quartus command:

```
qmegawiz -silent
```

## 5. Synthesize the design.

When the physical synthesis tool encounters an ALTSYNCRAM megafunction, it automatically executes a Quartus function which determines how to implement the component type and defparams, and how to write out the contents in the final netlist (.vqm).

For this example, the Synplify Premier software writes out a `stratixii_ram_block` primitive for this component in the final .vqm netlist.

```
stratixii_ram_block altsyncram_component_ram_blockla_0_0_0_Z (
 .portadatain({data_c[0]}),
 .portaaddr({address_c[7], address_c[6], address_c[5], address_c[4],
address_c[3], address_c[2], address_c[1], address_c[0]}),
 .portawe(wren_c),
 .clk0(clock_c),
 .portadataout({q_c[0]})
);
defparam altsyncram_component_ram_blockla_0_0_0_Z.connectivity_checking = "OFF";
defparam altsyncram_component_ram_blockla_0_0_0_Z.init_file =
 "C:/public/qinghong/ram_init/rev_1/init_values.mif";
defparam altsyncram_component_ram_blockla_0_0_0_Z.init_file_layout = "port_a";
defparam altsyncram_component_ram_blockla_0_0_0_Z.logical_ram_name = "ALTSYNCRAM";
defparam altsyncram_component_ram_blockla_0_0_0_Z.operation_mode = "single_port";
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_address_width = 8;
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_data_out_clear = "none";
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_data_out_clock =
 "clock0";
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_data_width = 1;
defparam
 altsyncram_component_ram_blockla_0_0_0_Z.port_a_disable_ce_on_input_registers =
 "on";
defparam
 altsyncram_component_ram_blockla_0_0_0_Z.port_a_disable_ce_on_output_registers
 = "on";
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_first_address = 0;
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_first_bit_number = 0;
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_last_address = 255;
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_logical_ram_depth
 = 256;
defparam altsyncram_component_ram_blockla_0_0_0_Z.port_a_logical_ram_width = 4;
defparam altsyncram_component_ram_blockla_0_0_0_Z.power_up_uninitialized =
 "false";
defparam altsyncram_component_ram_blockla_0_0_0_Z.ram_block_type = "M512";
defparam altsyncram_component_ram_blockla_0_0_0_Z.lpm_type
 = "stratixii_ram_block";
```

## LPM / Megafunction Example

The following example shows the megafunction wrapper and the associated defparams generated by the Altera MegaWizard Plug-in Manager for a single-port RAM megafunction, ALTSYNCRAM.

```

// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: Wm1.0
// MODULE: altsyncram

// =====
// File Name: my_ram.v
// Megafunction Name(s):
// altsyncram
//
// Simulation Library Files(s):
// altera_mf
// =====
// *****
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 7.0 Build 33 02/05/2007 SJ Full Version
// *****

//Copyright (C) 1991-2007 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors. Please refer to the
//applicable agreement for further details.

// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module my_ram (
 address,
 clock,
 data,
 wren,
 q);

 input[7:0] address;
 input clock;
 input[3:0] data;
 input wren;
 output[3:0] q;

 wire [3:0] sub_wire0;
 wire [3:0] q = sub_wire0[3:0];

 altsyncram altsyncram_component (
 .wren_a (wren),
 .clock0 (clock),
 .address_a (address),
 .data_a (data),
 .q_a (sub_wire0),
 .aclr0 (1'b0),
 .aclr1 (1'b0),
 .address_b (1'b1),
 .addressstall_a (1'b0),

```



```

 .addressstall_b (1'b0),
 .byteena_a (1'b1),
 .byteena_b (1'b1),
 .clock1 (1'b1),
 .clocken0 (1'b1),
 .clocken1 (1'b1),
 .clocken2 (1'b1),
 .clocken3 (1'b1),
 .data_b (1'b1),
 .eccstatus (),
 .q_b (),
 .rden_a (1'b1),
 .rden_b (1'b1),
 .wren_b (1'b0));
defparam
 altsyncram_component.clock_enable_input_a = "BYPASS",
 altsyncram_component.clock_enable_output_a = "BYPASS",
 altsyncram_component.init_file = "init_values.mif",
 altsyncram_component.intended_device_family = "Stratix II",
 altsyncram_component.lpm_type = "altsyncram",
 altsyncram_component.numwords_a = 256,
 altsyncram_component.operation_mode = "SINGLE_PORT",
 altsyncram_component.outdata_aclr_a = "NONE",
 altsyncram_component.outdata_reg_a = "CLOCK0",
 altsyncram_component.power_up_uninitialized = "FALSE",
 altsyncram_component.ram_block_type = "M512",
 altsyncram_component.widthad_a = 8,
 altsyncram_component.width_a = 4,
 altsyncram_component.width_byteena_a = 1;
endmodule

```

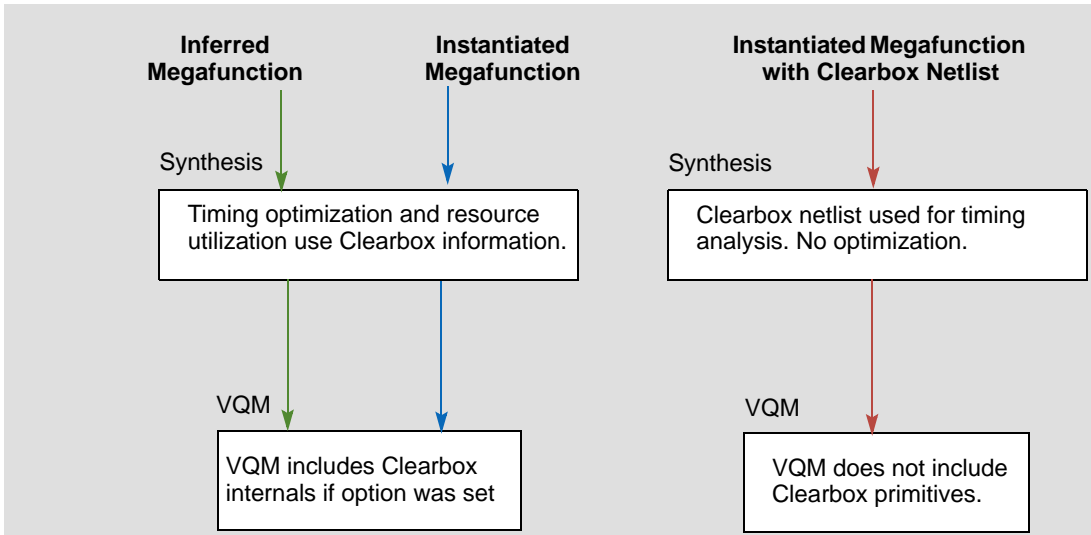
## Implementing Megafunctions with Clearbox Models

Generally, user-instantiated Quartus megafunctions do not have timing information and are treated as black boxes, so the synthesis tool cannot optimize timing at the megafunction boundary. For example, the synthesis software does not move the registers of a pipelined LPM\_MULT to improve FMAX. Instead of black boxes, you can implement the megafunctions as grey boxes (see [Implementing Megafunctions with Grey Box Models, on page 175](#)) or clear boxes, as described here. The Synplify software does not support this flow.

Altera Clearbox netlists provide structural information for modules containing the following primitives `lcell`, `mac_mult`, `mac_out`, and `ram_block`. The Clearbox netlist is a fully synthesizable Altera megafunction. When you synthesize with a clear box model, you get better timing and resource utilization estimates, because the synthesis tool knows the architectural details used in the Quartus II software. For details, see the following:

- [Automatically Inferring Megafunctions with Clearbox Information](#), on page 166

- [Using Clearbox Information for Instantiated Megafunctions](#), on page 170
- [Instantiating Clearbox Netlists for Megafunctions](#), on page 173



## Automatically Inferring Megafunctions with Clearbox Information

Use this method with the newer Altera families. With this method, you do not have to explicitly do anything with the Clearbox megafunction, as the synthesis tool automatically infers the megafunction and calls Quartus for the supporting Clearbox details.

1. Structure the RTL so that the synthesis tool can infer the megafunctions from the code.

The following table lists some tips for controlling inference:

|                           |                                                                                                                                                                                                                                                                                           |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Multipliers in DSP blocks | Use <code>syn_multstyle</code> to control inference.                                                                                                                                                                                                                                      |
| ROMs                      | <ul style="list-style-type: none"> <li>• The address line must be at least two bits wide.</li> <li>• The ROM must be at least half full.</li> <li>• A <code>CASE</code> or <code>IF</code> statement must make 16 or more assignments using constant values of the same width.</li> </ul> |
| Shift registers           | Use <code>syn_srlstyle</code> to control inference.                                                                                                                                                                                                                                       |

## RAMs

- The address line must be at least two bits wide.
- Do not have resets on the memory.
- Check whether read and write ports must be synchronous for your target family.
- Avoid blocking statements when modeling the RAM, because not all Verilog HDL blocking assignments are mapped to RAM blocks.
- Use `syn_ramstyle` to control inference.
- Use `$readmemb` or `$readmemh` to initialize RAMs.

---

See the *Reference Manual* for details about the attributes.

2. Set up the synthesis tool to use the clearbox information.

- Make sure the `QUARTUS_ROOTDIR` environment variable is set and pointing to the same Quartus version as the library.
- In the synthesis tool, open the Implementation Options dialog box to the Device tab, and set the synthesis tool to a supported family:

|                              |                                                         |
|------------------------------|---------------------------------------------------------|
| Synplify Pro                 | Stratix II, Stratix III, Stratix IV, Arria II, Arria GX |
| Synplify Premier (placement) | Stratix II, Stratix III, Stratix IV                     |

---

- On the same tab, check that Verification Mode is disabled.
- Set the Altera Models device option.

| To generate vqm that contains...                                                               | Set it to...  |
|------------------------------------------------------------------------------------------------|---------------|
| The contents of the megafunction as well as grey box netlists for any grey boxes in the design | on            |
| The contents of the megafunction                                                               | clearbox_only |
| The megafunction without its contents                                                          | off           |

---

- Click OK.

3. Set any other options you want, and click Run to synthesize the design.

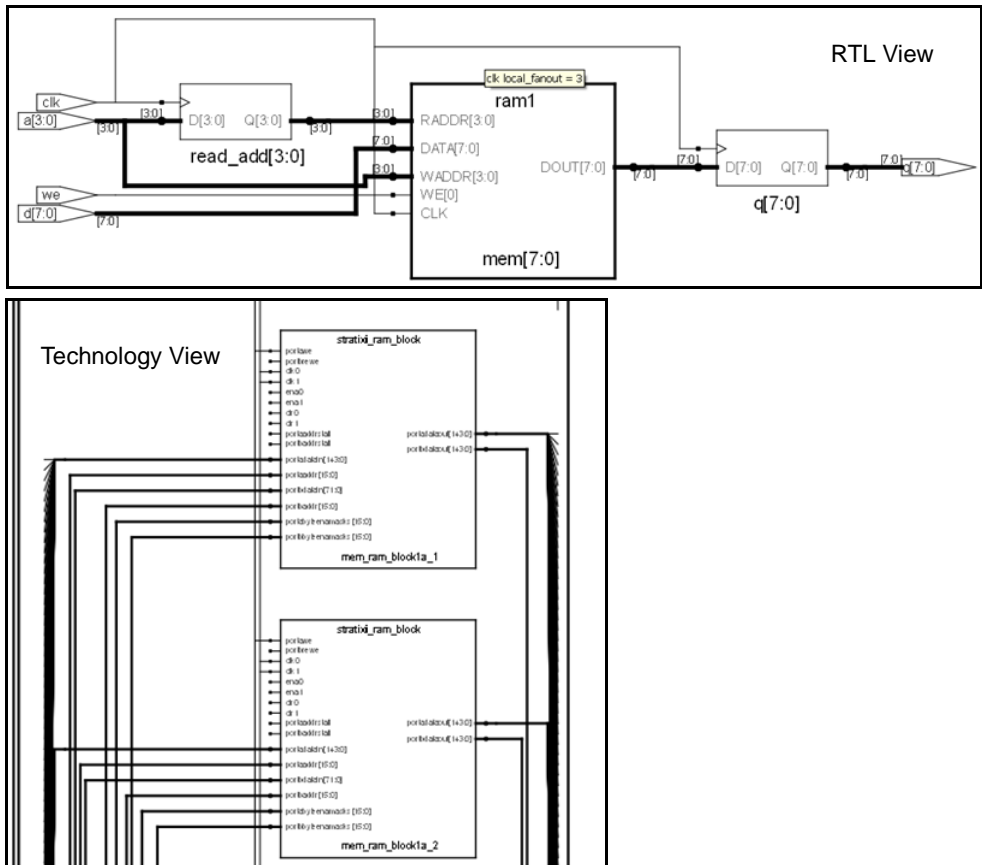
The synthesis tool infers the megafunction from the RTL code. For example, it infers a RAM from this code:

```
module ram(q, a, d, we, clk);
output[7:0] q;
input [7:0] d;
input [3:0] a;
input we, clk;
reg [7:0] q ;
reg [3:0] read_add;
reg [7:0] mem [0:15];

always @(posedge clk) begin
q = mem[read_add];
end

always @(posedge clk) begin
if(we)
mem[a] <= d;
read_add <= a;
end
endmodule
```

It then calls the Clearbox executable which returns a netlist containing the Clearbox internals for the inferred megafunction (based on the Altera Models setting). The synthesis tool uses this information to optimize timing and allocate resources. The RTL view shows the generic memory inferred, but the Technology view shows some of the `stratixii_ram_block` Clearbox primitives that were implemented after calling the Clearbox executable.



The tool writes out the Clearbox information in the vqm netlist, according to the Altera Models setting. In addition, for the alt2gxb\_reconfig, altasm\_parallel, altpll\_reconfig, and altremote\_update megafunctions, the synthesis tool writes out the vqm exactly as generated by the Altera Megawizard. The Altera tool defines the lower levels of content for these megafunctions ("clearbox=2" setting) with the parameters set for the megafunction, and that is how they are written to the vqm.

4. Use this vqm file to place and route in Quartus II.

## Using Clearbox Information for Instantiated Megafunctions

There are two ways of instantiating megafunctions and using Clearbox information in your synthesis design. The following is the recommended method for instantiation in the newer Altera technologies, where you instantiate just the megafunction, and the tool automatically infers the corresponding Clearbox details. For older technologies, instantiate the Clearbox netlist for the megafunction, as described in [Instantiating Clearbox Netlists for Megafunctions, on page 173](#).

1. Generate the Verilog or VHDL megafunction using the Altera Megafunction wizard.

This is just the megafunction file, not a Clearbox primitive netlist.

2. Set up the megafunction for synthesis.
  - Instantiate the megafunction in your synthesis design.
  - Add the megafunction wrapper file to your project file.
3. Set up the synthesis tool to use the Clearbox information automatically.
  - Make sure the QUARTUS\_ROOTDIR environment variable is set and pointing to the same Quartus version as the library.
  - In the synthesis tool, open the Implementation Options dialog box to the Device tab, and set the synthesis tool target to a supported family:

|              |                                                         |
|--------------|---------------------------------------------------------|
| Synplify Pro | Stratix II, Stratix III, Stratix IV, Arria II, Arria GX |
|--------------|---------------------------------------------------------|

|                              |                                     |
|------------------------------|-------------------------------------|
| Synplify Premier (placement) | Stratix II, Stratix III, Stratix IV |
|------------------------------|-------------------------------------|

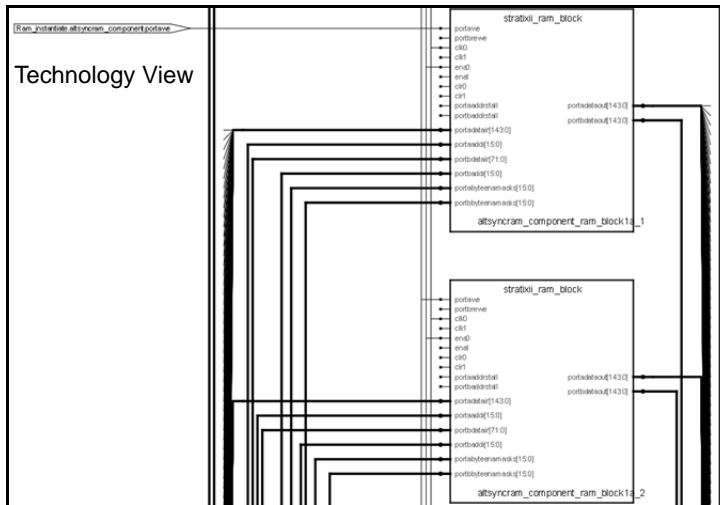
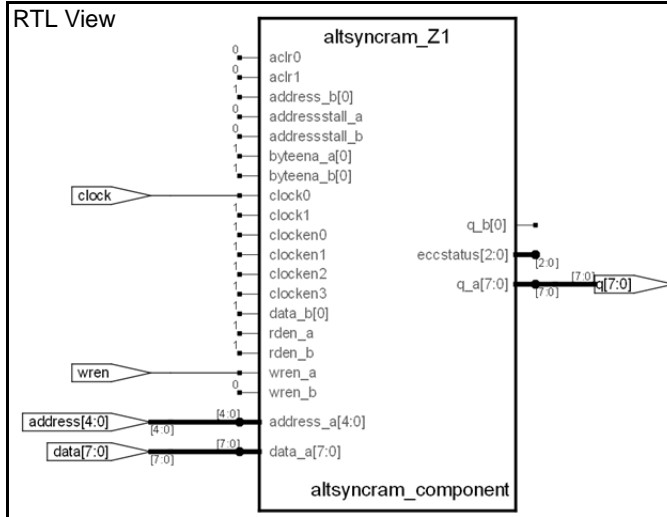
- On the same tab, check that Verification Mode is disabled.
- Set the Altera Models device option.

| To generate vqm that contains...                                                               | Set it to...  |
|------------------------------------------------------------------------------------------------|---------------|
| The contents of the megafunction as well as grey box netlists for any grey boxes in the design | on            |
| The contents of the megafunction                                                               | clearbox_only |
| The megafunction without its contents                                                          | off           |

- Click OK.
4. Set any other options you want, and click Run to synthesize the design.

The tool instantiates the megafunction and calls the Clearbox executable, which returns a netlist containing the Clearbox internals for the megafunction (based on the Altera Models setting). The synthesis tool uses this information to optimize timing and allocate resources.

The RTL view below shows an instantiated megafunction, ALTSYNCRAM. The corresponding Technology view shows the stratixii\_ram\_block Clearbox primitives. The tool generated the Clearbox information for the instantiated megafunction by calling Quartus.



The tool writes out the Clearbox information in the vqm netlist, according to the Altera Models setting. In addition, for the `alt2gxb_reconfig`, `altasmi_parallel`, `altpll_reconfig`, and `altremote_update` megafunctions, the synthesis tool writes out the vqm exactly as generated by the Altera Megawizard. The Altera tool defines the lower levels of content for these megafunctions ("`clearbox=2`" setting) with the parameters set for the megafunction, and that is how they are written to the vqm.

##### 5. Use the vqm file to place and route in Quartus II.



## Instantiating Clearbox Netlists for Megafunctions

For older Altera technologies which do not supported the automatic inference of Clearbox information (see [Automatically Inferring Megafunctions with Clearbox Information](#), on page 166 and [Using Clearbox Information for Instantiated Megafunctions](#), on page 170 ), you can read in a Clearbox netlist. You only need to use the netlist method described here for older Altera target families that do not support the other flows.

1. Generate the megafunction files with a Clearbox netlist.
  - Use the Altera Megafunction wizard to generate structural VHDL or Verilog files for the megafunctions in your design. Make sure the Synthesized Timing Netlist option is disabled. The Clearbox netlist has the full content of the megafunction either in VHDL or Verilog. The synthesis software uses this timing and resource information for the megafunctions, but does not synthesize the internals of the megafunctions.
  - If you are using VHDL, comment out the LIBRARY and USE clauses in the file generated by the Altera MegaWizard tool. This is because because the Altera MegaWizard file declares the Clearbox components before instantiating them, so you do not need references to the vhd files that contain the component declarations. The following shows a Stratix example of the lines to be commented out; for other technologies, comment out the corresponding lines:

```
LIBRARY stratix;
USE stratix.all;
```

- Make sure the Clearbox components match the Quartus version. Because of ongoing modifications in Quartus, the component declarations may not match. The component declarations are packaged with the software in the `lib/altera/quartus_11nn` subdirectory. Use the file from the subdirectory that corresponds to the Quartus version that you are using. For example, the `stratix.vhd` and `stratix.v` files for use with Quartus 8.1 are in the `quartus_1181` subdirectory.
  - If you change from one version of Quartus to another or if you change the target device, regenerate the Clearbox files using the Altera Megafunction wizard before proceeding. Failure to regenerate these files can result in a parameter-mismatch error.
2. Instantiate the megafunction in your design.

3. Add the megafunction file (which includes the Clearbox components) to your project.
  - Add the Clearbox Verilog/VHDL file to your project. It contains the port and parameter definitions for the Clearbox primitives.
  - If you are using Verilog, the software does not automatically include the definitions because Verilog does not support library statements.
4. Set implementation options for the megafunction.
  - Click Implementation Options, and set the target technology on the Device tab.
  - On the Implementation Results tab, select the appropriate Quartus version. This is important, because the version determines the format of the output .vqm file, which varies with different Quartus versions. If you are using Verilog, the software automatically adds the Verilog component declaration files for the selected target technology to your project from the lib/altera/quartus\_11nn subdirectory. Failing to specify the version can result in a parameter mismatch error.
  - Click OK.
5. Optionally, set up the files so that you can run Quartus from the synthesis tool by doing either of the following:
  - Select the Clearbox file from the project file list, right-click and select File Options. Set File Type to Clearbox Verilog or Clearbox VHDL and click OK.
  - Use the Tcl command appropriate to your file type:

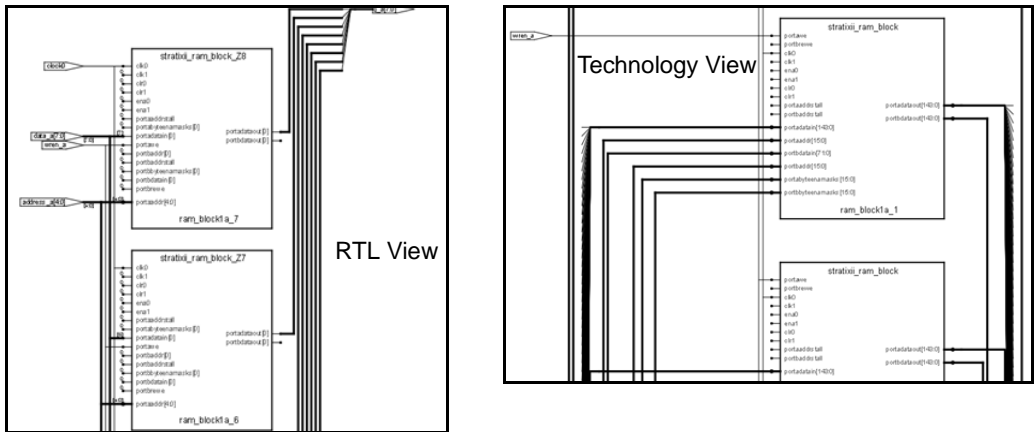
```
add_file -clearbox_verilog "dsp/my_dsp_syn.v"
add_file -clearbox_vhdl "dsp/my_dsp_syn.vhd"
```

When you run Quartus from the synthesis UI, the Clearbox files must be in the *<implementation>/par\_1* directory, which is only created after the synthesis run. Specifying the Clearbox files ensures that they are copied to the directory after it is created. You can now synthesize the design, as described in the previous step.

6. Set any other options and synthesize the design.

The software uses the Clearbox timing and resource information from the structural files to calculate paths more accurately. It implements the megafunctions as hierarchical instances, not black boxes. The RTL and

Technology views both show the lowest-level primitives. The following figure for example, shows stratixii\_ram\_blocks.



The .vqm file generated for Quartus after synthesis only contains a wrapper; it does not include the Clearbox primitives. The description of the primitives is in the Clearbox netlist generated in step 1 and used as input to synthesis.

7. Before you run Quartus, put all these files in the same result directory:
  - The structural Verilog/VHDL Clearbox netlist generated by Quartus and used as input to synthesis. This file contains timing and resource usage definitions for the primitives.
  - The .vqm file generated after synthesis, which contains the wrapper.
  - The Quartus project file.

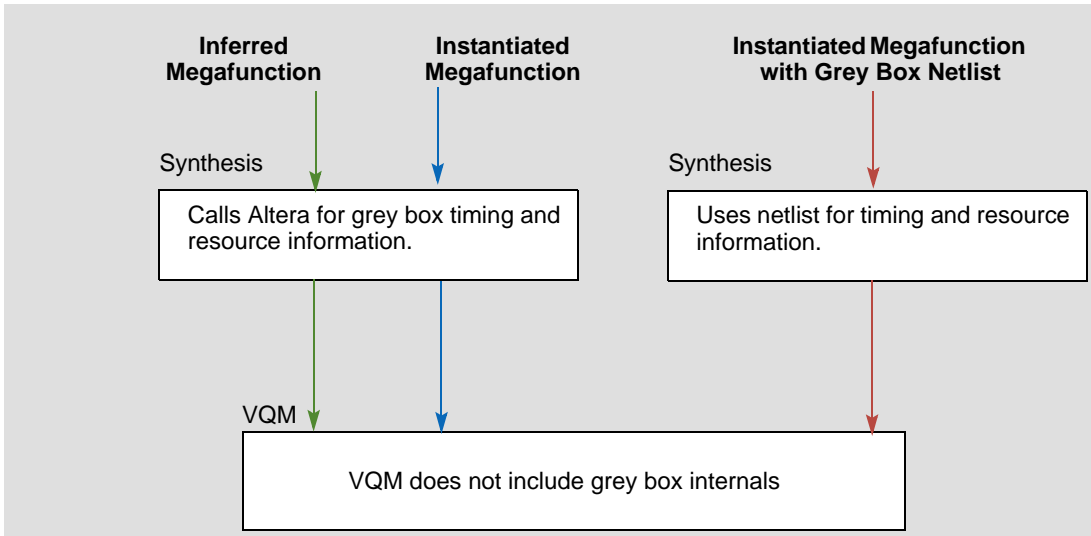
Placing these files in the same directory ensures that the Quartus software can find all the information it needs in the .vqm file and the original structural Verilog/VHDL files.

## Implementing Megafunctions with Grey Box Models

Altera provides the capability of implementing various MegaCore® IP cores generated with the Altera MegaWizard tool. These cores use proprietary RTL code for parameterization, generation, and instantiation in your design. Generally, user-instantiated Quartus megafunctions do not come with any timing information and are treated as black boxes, so the synthesis tool

cannot optimize timing at the megafunction boundary. Instead of using black boxes, you can implement the megafunctions using Clearbox primitives (see [Implementing Megafunctions with Clearbox Models, on page 165](#)) or as grey boxes, as described here. Use the grey box methodology when logic is encrypted or when there are no Clearbox models for the megafunction.

There are three ways to use grey boxes:



The following procedures show you how to implement an Altera megafunction as a grey box in the Synplify Pro and Synplify Premier tools. The Synplify software does not support grey box flows.

- [Automatically Using Grey Box Information for Megafunctions](#), on page 176
- [Using Grey Box Information for Instantiated Megafunctions](#), on page 177
- [Instantiating Megafunctions Using Grey Box Netlists](#), on page 179

## Automatically Using Grey Box Information for Megafunctions

1. Structure the RTL so that the synthesis tool can infer the megafunctions from the code.
2. Set up the synthesis tool to use the clearbox information.

- Make sure the `QUARTUS_ROOTDIR` environment variable is set and pointing to the same Quartus version as the library.
- In the synthesis tool, open the Implementation Options dialog box to the Device tab, and set the synthesis tool to a supported family:

|              |                                                         |
|--------------|---------------------------------------------------------|
| Synplify Pro | Stratix II, Stratix III, Stratix IV, Arria II, Arria GX |
|--------------|---------------------------------------------------------|

|                                 |                                     |
|---------------------------------|-------------------------------------|
| Synplify Premier<br>(placement) | Stratix II, Stratix III, Stratix IV |
|---------------------------------|-------------------------------------|

- On the same tab, check that Verification Mode is disabled.
  - To use grey box timing information, set Altera Models device option to on.
  - Click OK.
3. Set any other options you want, and click Run to synthesize the design.

The synthesis tool infers the megafunction from the RTL code. It then calls the Altera grey box executable which returns a netlist containing the timing and resource information for the inferred megafunction. The synthesis tool uses this information to optimize timing and allocate resources. The RTL view shows the generic memory inferred, but the Technology view shows the primitives that were implemented after calling the grey box executable.

The tool does not include the grey box information in the output vqm netlist. (

4. To place and route in Quartus II, use the following files:
  - The synthesis vqm output netlist
  - The encrypted file for the megafunction

## Using Grey Box Information for Instantiated Megafunctions

There are two ways of instantiating grey box megafunctions in your synthesis design. The following procedure shows you how to instantiate a grey box megafunction without a grey box netlist; to instantiate one with a grey box netlist, refer to the procedure in [Instantiating Megafunctions Using Grey Box Netlists](#), on page 179.

1. Generate the Verilog or VHDL megafunction using the Altera Megafunction wizard.

This is just the megafunction wrapper file, and does not include a grey box netlist.

2. Set up the megafunction for synthesis.
  - Instantiate the megafunction in your synthesis design.
  - Add the megafunction wrapper file to your project file.
3. Set up the synthesis tool to use the grey box information.
  - Make sure the QUARTUS\_ROOTDIR environment variable is set and pointing to the same Quartus version as the library.
  - In the synthesis tool, open the Implementation Options dialog box to the Device tab, and set the synthesis tool target to a supported family:

|              |                                                         |
|--------------|---------------------------------------------------------|
| Synplify Pro | Stratix II, Stratix III, Stratix IV, Arria II, Arria GX |
|--------------|---------------------------------------------------------|

|                                 |                                     |
|---------------------------------|-------------------------------------|
| Synplify Premier<br>(placement) | Stratix II, Stratix III, Stratix IV |
|---------------------------------|-------------------------------------|

- On the same tab, check that Verification Mode is disabled.
  - To use grey box timing information, set Altera Models device option to on.
  - Click OK.
4. Set any other options you want, and click Run to synthesize the design.

The tool instantiates the megafunction and calls the grey box executable, which returns a netlist containing the grey box timing for the instantiated megafunction. The RTL view shows the instantiated megafunction. The corresponding Technology view shows the primitives.

The tool instantiates the megafunction in the output vqm netlist, but does not include the grey box timing information.

5. To place and route in Quartus II, use the following files:
  - The synthesis vqm output netlist, which includes an empty instantiation of the megafunction.
  - The encrypted file for the megafunction.

## Instantiating Megafunctions Using Grey Box Netlists

The following procedure shows you how to use a greybox netlist to incorporate cores in a Synplify Pro or Synplify Premier design. The greybox netlist file is only used for synthesis.

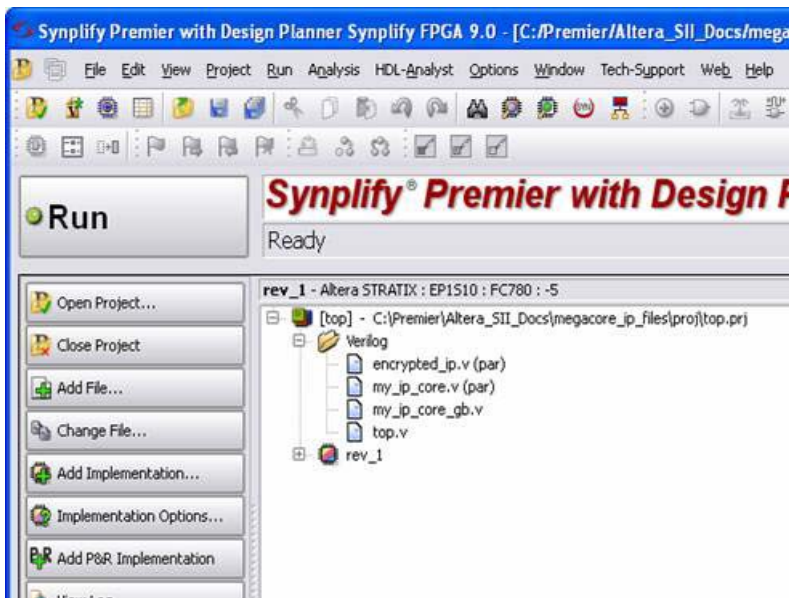
1. Make sure you have installed Quartus II 7.2 or later.
2. Use the Altera MegaWizard tool to generate the files for the IP core and a greybox netlist. The following example shows the files needed for a project:

|                              |                                                                                                                                                                           |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>top.v</code>           | Top level design file. See <a href="#">top.v</a> , on page 185.                                                                                                           |
| <code>my_ip_core.v</code>    | Top-level wrapper generated by the MegaWizard tool, which instantiates the encrypted module <code>encrypted_ip.v</code> . See <a href="#">my_ip_core.v</a> , on page 186. |
| <code>encrypted_ip.v</code>  | Encrypted IP core that is not readable.                                                                                                                                   |
| <code>my_ip_core_gb.v</code> | Greybox netlist that contains mapped instances of the IP core. The LUT masks are scrambled.                                                                               |

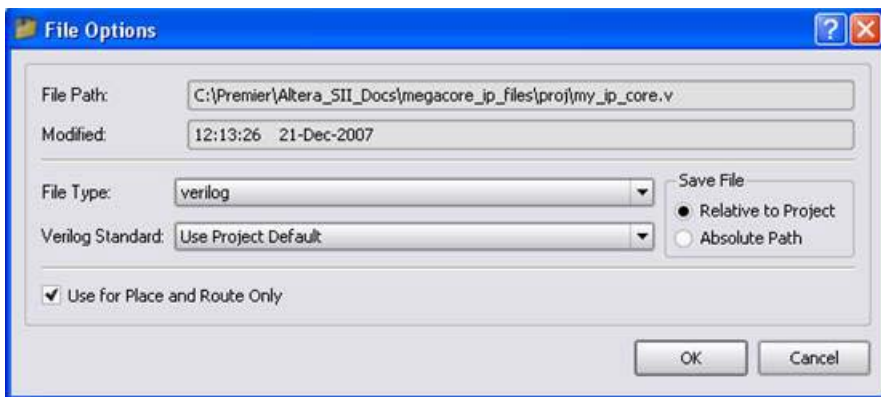
- To generate the greybox netlist, enable the Generate netlist option in the Megawizard tool when you set up simulation. The greybox netlist provides the logic connectivity of specific mapped instances, but does not represent the true functionality of the MegaCore IP.
  - Parameterize the IP core and generate the IP files. The tool outputs a greybox file along with the other synthesis files.
3. Set up your design.
    - Instantiate the megafunction in your synthesis design.
    - In the synthesis tool, add the megafunction wrapper and the grey box netlist files to your synthesis project.
    - If you have the encrypted megafunction file, add that to your project too. It will not be used for synthesis, but passed to P&R.

If your core uses encrypted IP that is part of the Quartus install directory IP (not generated by the MegaWizard IP tool), you do not need to add it to the project. Just make sure you are linked to the appropriate Quartus directory. Paths to these Quartus IP libraries are automatically forward-annotated in a Tcl file for place and route.

The following shows the project view with the IP files added:



4. Designate the files to be passed to the P&R tool, like the wrapper file and the encrypted file. Do the following for each file you want to pass to the P&R tool:
  - Right-click a file in the Project view and select File Options.
  - Enable Use for Place and Route Only in the dialog box and click OK.



- Do this for every IP component file instantiated in the top-level IP wrapper.



The synthesis tool ignores these tagged files for synthesis but copies them to the P&R directory after synthesis is done so that they can be used by the P&R tool.

5. Set implementation options.

- Click Implementation Options and set the target technology on the Device tab.
- Go to the Implementation Results tab and specify the correct version for the place-and-route tool. This is important because the version determines the format for the `vqm` output file, which varies with different versions.
- Set any other options or constraints you want.

6. Synthesize the design.

The synthesis tool uses the greybox netlist file for synthesis and timing analysis. The RTL and Technology views show the internals of the core IP, because the greybox netlist file contains mapped instances. The log file reports any critical paths found within the core, and the Technology view displays timing numbers and critical paths.

After synthesis, the `vqm` netlist does not contain the mapped instances found in the greybox netlist. It only contains a top-level instantiation of `my_ip_core`, not its contents. If you synthesized with the Synplify Premier tool, the placement locations of instances in the greybox netlist are not forward-annotated to the P&R tool.

7. To place and route in Quartus II, use the following files:

- The synthesis `vqm` output netlist.
- The Altera wrapper file for the megafunction (not used for synthesis).
- The encrypted file for the megafunction (not used for synthesis).

## Including Altera MegaCore IP Using an IP Package

You can include MegaCore® IP cores generated with the Altera MegaWizard tool using the method described below, but it is preferred that you use a greybox netlist instead (*Instantiating Megafunctions Using Grey Box Netlists, on page 179*).

For information about including cores generated with SOPC Builder, see [Including Altera Processor Cores Generated in SOPC Builder, on page 186](#).

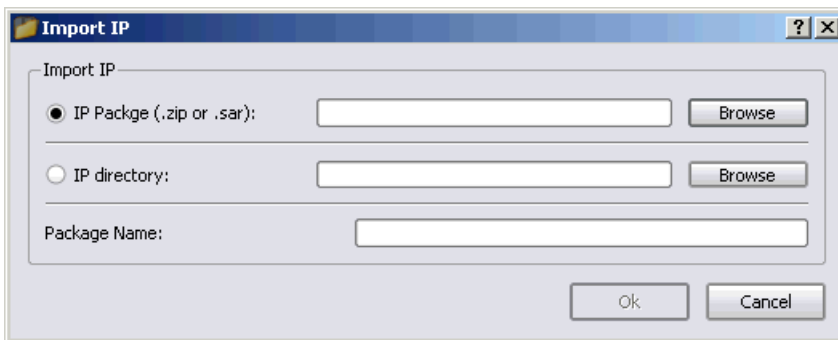
1. Make sure you have installed Quartus II 7.2 or later.
2. Use the Altera MegaWizard tool to generate the files for the IP core. The following is an example of the files needed for the project:

|                  |                                                                                                                                                             |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| top.v            | Top level design file. See <a href="#">top.v, on page 185</a> .                                                                                             |
| my_ip_core.v     | Top-level wrapper generated by the MegaWizard tool, which instantiates the encrypted module encrypted_ip.v. See <a href="#">my_ip_core.v, on page 186</a> . |
| my_ip_core_enc.v | Encrypted IP core that is not readable.                                                                                                                     |

3. Copy the MegaCore IP and associated library files into a single directory.

You can find the library files associated with the IP core in the MegaWizard output directory and in the IP library files in the Quartus installation directory (for example, altera/72/pc\_compiler.lib).

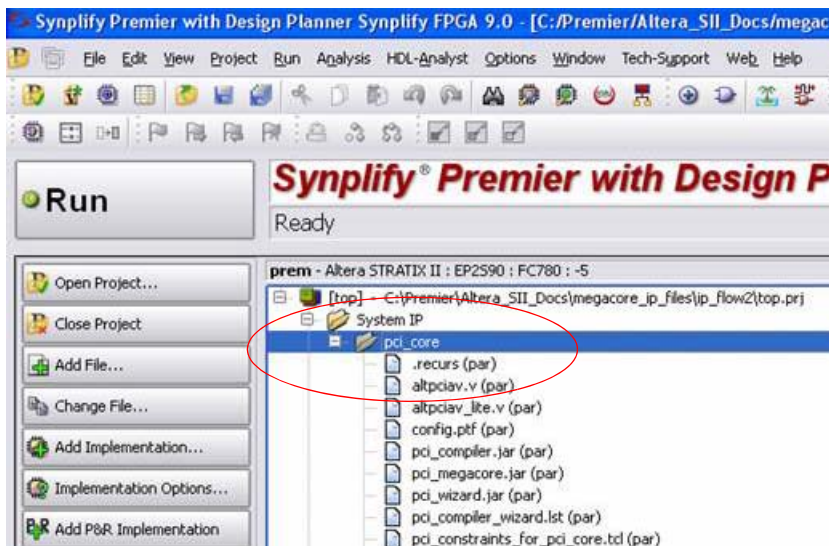
4. Import the core into the synthesis design.
  - Start the synthesis tool, and make sure the technology you are targeting in Synplify Pro or Synplify Premier is either Stratix II, Stratix II-GX, Stratix III, or Stratix IV.
  - In the synthesis UI, select Import IP->Import IP Package.



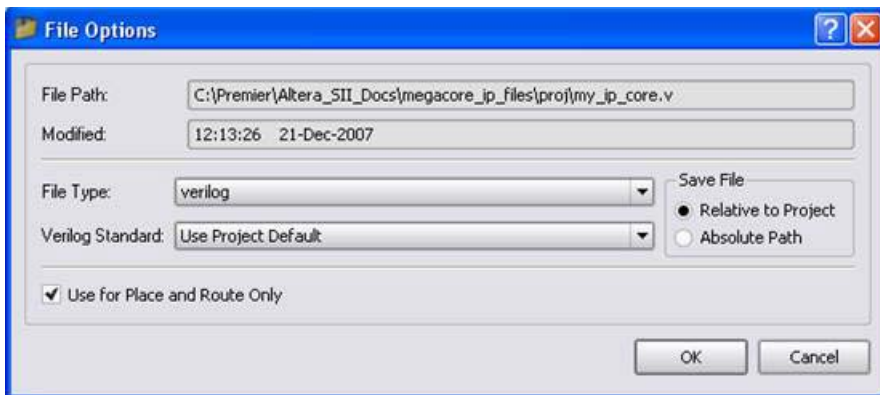
- In the IP Directory field, enter the path to the directory with the consolidated files.

- In the Package Name field, enter the name of the top-level module. In our example, this is my\_ip\_core.
- Click OK.

The tool imports the file and creates a directory called System IP. This includes a sub-directory with the package name (pci\_core in our example), which contains all the IP-related files.

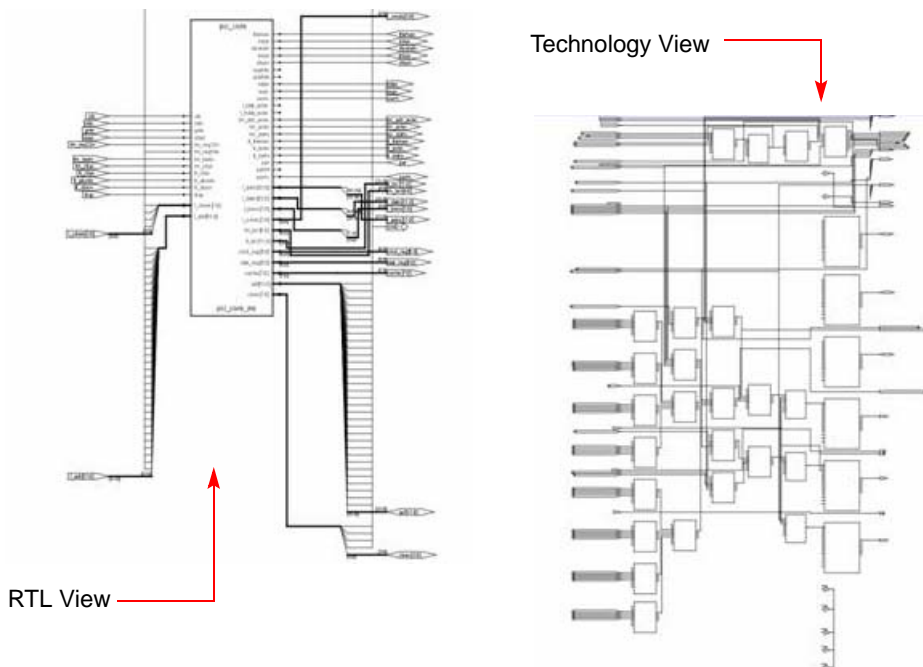


5. Tag the IP component files so that they are not compiled for synthesis. They are used for P&R, but not for synthesis.
  - Right-click a file and select File Options.
  - Enable Use for Place and Route Only in the dialog box and click OK.



- Do this for every IP component file instantiated in the top-level IP wrapper.
6. Instantiate the NIOS core in the top-level file for your synthesis design.
  7. Synthesize the design.

The tool automatically generates a greybox netlist for the IP core, and uses it for timing. It does not use the internals of the core. The RTL view only displays the top-level of the core, but you can view the internals when you push down into the core in the Technology view.



The log file reports any critical paths found within the core, and the Technology view displays timing numbers and critical paths.

After synthesis, the `vqm` netlist that is written out does not contain the mapped instances found in the greybox netlist. It only contains a top-level instantiation of `my_ip_core`, not its contents.

If you synthesized with the Synplify Premier tool, the placement locations of instances in the greybox netlist are not forward-annotated to the P&R tool.

## Examples of MegaCore IP Files for Synthesis

### top.v

The following is a simple example of a top-level MegaCore IP file (`top.v`) that is included in a synthesis project.

```
module top (in1, in2, out1, out2)
```

```
input in1, in2;
output out1, out2;

my_ip_core (in1, in2, out1, out2);

endmodule;
```

### my\_ip\_core.v

The following is a simple example of a top-level MegaCore wrapper file that is included in a synthesis project.

```
module my_ip_core (in1, in2, out1, out2)

input in1, in2;
output out1, out2;

encrypted_ip (in1, in2, out1, out2);

endmodule;
```

## Including Altera Processor Cores Generated in SOPC Builder

Altera provides the capability of implementing configurable processor cores. NIOS® II cores are created using the Altera SOPC Builder tool, which uses proprietary RTL code to parameterize, generate, and instantiate NIOS II cores in your design. The following procedure shows you how to use the greybox flow to incorporate these cores in a Synplify Pro or Synplify Premier design targeting certain Altera technologies. For information about including MegaCore cores, see [Instantiating Megafunctions Using Grey Box Netlists, on page 179](#).

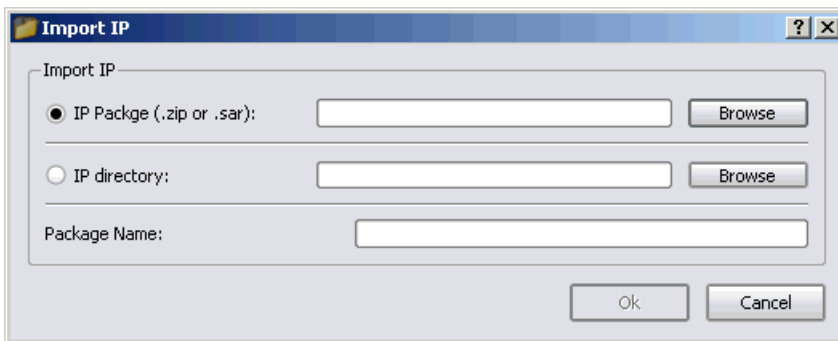
1. Make sure the following requirements are in place:
  - Install Quartus II 7.2 or later.
  - Install the MegaCore® IP library.
2. Generate the NIOS II core files as follows:
  - In Altera SOPC Builder, generate Verilog or VHDL output for the NIOS II core files. Note that this flow does not support a block symbol file (.bsf); you must generate RTL code for the cores. The files may or may not be encrypted.

- Create a top-level wrapper for the core. In most cases, this wrapper instantiates the components used to create the embedded system. For example, if the embedded system consists of a NIOS II processor that uses a PCI bus to interfaces to internal memory, the wrapper would contain instantiations for the processor, memory and the PCI bus.
- Copy all generated IP core output files and the corresponding library files into a single directory. Typically you must include the generated HDL files, as well as any MegaCore IP cores in your design. Depending on your design, the MegaCore IP files can be in the MegaWizard IP tool output directory and in the IP library files in the Quartus install directory (`altera/72/ip/pci_compiler/lib`).

The following example shows the list of files for a design that contains a NIOS II core processor with internal memory that drives an LCD display. This example does not have any MegaCore IP.

| File                                         | Description                                             |
|----------------------------------------------|---------------------------------------------------------|
| <code>top.v</code>                           | User-defined top level of the design                    |
| <code>first_nios2_system_bb.v</code>         | User-defined module definition                          |
| <code>first_nios2_system.v</code>            | Top-level wrapper for SOPC system (SOPC Builder)        |
| <code>cpu.v</code>                           | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>cpu_jtag_debug_module.v</code>         | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>cpu_jtag_debug_module_wrapper.v</code> | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>cpu_mult_cell.v</code>                 | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>cpu_test_bench.v</code>                | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>first_nios2_system.v</code>            | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>jtag_uart.v</code>                     | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>led_pio.v</code>                       | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>onchip_mem.v</code>                    | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>sys_clk_timer.v</code>                 | Module instantiated in top-level wrapper (SOPC Builder) |
| <code>sysid.v</code>                         | Module instantiated in top-level wrapper (SOPC Builder) |

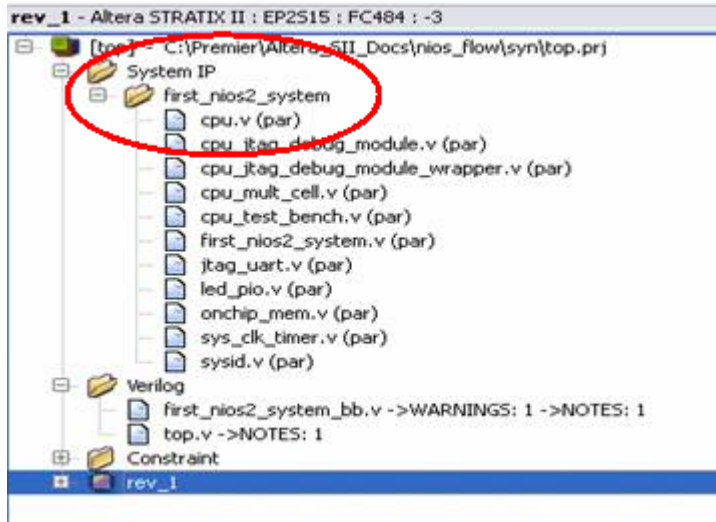
3. Import the core into the synthesis design.
  - Start the synthesis tool, and make sure the technology you are targeting in Synplify Pro or Synplify Premier is either Stratix II, Stratix II-GX, Stratix III, or Stratix IV.
  - In the synthesis UI, select Import IP->Import IP Package.



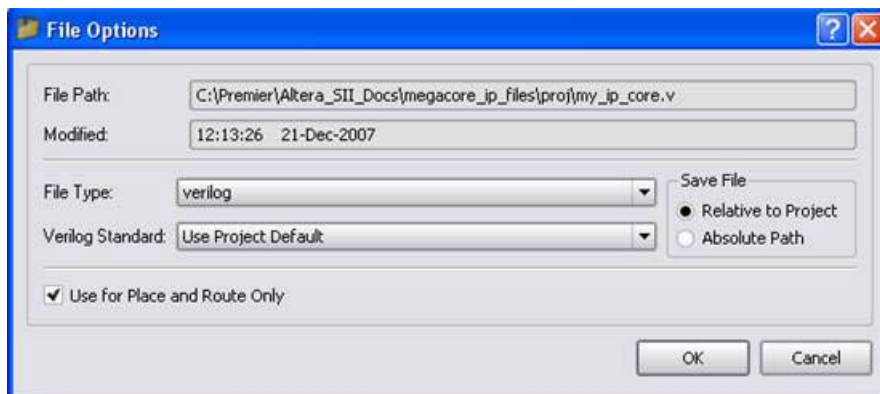
- In the IP Directory field, enter the path to the directory with the consolidated files.
- In the Package Name field, enter the name of the top-level module. In our example, this is `first_nios2_system`.
- Click OK.

The tool imports the file and creates a directory called System IP. This includes a sub-directory with the package name (`first_nios2_system` in our example), which contains all the IP-related files.



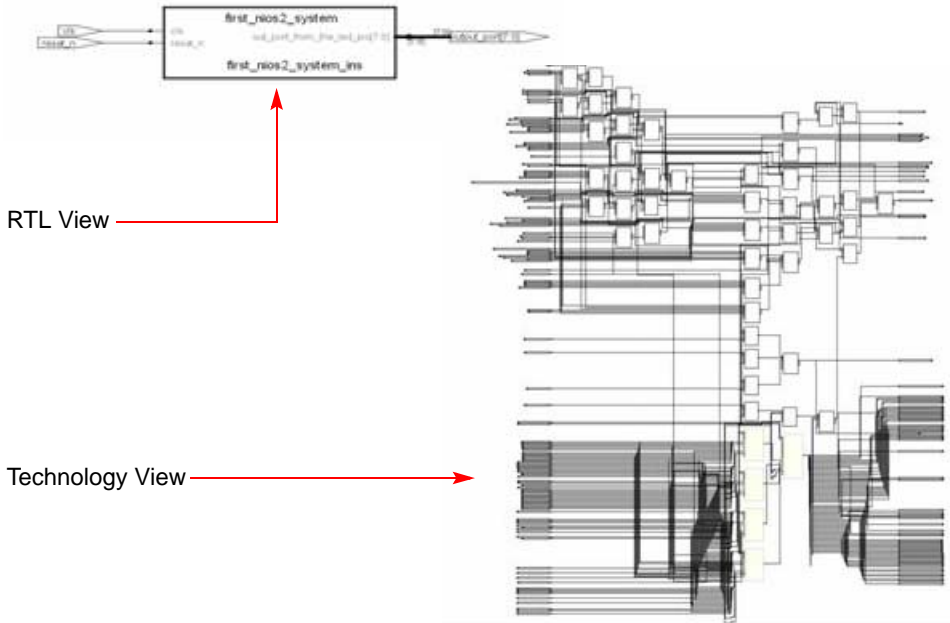


4. Tag the IP component files so that they are not compiled for synthesis. They are used for P&R, but not for synthesis.
  - Right-click a file and select File Options.
  - Enable Use for Place and Route Only in the dialog box and click OK.

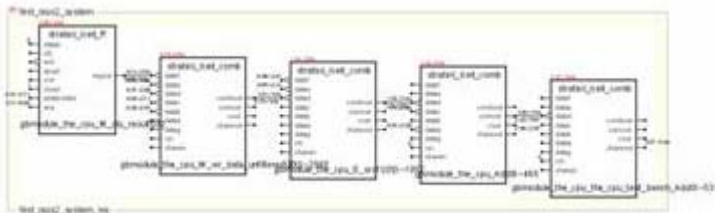


- Do this for every IP component file instantiated in the top-level IP wrapper.
5. Instantiate the NIOS core in the top-level file for your synthesis design.
  6. Synthesize the design.

The tool automatically generates a greybox netlist for the IP core, and uses it for timing. It does not use the internals of the core. The RTL view only displays the top-level of the core, but you can view the internals when you push down into the core in the Technology view.



The log file reports any critical paths found within the NIOS II core:



Worst Path Information  
 \*\*\*\*\*

```

Path information for path number 1:
 Requested Period: 4.000
 - Setup time: 0.187
 + Clock latency at ending point: 0.000
 = Required time: 3.813

 - Propagation time: 6.856
 - Clock latency at starting point: 0.000
 = Slack (critical) : -3.043

Number of logic level(s): 38
Starting point:
first_nios2_system_ins.gbmodule_the_cpu_M_alu_result[0] / regout
Ending point:
first_nios2_system_ins.gbmodule_the_cpu_M_status_reg_pie / datain
The start point is clocked by clk [rising] on pin clk
The end point is clocked by clk [rising] on pin clk

```

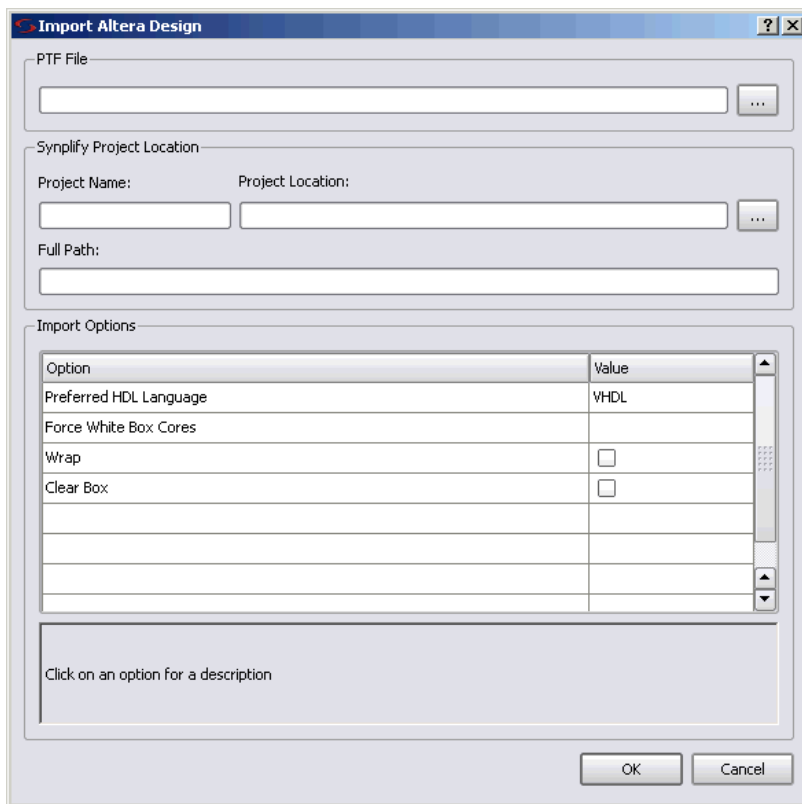
After synthesis, the vqm netlist that is written out does not contain the mapped instances found in the greybox netlist. It only contains a top-level instantiation of `first_nios2_system`, not its contents.

If you synthesized with the Synplify Premier tool, the placement locations of instances in the greybox netlist are not forward-annotated to the P&R tool.

## Working with SOPC Builder Components

If you want to include subsystems created with the Altera SOPC Builder, use the following procedure, which uses the underlying `sopc2syn` utility. The following procedure shows you how to set up a SOPC project for synthesis. For information on incorporating SOPC Builder cores, see [Including Altera Processor Cores Generated in SOPC Builder](#), on page 186.

1. Start the Synplify Pro/Premier software, and select Import IP->Import Altera SOPC Project. This opens the Import Altera Design dialog box.



2. Do the following to import the SOPC project you created with the Altera tools.

- In PTF File, specify the Altera `ptf` file you want to import.
- Specify a location for the synthesis project to be created in Project Location, and a name for the project in Project Name.
- Set the options you want in the Import Options section. Define the black boxes and white boxes in your design with the Black Box, Force Black Box Cores, and Force White Box Cores options, as described in [Specifying SOPC Components as Black Boxes and White Boxes](#), on page 193.

The tool will not complete synthesis if it finds inadequately defined components, and you will have to iterate through synthesis again.

- Click OK.

The software uses the underlying socp2syn functionality (see [The SOPC2Syn Utility](#), on page 629 in the *Reference Manual*) to read the Altera files and include the information from them into the synthesis project.

3. To include an SOPC component as a subsystem in a larger design, create the subsystem as described in the previous steps and instantiate the subsystem in the top-level HDL.

## Specifying SOPC Components as Black Boxes and White Boxes

Accurately defining SOPC components as black boxes and white boxes is very important, because the synthesis run will fail if the design is not correctly specified, and you will have to iterate through another run. A black box is a component that does not have any definitions.

1. In the Synplify Pro/Premier software, and select Import IP->Import Altera Project. This opens the Import Altera Design dialog box.
2. If you want to treat a core as a black box during synthesis, list the component in the Force White Box Cores field, and set disable the Clear Box option.

With these settings, the tool copies the core wrapper file to the Synplify folder and edits it to add the black box attribute. The component is treated as a black box during synthesis. If the tool does not find the named component, it issues a warning message.

3. If you want to treat a core as a white box during synthesis, list the component in the Force White Box field, and enable the Clear Box option.

With these settings, the tool goes through the Clearbox flow, and treats the core as a white box during synthesis. If it does not find the named component or the Clearbox file, it issues a warning message.

## Working with Lattice IP

The Lattice IP encryption scheme uses the flow described in [Encryption and Decryption, on page 144](#). The following procedure details how to incorporate Lattice encrypted IP in your design.

1. Create a synthesis project.
2. Obtain the IP from Lattice, and add the encrypted file to your project.

The Lattice IP is encrypted using the `persistent_key` method described in [Specifying the Output Method for encryptIP, on page 151](#).

3. Synthesize your design.

The tool automatically decrypts the protected IP and synthesizes it with the other unencrypted files in the design. During synthesis, the IP is treated as a black box and you cannot view its contents in HDL Analyst views.

After synthesis, the tool generates one output netlist. The IP is re-encrypted and included in this netlist. The rest of the netlist is not encrypted and can be read. The IP is not included in any simulation netlists that are written out; it is treated as a black box.

See [Encryption and Decryption, on page 144](#) for a flow diagram of this process.

## Working with Xilinx IP

You can incorporate Xilinx IP into your design in different ways, depending on the format of the IP. For further information about secure and non-secure edn, ngc, and ngo files, see the following:

- [Xilinx IP Cores](#), on page 195
- [Including Xilinx Cores for Logic and Physical Synthesis](#), on page 196

For information about including EDK cores or IP produced with encryptIP, see [Working with EDK Cores, on page 204](#) and [Evaluating Vendor IP, on page 158](#), respectively.

### Xilinx IP Cores

The following table describes the Xilinx IP cores:

|                  |                                                                                                                                                                                                                                                                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EDN              | The tool can read the contents of an EDN core. This means that it can absorb and optimize the contents, and place them along with the rest of the design during physical synthesis. The tool includes the core contents in the synthesized EDIF, and forward-annotates any placement constraints in the accompanying ncf file.                       |
| NGO              | The tool can read the contents of the NGO core. This means that it can absorb and optimize the contents, and place them along with the rest of the design during physical synthesis. The tool includes the core contents in the synthesized EDIF, and forward-annotates any placement constraints in the accompanying ncf file.                      |
| NGC, non-secured | The tool can read the core contents, and perform limited optimizations of the core, like constant propagation. For physical synthesis, it can also place the contents along with the rest of the design. The tool annotates the core contents to the synthesized EDIF, and forward-annotates any placement constraints in the accompanying ncf file. |
| NGC, secured     | The tool can read the contents of secure NGC cores. The tool might perform limited optimizations of the core like constant propagation. The tool writes a separate encrypted EDIF netlist for each core.                                                                                                                                             |

The following table broadly summarizes how the synthesis tool treats various kinds of IP:

| <b>IP Format</b>             | <b>Synthesis Input</b>                                                  | <b>Synthesis Output Format</b>                     |
|------------------------------|-------------------------------------------------------------------------|----------------------------------------------------|
| EDN, Non-secure NGC, NGO     | Add file                                                                | Plain text                                         |
| Secure NGC                   | Add file                                                                | Encrypted EDIF                                     |
| Encrypted EDK                | Add IP with Import IP->Import Xilinx EDK Project                        | Black boxes or white boxes                         |
| Encrypted RTL from encryptIP | Download with Import IP->Download IP from Synopsys, unzip, and add file | Black box or plain text, as determined by IP owner |

## Including Xilinx Cores for Logic and Physical Synthesis

The following procedure shows you how to include Xilinx secure and non-secure cores in your project for logic synthesis or graph-based physical synthesis.

The procedure itself is the same for both secure and non-secure cores, but the implementation details are different, because the two kinds of cores are treated differently.

1. Instantiate the Xilinx core in your top-level RTL.
2. Make sure you are targeting a technology that supports this design flow.
3. Add the `edn`, `ngo`, and/or `ngc` core files directly to your project. The synthesis tools can read these formats and incorporate the cores into the design.



| IP Core File                   | IP Visibility                               | Effect of Synthesis Optimizations                                                                                                                                                                                                                            |
|--------------------------------|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EDN<br>NGO<br>NGC, non-secured | Core contents visible to the synthesis tool | The tool can do some optimizations on the core, like constant propagation. For physical synthesis, the contents can be placed along with the rest of design.                                                                                                 |
| NGC, secured                   | Core contents visible to the synthesis tool | The tool cannot optimize or place cores or absorb them into the netlist. During synthesis, the secure core is treated as a <i>white box</i> (a model that includes the port interface and timing information only), and all optimizations are based on this. |

- Set the `syn_macro` attribute to determine how you want the cores to be treated during synthesis.

| To...                                                                                                                                              | Set...                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| Optimize the core and write it out to the netlist                                                                                                  | <code>syn_macro=0</code>                  |
| Prevent optimizations, and write out the core to the netlist as a white box (a model that includes the port interface and timing information only) | <code>syn_macro=1</code><br>(on the view) |

- Synthesize the design.
  - Optionally, set the option to automatically run P&R from the synthesis tool interface after synthesis is complete. If you chose to run P&R from the synthesis tool, the output netlist and constraint files are automatically copied to the P&R directory.
  - Set any other options.
  - Click the Run button to run synthesis.

The synthesis tools read the timing and resource usage information from the core files. For physical synthesis, the tool runs logic synthesis and places the design at the same time. The synthesis process treats the cores as follows:

|                                                       |                                                                                                                                                                      |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Non-secure core with no black box attributes attached | The synthesis tools might perform limited optimizations like constant propagation as needed. You can view the internals of the core in the RTL and Technology views. |
| Non-secure core marked as a white box                 | The synthesis tool does not modify the core or write out the internals of the core in the synthesized netlist.                                                       |
| Secure cores                                          | The tool might perform limited optimizations, like constant propagation. You can view the internals of the core in the Technology view.                              |

After synthesis, the tool generates core output files for P&R:

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EDN<br>NGC, non-secured<br>NGO | <ul style="list-style-type: none"> <li>• The tool generates one main output netlist that includes all the unencrypted cores. The log file resource usage report includes the resources used by the cores.</li> <li>• All timing constraints are forward-annotated in the <code>synplicity.ucf</code> file. This file includes imported UCF constraints (see <a href="#">Converting and Using Xilinx UCF Constraints, on page 255</a>) as well as user-specified timing constraints.</li> <li>• Placement constraints are forward-annotated in the <code>&lt;design&gt;.ncf</code> file.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                         |
| NGC, secured                   | <ul style="list-style-type: none"> <li>• The tool writes out a top-level EDIF file which references individual EDIFs for each instantiation of a secure core. These files are not included in the main netlist. The tool suffixes the original core name with <code>_syn</code> when it names the lower-level files. The log file report of resource usage includes the resources used by the cores.</li> <li>• The synthesis tool puts all timing constraints into one <code>synplicity.ucf</code> file for the P&amp;R tool. This file includes imported UCF constraints (see <a href="#">Converting and Using Xilinx UCF Constraints, on page 255</a>) as well as user-specified timing constraints.</li> <li>• Placement constraints, excluding constraints for secure cores, are forward-annotated in a <code>&lt;design&gt;.ncf</code> file.</li> <li>• For each secure core, the tool generates an individual <code>.ncf</code> file with the constraints for that core.</li> </ul> |

## Including Xilinx EDK Cores

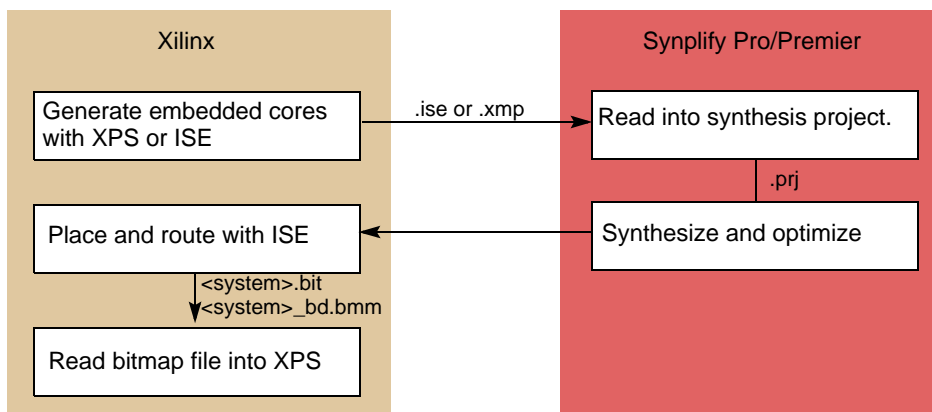
The Xilinx Embedded Development Kit (EDK) allows you to implement embedded designs using IBM PowerPC™ hard processor cores, Xilinx MicroBlaze™ soft processor cores, and Xilinx-supplied implementations of buses, block RAMs, and peripherals such as USB and PCIs. You can use EDK-generated embedded processor subsystems in a Synplify Pro or Synplify Premier synthesis flow, which treats the entire design as one entity. This allows you to optimize and debug across the entire design, including the embedded core. The heart of this functionality is the `edk2syn` utility, which is described in *The EDK2Syn Utility*, on page 622 in the *Reference Manual*.

This section describes how to implement the flow, including the following:

- [The Synplify-EDK Design Flow](#), on page 199
- [Xilinx Hardware Development Flows](#), on page 207
- [Working with EDK Cores](#), on page 204

### The Synplify-EDK Design Flow

The following figure summarizes the process used to incorporate Xilinx embedded cores into a single FPGA design. The steps that follow go into more detail.



1. Create your embedded design, using one of the following Xilinx methodologies:
  - Use ISE and the Xilinx EDK-ISE hardware development flow to generate an ise file. This is the recommended flow for creating processor-based subsystems. See [Xilinx EDK-ISE Hardware Development Flow, on page 208](#) for details.
  - Use EDK and the Xilinx standalone EDK hardware flow to generate an xmp file. See [Xilinx Standalone EDK Hardware Development Flow, on page 209](#).
2. Specify whether the embedded design is a top-level module or a submodule, as described in [Specifying a Subsystem as a Top-Level Module, on page 204](#) and [Specifying a Subsystem as a Submodule, on page 205](#).
3. Create the EDK synthesis project and generate core netlists.
  - Select Project->Project Options->Hierarchy and Flow. Select Implement Design in ISE (Export to Project Navigator Flow: DEPRECATED).
  - Run the project using Hardware->Generate Netlist.

The tool runs XST on all the cores one by one to generate the synthesized NGC netlists. Synopsys synthesis tools do not use XST-generated NGC files if an IP core is not encrypted. The XST-generated NGC files are only needed for encrypted cores like Microblaze.

Depending on the flow you used, you now have an ise file (Xilinx EDK-ISE hardware flow) or xmp file (Xilinx standalone EDK hardware flow).
4. Import the ISE/EDK project into the Synplify Pro or Premier tool.

See [Setting up the EDK Synthesis Project, on page 201](#) for details.
5. Synthesize, place, and route the design. See [Synthesizing EDK Cores, on page 203](#) for details.
6. Import the bitmap file back into the XPS environment.
  - If you ran P&R from the synthesis tool GUI, copy the `<design>.bit` and `<design>_bd.bmm` files from the `synplify/rev_1/par_1` directory to the implementation directory in the EDK project.
  - Go back to the XPS GUI and select Project->Import from ProjNav.
  - Select the .bit and .bmm files.

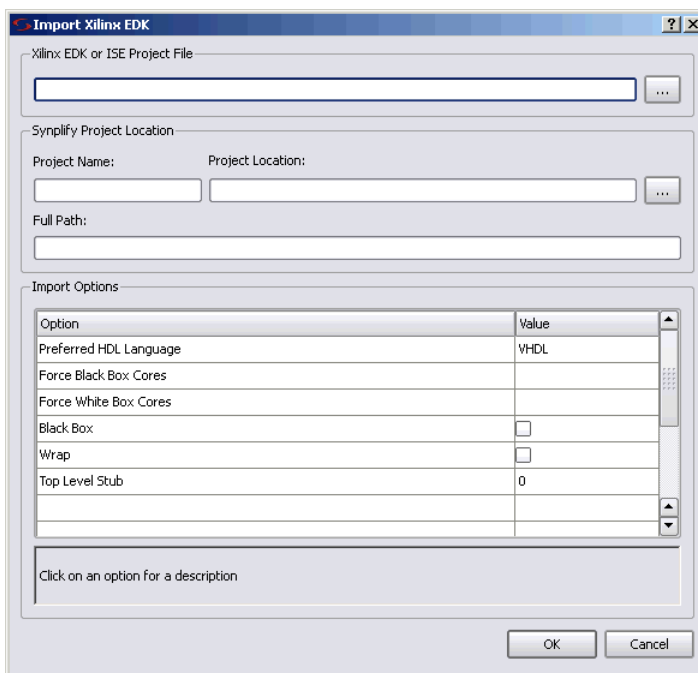
- Click OK.
- Run Device Configuration->Update Bitstream.

The bitstream can now be downloaded into the FPGA.

## Setting up the EDK Synthesis Project

The following procedure provides the details on setting up an EDK project for synthesis.

1. Start the Synplify Pro/Premier software, and select Import IP->Import Xilinx EDK Project. This opens the Import Xilinx EDK dialog box.



2. Do the following in the Import Xilinx EDK dialog box to import the EDK project you created with the Xilinx tools (described in the first steps of the procedure in [The Synplify-EDK Design Flow, on page 199](#)).
  - Go to Xilinx EDK or ISE Project File and specify the EDK file (.xmp) or ISE file (.ise) you want to import. For example:

```
E:/basic_ppc/top.ise
E:/basic_mb/system.xmp
```

- Specify a location for the synthesis project to be created in Project Location, and a name for the project in Project Name.
- Define the black boxes and white boxes in your design with the Black Box, Force Black Box Cores, and Force White Box Cores options, as described in [Specifying EDK Cores as Black Boxes and White Boxes, on page 206](#). If you are using an encrypted core, make sure to specify how you want to handle it. See [Dealing with Encrypted EDK Cores, on page 207](#) for details.

The tool will not complete synthesis if it finds inadequately defined components, and you will have to iterate through synthesis again.

- Set the options you want in the Import Options section.
- Click OK.

Synplify uses underlying edk2syn functionality (see [The EDK2Syn Utility, on page 622](#) in the *Reference Manual*) to read the following Xilinx files and include the information from them into the synthesis project.

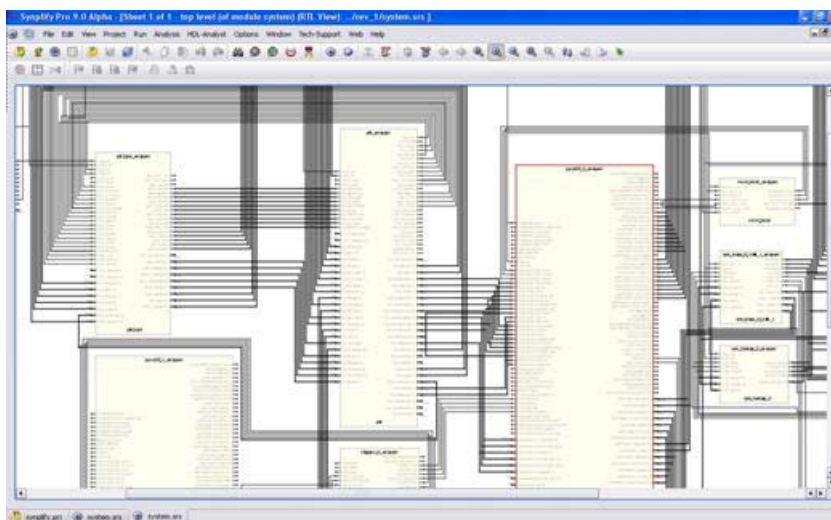
|                                                    |                                                                                                                                                           |
|----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| .ise                                               | Project file. In the ISE project directory, if created                                                                                                    |
| .xpm and .mhs                                      | Project file. In the EDK project directory, if created                                                                                                    |
| .mpd, .pao,<br>HDL (.v, .vhd),<br>.edn, .ngo, .ngc | Design specification files. In EDK project directory, if available, or in the EDK hardware library at the location specified in the EDK installation path |

It generates a log file called `edk2syn.log`.

3. If you used the Xilinx standalone EDK hardware development flow to create the system, add any custom files manually to the Synplify project.

If you used the Xilinx EDK-ISE hardware development flow, you do not need to do this, because any custom logic is included in the `.ise` project file and automatically read by the synthesis tools.

The cores are now part of the Synplify design. The following figure shows an RTL view of a design that includes EDK cores.



4. To include an EDK core as a subsystem in a larger design, create the subsystem as described in the previous steps and instantiate the subsystem in the top-level HDL.

## Synthesizing EDK Cores

After you have set up your synthesis project as described in [Setting up the EDK Synthesis Project, on page 201](#), you can synthesize your design. You can also run Xilinx placement and routing from the synthesis tool interface.

1. If you have ucf constraints, convert them to the sdc format used by the synthesis tools. See [Converting and Using Xilinx UCF Constraints, on page 255](#) for more information about using this utility.
2. Set implementation options for synthesizing the EDK cores:
  - Click Impl Options. Go to the Place and Route tab and make sure that P&R Job is enabled. This option is on by default. When it is enabled, the software automatically runs Xilinx P&R on the design after synthesis.
  - If you are using Synplify Premier, enable the Physical Synthesis option to take advantage of physical synthesis optimizations.

3. Set other options and constraints as usual.
4. Click Run to synthesize, place, and route the embedded hardware system. The synthesis tool runs the Xilinx place and route tool after synthesizing the design.

After the run, the Synplify and P&R tools generate the following files:

|                                |                                                                                                                                  |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| .prj                           | Synthesis project file                                                                                                           |
| .sdc                           | Synthesis constraint file                                                                                                        |
| .ucf                           | Post-synthesis timing constraint file generated for Xilinx P&R                                                                   |
| OPT file and core wrapper file | Xilinx Xflow OPT file and core wrapper file, generated after synthesis for any cores that are black boxes for the synthesis tool |
| .bmm                           | Bitmap file, generated after place and route and located in the Synplify place and route (par) directory                         |
| .bit                           | Bitmap file, generated after place and route and located in the Synplify place and route (par) directory                         |

If you use a Tcl script, do not save it from the synthesis tool interface. Doing so will result in hard-coded paths, and the Tcl file will not be portable. Save the Tcl file manually from outside the synthesis tool.

## Working with EDK Cores

The following describe how to handle some issues with EDK cores:

- [Specifying a Subsystem as a Top-Level Module](#), on page 204
- [Specifying a Subsystem as a Submodule](#), on page 205
- [Specifying EDK Cores as Black Boxes and White Boxes](#), on page 206
- [Dealing with Encrypted EDK Cores](#), on page 207

### Specifying a Subsystem as a Top-Level Module

The following procedure shows how to use the EDK-ISE flow to integrate Xilinx processor subsystems as top-level modules.

1. Start ISE, and in a new or existing project, do one of the following:



- If you have an existing processor design XMP file, add it with Add Source.
  - If you want to create a new processor design, use New Source, selecting Embedded Processor as a source type.
2. In the XPS GUI, select Hardware->Generate Netlist and generate the netlist.
  3. Go back to ISE and do the following:
    - Run the View HDL Instantiation Template process in the Processes window to open the template in the Project Navigator editor pane.
    - Copy the component declaration for the embedded system from the template and paste it into your top-level design architecture.
    - Copy the instantiation sample of the embedded system from the template into your top-level design and provide net name connections as necessary.
    - Add a User Constraint File (UCF) from the XPS project data directory.
    - Implement the design.

## Specifying a Subsystem as a Submodule

The following procedure shows how to use the EDK-ISE flow to integrate Xilinx processor subsystems as submodules. You can use either a top-down or bottom-up approach.

1. To use the top-down approach, do the following:
  - Start the ISE software and create a top-level project.
  - Create a new embedded processor source to include in the top-level design. This automatically starts XPS.
  - Develop your embedded submodule in XPS.
  - Generate a netlist in XPS.
  - Return to the Project Navigator and synthesize your top-level design in ISE. You can then instantiate and connect the embedded subsystem to your top-level FPGA design.
2. To use the bottom-up approach, do the following:
  - Start XPS and develop your embedded processor design as a submodule.

- Generate a netlist in XPS.
- Start the ISE software and add the embedded submodule as a source to include in your top-level ISE software project.

## Specifying EDK Cores as Black Boxes and White Boxes

Accurately defining EDK cores as black boxes and white boxes is very important, because the synthesis run will fail if the design is not correctly specified, and you will have to iterate through another run. A black box is a component that does not have any definitions. A white box for the EDK flow has a core definition in either .edn or .ngc format.

1. In the Synplify Pro/Premier software, and select Import IP->Import Xilinx EDK Project. This opens the Import Xilinx EDK dialog box.
2. If the core does not have any definitions, as with the encrypted cores ([Dealing with Encrypted EDK Cores, on page 207](#)), enable the Black Box option.

Enabling this option causes the core wrapper file to be copied to the project folder and the black box attribute to be added for the core, so that the core is treated as a black box during synthesis.

3. If you want to treat a core as a black box during synthesis, enable the Force Black Box option.

When this option is enabled, the software searches for the core in the mhs file, copies the core wrapper file to the Synplify folder and edits it to add the black box attribute. The core is treated as a black box during synthesis. If it does not find the named core, it issues a warning message.

4. If you want to treat a core as a white box during synthesis, enable the Force White Box option.

The software searches for the core in the mhs file, and for a definition file named implementation/<core>.ngc. It adds this file to the project. The core is treated as a white box during synthesis. If it does not find the named core or the .ngc file, it issues a warning message.

## Dealing with Encrypted EDK Cores

Sometimes, as with MicroBlaze cores, a core is encrypted. In this case, you have a choice on how to handle the core.

- Define the encrypted core as a black box

To treat the core as a black box, make sure to enable the Black Box option for the core when you are setting options in the Import Xilinx EDK dialog box (step 3 in the process described in [The Synplify-EDK Design Flow, on page 199](#)). This signifies that the core is a black box for Synplify, and the tool reads the PAO file and the pointer to the encrypted definition file. It uses the information from the core wrapper file. The core is not optimized during synthesis. After synthesis, the tool generates core wrapper files for the black boxes.

- Do not define the encrypted core as a black box

If you have a netlist for your core, disable the Black Box option for the core when you are setting options in the Import Xilinx EDK dialog box (step 3 in the process described in [The Synplify-EDK Design Flow, on page 199](#)). This causes the Synplify tool to look for a definition file named `implementation/<core>.ngc`, which it adds to the synthesis project. If it does not find the file, you get an error message. These cores are freely optimized during synthesis, and are not encrypted in the Synplify output.

- Define the encrypted core as a white box, using the secure ngc flow

Beginning with Synplify Pro and Synplify Premier 9.0.2, you can use the secure ngc flow to include encrypted cores as white boxes in your synthesis design. A white box is a model that includes the port interface and timing information only. See [Including Xilinx Cores for Logic and Physical Synthesis, on page 196](#) for details about this methodology.

## Xilinx Hardware Development Flows

There are two Xilinx hardware flows that you can use to generate embedded hardware in HDL format:

- The Xilinx EDK-ISE Hardware Development Flow, using Xilinx ISE. This is the recommended flow. See [Xilinx EDK-ISE Hardware Development Flow, on page 208](#) for details.

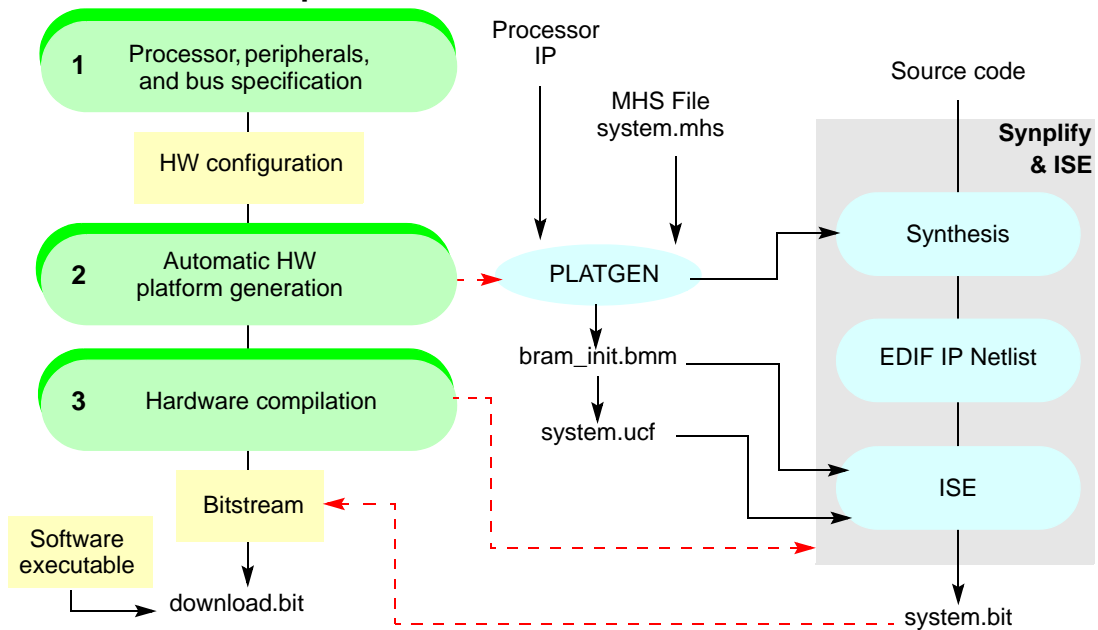
- The Xilinx Standalone EDK Hardware Development Flow, using Xilinx Platform Studio (XPS). See [Xilinx Standalone EDK Hardware Development Flow](#), on page 209 for details.

## Xilinx EDK-ISE Hardware Development Flow

The EDK-ISE flow is the recommended flow for creating processor-based embedded sub-systems. The biggest advantage of this flow is that it supports the addition of non-peripheral custom logic to the system. Unlike the EDK-XFLOW, this flow lets you add HDL files that contain non-peripheral custom logic directly to the project.

The following figure shows the entire Xilinx hardware flow, with the hardware compilation phase expanded to show the component synthesis (Synplify) and P&R (ISE) steps. In this flow the EDK platform generator (Platgen) writes out the hardware platform. During the hardware compilation phase, synthesis and place-and-route tools are used to generate a hardware bitstream, which is then merged with the software executable to generate the final download.bit file.

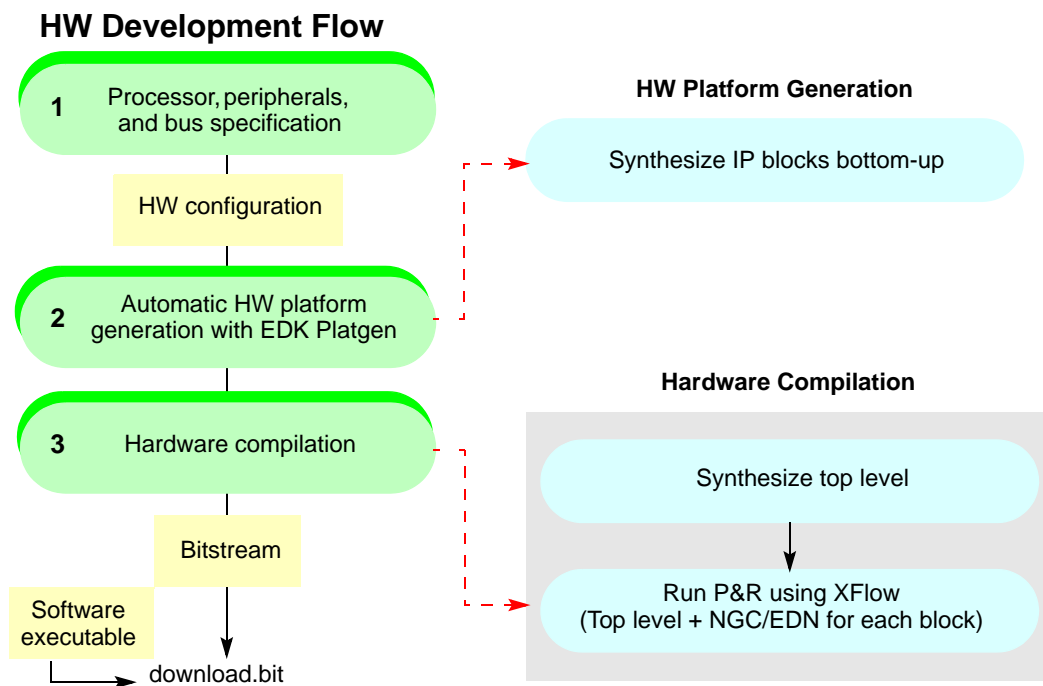
### Xilinx HW Development Flow



You can use the EDK-ISE flow to integrate processor subsystems as top-level modules or submodules, as described in [Specifying a Subsystem as a Top-Level Module, on page 204](#) and [Specifying a Subsystem as a Submodule, on page 205](#).

## Xilinx Standalone EDK Hardware Development Flow

Use the following flow to generate PowerPC- or MicroBlaze-based embedded processor systems or submodules by directly invoking Xilinx Platform Studio (XPS), instead of invoking it through the Integrated System Environment (ISE)<sup>™</sup>. XPS provides an integrated environment for creating the software and hardware specification flows for an Embedded Processor system.



The following figure shows the entire Xilinx hardware flow, with the hardware compilation phase expanded to show the component synthesis (Synplify) and P&R (ISE) steps. In this flow, EDK Platform Generator (Platgen) writes out the HDL hardware platform in a specific format. The EDK-XFLOW treats each hardware component as an independent core, and generates a separate XST project file for each core. During synthesis, the process runs XST on all the cores one by one, to generate synthesized netlists (NGC files).

XFLOW is a built-in feature that runs the ISE tools in the background through a command-line script file. It runs system top-level hardware synthesis, mapping, and P&R processes in the background so that you do not have to work with the Xilinx Integrated System Environment (ISE) to generate the configuration bitstream file. This flow also compiles and executes multiple software applications, generates libraries for the code, and merges software with hardware files for downloading to a target board.

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 5

# Specifying Constraints

---

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

- [Using the SCOPE UI](#), on page 212
- [Specifying Timing Constraints](#), on page 219
- [Specifying Timing Exceptions](#), on page 230
- [Using Collections](#), on page 237
- [Using Auto Constraints](#), on page 251
- [Translating Altera QSF Constraints](#), on page 253
- [Converting and Using Xilinx UCF Constraints](#), on page 255

For additional information about working with constraints, see [Working with Constraint Files](#), on page 98.

For information about specifying attributes and directives, and setting project options, see [Setting up a Logic Synthesis Project](#), on page 269.


## Using the SCOPE UI

You can use a text editor to create a constraint file as described in [Working with Constraint Files, on page 98](#), but it is easier to use the SCOPE (Synthesis Constraint Optimization Environment) window, which provides a spreadsheet-like interface for entering constraints. The SCOPE interface 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. If you want to use a text editor to edit a constraint file, close the SCOPE window before editing the file, or you will overwrite results.

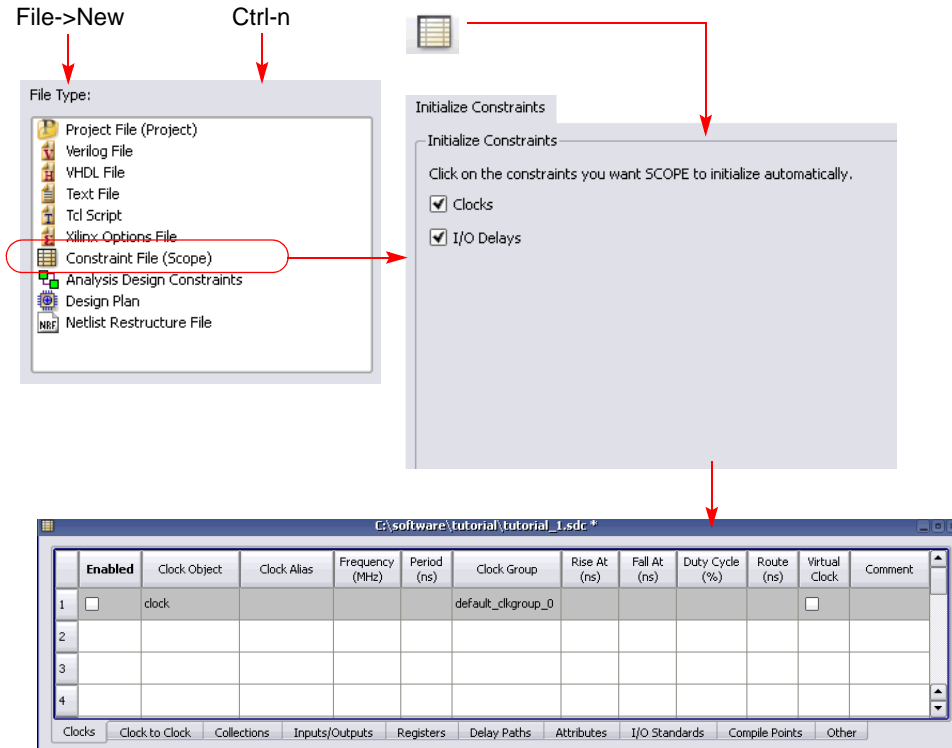
You can also use the SCOPE window to add attributes, define collections and specify constraints for them. For details, see [Specifying Attributes Using the SCOPE Editor, on page 308](#) and [Creating and Using Collections \(SCOPE Window\), on page 238](#).

## Creating a Constraint File Using the SCOPE Window

The following procedure shows you how to open the SCOPE window to generate constraint files. For details about generating constraint files for compile point modules (in the Synplify Premier and Synplify Pro tools), see [Logical Compile-Point Synthesis, on page 560](#).

1. To create a new constraint file, follow these steps:
  - Compile the design (F7). If you do not compile the design, you can still use the SCOPE window, but the software does not automatically initialize the clocks and I/O ports. You have to type in entries manually because the software has no knowledge of the design.
  - Open the SCOPE window by clicking the SCOPE icon in the toolbar () , pressing Ctrl-n, or selecting File -> New. If you use one of the latter two methods, select Constraint File (SCOPE) as the type of file to open. This opens the Initialize New Constraint File dialog box.





- Optionally, select the constraints to be initialized and click OK. If you started with a compiled design, setting these options automatically initializes the Clock and Inputs/Outputs tabs with the appropriate signals.

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.

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

The SCOPE window opens with the file you specified. For details about editing the file, see [Entering and Editing Constraints in the SCOPE](#)

[Window, on page 214](#). If you want to edit the Tcl file directly, see [Working with Constraint Files, on page 98](#).

## Entering and Editing Constraints in the SCOPE Window

Enter constraints directly in the SCOPE window. You can use the Initialize Constraint panel to enter default constraints, and then use the direct method to modify, add, or delete constraints.

The Synplify Pro tool also lets you add constraints automatically. For information about auto constraints, see [Using Auto Constraints, on page 251](#).

1. Click the appropriate tab at the bottom of the window to enter the kind of constraint you want to create:

| To define...                                                                                                                                                                            | Click...           |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| Clock frequency for a clock signal output of clock divider logic<br>A specific clock frequency that overrides the global frequency                                                      | Clocks             |
| Edge-to-edge clock delay that overrides the automatically calculated delay.                                                                                                             | Clock to Clock     |
| Constraints for a group of objects you have defined as a collection with the Tcl command. For details, see <a href="#">Creating and Using Collections (SCOPE Window), on page 238</a> . | Collections        |
| Input/output delays that model your FPGA input/output interface with the outside environment                                                                                            | Inputs/<br>Outputs |
| Delay constraints for paths feeding into/out of registers                                                                                                                               | Registers          |
| Paths that require multiple clock cycles                                                                                                                                                | Delay Paths        |
| Paths to ignore for timing analysis (false paths)                                                                                                                                       | Delay Paths        |
| Maximum delay for paths                                                                                                                                                                 | Delay Paths        |
| Attributes, like <code>syn_reference_clock</code> , that were not entered in the source files                                                                                           | Attributes         |

| To define...                                                                                                                                                                                                   | Click...       |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| I/O standards for certain technologies of the Actel, Altera, and Xilinx devices for any port in the I/O Standard panel of the SCOPE window.                                                                    | I/O Standard   |
| Compile points in a top-level constraint file. See <a href="#">Using Compile-point Synthesis, on page 573</a> for more information about compile points. (The Synplify tool does not support this flow.)       | Compile Points |
| Place and route tool constraints<br>Other constraints not used for synthesis, but which are passed to other tools. For example, multiple clock cycles from a register or input pin to a register or output pin | Other          |

The SCOPE window displays columns appropriate to the kind of constraint you picked. You can now enter constraints using the wizard, or work directly in the SCOPE window.

2. 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...               |
|------------------------|----------------------------------|
| <i>v:design_name</i>   | hierarchies or “views” (modules) |
| <i>c:clock_name</i>    | clocks                           |
| <i>i:instance_name</i> | instances (blocks)               |
| <i>p:port_name</i>     | ports (off-chip)                 |

| Prefix Identifiers | Description for...                                 |
|--------------------|----------------------------------------------------|
| 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, Frequency or 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 Timing Constraints, on page 219](#).

3. 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.                      |

4. Save the file by clicking the Save icon and naming the file.

The software creates a TCL constraint file (.sdc). See [Working with Constraint Files, on page 98](#) for information about the commands in this file.

5. To apply the constraints to your design, you must add the file to the project now or later.
  - Add it immediately by clicking Yes in the prompt box that opens after you save the constraint file.
  - Add it later, following the procedure for adding a file described in [Making Changes to a Project, on page 274](#).

## Setting SCOPE Display Preferences

You can set format and colors in the SCOPE window. The following table lists some preferences and shows you how to set them.

| To...                                                      | Do this...                                                                                                                                                                                                                                                                                                                     |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Set the appearance of lines and buttons in the SCOPE table | <p>With a SCOPE window open, select View-&gt; Properties.</p> <p>Set the options you want on the Display Settings form.</p> <p>Check the Save settings to profile option if you want to settings to be the default.</p>                                                                                                        |
| Set fonts, colors, and borders for a row                   | <p>Select a SCOPE row.</p> <p>Select Format -&gt; Style.</p> <p>On the Styles form, check Save as Default if you want the new settings to be the default.</p> <p>Select the category you want to change (Row Header or Standard), and click Change.</p> <p>Set the display options you want and click OK on both forms.</p>    |
| Set fonts, colors, and borders for a column                | <p>Select a SCOPE row.</p> <p>Select Format -&gt; Style.</p> <p>On the Styles form, check Save as Default if you want the new settings to be the default.</p> <p>Select the category you want to change (Column Header or Standard), and click Change.</p> <p>Set the display options you want and click OK on both forms.</p> |
| Set fonts, colors, and borders for a single cell           | <p>Select a SCOPE cell.</p> <p>Select Format -&gt; Cells.</p> <p>Set the display options you want and click OK.</p>                                                                                                                                                                                                            |

---

| <b>To...</b>                   | <b>Do this...</b>                                                                                                               |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| Align text in columns and rows | Select a column or row in the SCOPE window.<br>Select Format -> Align.<br>Click the alignment you want and click OK.            |
| Size columns/rows to text      | Select a column or row in the SCOPE window.<br>Select Format -> Resize Rows or Format -> Resize Columns.                        |
| Hide/show cells                | Select a SCOPE cell.<br>Select Format -> Cover Cells to hide a cell.<br>Select Format -> Remove Covering to show a hidden cell. |

---

## Specifying Timing Constraints

You can define timing constraints in the SCOPE interface, which automatically generates a Tcl constraints file, or manually with a text editor, as described in [Using a Text Editor for Constraint Files](#), on page 100.

The SCOPE interface is much easier to use, and you can define various timing constraints in it. For the equivalent Tcl syntax, see [Chapter 14, Tcl Commands and Scripts](#) in the *Reference Manual*. See the following for different timing constraints:

- [Entering Default Constraints](#), on page 219
- [Setting Clock and Path Constraints](#), on page 220
- [Defining Clocks](#), on page 222
- [Defining Input and Output Constraints](#), on page 226
- [Specifying Standard I/O Pad Types](#), on page 228
- [Specifying Xilinx Timing Constraints](#), on page 228
- [Using -route for Physical Synthesis in Xilinx Designs](#), on page 230

To set constraints for timing exceptions like false paths and multicycle paths, see [Specifying Timing Exceptions](#), on page 230.

For information about physical constraints, see [Setting Constraints for Physical Synthesis](#), on page 347

## Entering Default Constraints

To edit or set individual constraints, or to create constraints in the Other tab, work directly in the SCOPE window ([Setting Clock and Path Constraints](#), on page 220). For auto constraints in Synplify Pro, see [Using Auto Constraints](#), on page 251. To apply the constraints, add the file to the project according to the procedure described in [Making Changes to a Project](#), on page 274. The constraints file has an .sdc extension. See [Working with Constraint Files](#), on page 98 for more information about constraint files.

## Setting Clock and Path Constraints

The following table summarizes how to set different clock and path constraints from the SCOPE window. For information about setting compile point constraints or attributes, see [Logical Compile-Point Synthesis, on page 560](#) for more information about compile points and [Specifying Attributes Using the SCOPE Editor, on page 308](#). For information about setting default constraints, see [Entering Default Constraints, on page 219](#).

| To define...             | Pane                                     | Do this to set the constraint...                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Clocks                   | Clock                                    | Select the clock object (Clock).<br>Specify a clock name (Clock Alias), if required.<br>Type a frequency value (Frequency) or a period (Period).<br>Change the default Duty Cycle or set Rise/Fall At, if needed.<br>Change the default clock group, if needed<br>Check the Enabled box.<br>See <a href="#">Defining Clocks, on page 222</a> for information about clock attributes. |
| Virtual clocks           | Clock                                    | Set the clock constraints as described for clocks, above.<br>Check the Virtual Clock box.                                                                                                                                                                                                                                                                                            |
| Route delay              | Clock<br>Inputs/<br>Outputs<br>Registers | Specify the route delay in nanoseconds. Refer to <a href="#">Defining Clocks, on page 222</a> , <a href="#">Defining Input and Output Constraints, on page 226</a> and the Register Delays section of this table details.                                                                                                                                                            |
| Edge-to-edge clock delay | Clock to<br>Clock                        | Select the starting edge for the delay constraint (From Clock Edge).<br>Select the ending edge for the constraint (To Clock Edge).<br>Enter a delay value.<br>Mark the Enabled check box.                                                                                                                                                                                            |
| Input/output delays      | Inputs/<br>Outputs                       | See <a href="#">Defining Input and Output Constraints, on page 226</a> for information about setting I/O constraints.                                                                                                                                                                                                                                                                |



| To define...       | Pane                             | Do this to set the constraint...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Register delays    | Registers                        | <p>Select the register (Register).<br/>           Select the type of delay, input or output (Type).<br/>           Type a delay value (Value).<br/>           Check the Enabled box.</p> <p>If you do not meet timing goals after place-and-route, adjust the clock constraint as follows:</p> <ul style="list-style-type: none"> <li>• In the Route column for the constraint, 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 that input register.</li> <li>• Resynthesize your design.</li> </ul> |
| Maximum path delay | Delay Path                       | <p>Select the Delay Type path of Max Delay.<br/>           Select the port or register (From/Through). See <a href="#">Defining From/To/Through Points for Timing Exceptions, on page 231</a> for more information.<br/>           Select another port or register if needed (To/Through).<br/>           Set the delay value (Max Delay).<br/>           Check the Enabled box.</p>                                                                                                                                                                                                                                    |
| Multi-cycle paths  | Delay Paths                      | See <a href="#">Defining Multi-cycle Paths, on page 234</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| False paths        | Delay Paths<br>Clock to<br>Clock | See <a href="#">Defining False Paths, on page 235</a> for details.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Global attributes  | Attributes                       | <p>Set Object Type to &lt;global&gt;.<br/>           Select the object (Object).<br/>           Set the attribute (Attribute) and its value (Value).<br/>           Check the Enabled box.</p>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Attributes         | Attributes                       | <p>Do either of the following:</p> <ul style="list-style-type: none"> <li>• Select the type of object (Object Type).<br/>           Select the object (Object).<br/>           Set the attribute (Attribute) and its value (Value).<br/>           Check the Enabled box.</li> <li>• Set the attribute (Attribute) and its value (Value).<br/>           Select the object (Object).<br/>           Check the Enabled box.</li> </ul>                                                                                                                                                                                   |
| Other              | Other                            | <p>Type the TCL command for the constraint (Command).<br/>           Enter the arguments for the command (Arguments).<br/>           Check the Enabled box.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

## Defining Clocks

Clock frequency is the most important timing constraint, and must be set accurately. If you are planning to auto constrain your design ([Using Auto Constraints, on page 251](#)), do not define any clocks. The following procedures show you how to define clock frequency ([Defining Clock Frequency, on page 222](#)) and set other clock constraints that affect timing, like clock groups ([Defining Other Clock Requirements, on page 225](#)).

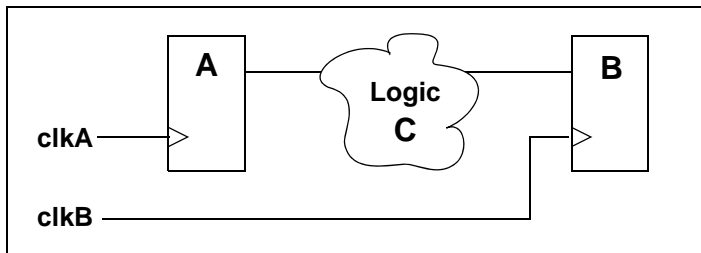
### Defining Clock Frequency

This section shows you how to define clock frequency either through the GUI or in a constraint file. See [Defining Other Clock Requirements, on page 225](#) for other clock constraints. If you want to use auto constraints (Synplify Pro only), do not define your clocks.

1. Define a realistic global frequency for the entire design, either in the Project view or the Constraints tab of the Implementation Options dialog box.

This target frequency applies to all clocks that do not have specified clock frequencies. If you do not specify any value, a default value of 1 MHz (or 1000 ns clock period) applies to all timing paths whenever the clock associated with both start and end points of the path is not specified. Each clock that uses the global frequency is assigned to its own clock group. See [Defining Other Clock Requirements, on page 225](#) for more information about clock group settings.

The global frequency also applies to any purely combinatorial paths. The following figure shows how the software determines constraints for specified and unspecified start or end clocks on a path:



| If clkA is... | And clkB is... | The effect for logic C is...                                                                                                                              |
|---------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Undefined     | Defined        | The path is unconstrained unless you specify that clkB be constrained to the inferred clock domain for clkA                                               |
| Defined       | Undefined      | The path is unconstrained unless you specify that clkA be constrained to the inferred clock domain for clkB.                                              |
| Defined       | Defined        | For related clocks in the same clock group, the relationship between clocks is calculated; all other paths between the clocks are treated as false paths. |
| Undefined     | Undefined      | The path is unconstrained.                                                                                                                                |

2. Define frequency for individual clocks on the Clocks tab of the SCOPE window (`define_clock` constraint).
  - Specify the frequency as either a frequency in the Frequency column (`-freq Tcl` option) or a time period in the Period column (`-period Tcl` option). When you enter a value in one column, the other is calculated automatically.
  - For asymmetrical clocks, specify values in the Rise At (`-rise`) and Fall At (`-fall`) columns. The software automatically calculates and fills out the Duty Cycle value.

The software infers all clocks, whether declared or undeclared, by tracing the clock pins of the flip-flops. However, it is recommended that you specify frequencies for all the clocks in your design. The defined frequency overrides the global frequency. Any undefined clocks default to the global frequency.

3. Define internal clock frequencies (clocks generated internally) on the SCOPE Clocks tab (`define_clock` constraint). Apply the constraint according to the source of the internal clock.

| Source                            | Add SCOPE constraint/define_clock to...                                                                                                                                                   |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Register                          | Register.                                                                                                                                                                                 |
| Instance, like a PLL or clock DLL | Instance. If the instance has more than one clock output, apply the clock constraints to each of the output nets, making sure to use the n: prefix (to signify a net) in the SCOPE table. |
| Combinatorial logic               | Net. Make sure to use the n: prefix in the SCOPE interface.                                                                                                                               |

- For signals other than clocks, define frequencies with the `syn_reference_clock` attribute. You can add this attribute on the SCOPE Attributes tab.

You might need to do this if your design uses an enable signal as a clocking signal because of limited clocking resources. If the enable is slower than the clock, defining the enable frequency separately instead slowing down the clock frequency ensures more accuracy. If you slow down the clock frequency, it affects all other registers driven by the clock, and can result in longer run times as the tool tries to optimize a non-critical path.

Define this attribute as follows:

- Define a dummy clock on the Clocks tab (`define_clock` constraint).
- Add the `syn_reference_clock` attribute (Attributes tab) to the affected registers to apply the clock. In the constraint file, you can use the Find command to find all registers enabled by a particular signal and then apply the attribute:

```
define_clock -virtual dummy -period 40.0
define_attribute {find -seq * -hier -filter @(enable == en40)}
 syn_reference_clock dummy
```

- For Altera PLLs and Xilinx DCMs and DLLs, define the clock at the primary inputs.
  - For Altera PLLs, you must define the input frequency, because the synthesis software does not use the input value you specified in the Mega wizard software. The synthesis tool assigns all the PLL outputs to the same clock group. It forward-annotates the PLL inputs.

- If needed, use the Xilinx properties directly to define the DCMs and DLLs. The synthesis software assigns defined DCMs and DLLs to the same clock group, because it considers these clocks to be related. It forward-annotates the DLL/DCM inputs. The following shows some examples of the properties you can specify

---

DLLs    Phase shift and frequency multiplication properties like `duty_cycle_correction` and `clkdv_divide`

---

DCMs    DCM properties like `clkfx_multiply` and `clkfx_divide`

---

6. After synthesis, check the Performance Summary section of the log file for a list of all the defined and inferred clocks in the design.
7. If you do not meet timing goals after place-and-route, adjust the clock constraint as follows:
  - Open the SCOPE window with the clock constraint.
  - In the Route column for the constraint, 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 all the input registers for that clock.
  - Resynthesize your design.

## Defining Other Clock Requirements

Besides clock frequency (described in [Defining Clock Frequency, on page 222](#)), you can also set other clock requirements, as follows:

- If you have limited clock resources, define clocks that do not need a clock buffer by attaching the `syn_noclockbuf` attribute to an individual port, or the entire module/architecture.
- Define the relationship between clocks by setting clock domains. By default, each clock is in a separate clock group named `default_clkgroup<n>` with a sequential number suffix.
  - On the SCOPE Clocks tab, group related clocks by putting them into the same clock group. Use the Clock Group field to assign all related clocks to the same clock group.
  - Make sure that unrelated clocks are in different clock groups. If you do not, the software calculates timing paths between unrelated clocks in the same clock group, instead of treating them as false paths.

- Input and output ports that belong to the System clock domain are considered a part of every clock group and will be timed. See [Defining Input and Output Constraints, on page 226](#) for more information.

The software does not check design rules, so it is best to define the relationship between clocks as completely as possible.

- Define all gated clocks with the `define_clock` constraint.

Avoid using gated clocks to eliminate clock skew. If possible, move the logic to the data pin instead of using gated clocks. If you do use gated clocks, you must define them explicitly, because the software does not propagate the frequency of clock ports to gated clocks.

To define a gated clock, attach the `define_clock` constraint to the clock source, as described above for internal clocks. To attach the constraint to a keepbuf (a keepbuf is a placeholder instance for clocks generated from combinatorial logic), do the following:

- Attach the `syn_keep` attribute to the gated clock to ensure that it retains the same name through changes to the RTL code.
- Attach the `define_clock` constraint to the keepbuf generated for the gated clock.
- Specify edge-to-edge clock delays on the Clock to Clock tab (`define_clock_delay`).
- After synthesis, check the Performance Summary section of the log file for a list of all the defined and inferred clocks in the design.

## 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 220](#), 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.
  - In the Route column for the input constraint, 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.
  - Resynthesize your design.

## Specifying Standard I/O Pad Types

For certain Actel, Altera, and Xilinx technologies, you can specify a standard I/O pad type to use in the design. The equivalent Tcl command is `define_io_standard`.

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 for different vendors, refer to the relevant section in the *Reference Manual*: [Actel I/O Standards, on page 417](#), [Altera I/O Standards, on page 418](#), and [Xilinx I/O Standards, on page 427](#).

4. Where applicable, set other parameters like drive strength, slew rate, and termination.

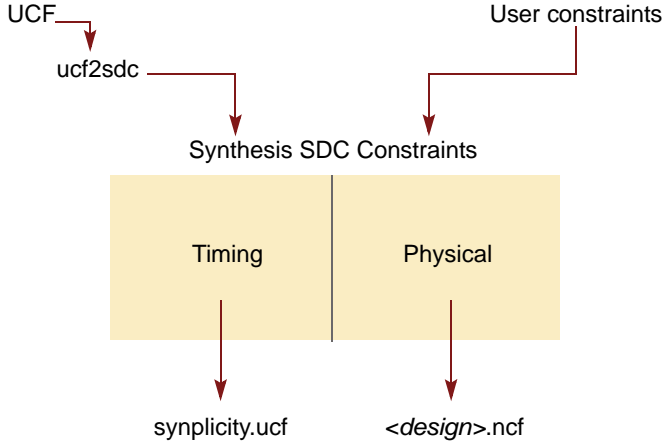
You cannot set these parameter values for industry I/O standards whose parameters are defined by the standard.

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.

## Specifying Xilinx Timing Constraints

For Xilinx designs, you can import Xilinx constraints from a `ucf` file in addition to specifying constraints within the synthesis tool. In the output files, the synthesis tool separates the timing constraints from the physical constraints. Timing constraints are written to the `synplicity.ucf` file and physical constraints to the `<design>.ncf` file, as shown in this figure:





1. To specify user constraints, use the SCOPE interface.

See [Entering and Editing Constraints in the SCOPE Window](#), on page 214 for details on how to specify constraints.

2. To use constraints from a Xilinx ucf file, use the procedures described in [Converting and Using Xilinx UCF Constraints](#), on page 255.
3. Synthesize the design.

The synthesis tool writes out the timing constraints and physical constraints into separate files:

|                |                                                                                       |
|----------------|---------------------------------------------------------------------------------------|
| synplicity.ucf | Contains all timing constraints, whether user-specified or translated from a ucf file |
|----------------|---------------------------------------------------------------------------------------|

---

|              |                                   |
|--------------|-----------------------------------|
| <design>.ncf | Contains all physical constraints |
|--------------|-----------------------------------|

---

4. Use synplicity.ucf and <design>.ncf as input to the Xilinx place-and-route tool. Update scripts or older par\_opt files if needed to ensure that these files are used to drive place-and-route.

## Using -route for Physical Synthesis in Xilinx Designs

For the Xilinx physical synthesis flows, use the -route constraint as follows:

- Remove it for logic synthesis

During logic synthesis the -route option either tightens or loosens timing constraints, without affecting constraints forward annotated to the Xilinx ISE place-and-route tool. The Synplify Pro software tunes the delay estimates to compensate for differences between logic synthesis and the actual place-and-route delays. The Synplify Premier tool uses a different timing model for interconnect delays, therefore, the -route constraint generally should not be used.

- Use it at a global level in a physical synthesis run to produce a better netlist during global placement.

The Synplify Premier graph-based timing estimation for critical paths automatically handles the correlation between the synthesis and place-and-route tools. For physical synthesis, apply the -route constraint to global clocks from the SCOPE Clock panel. For example:

```
define_clock {clk} -period 4 -clockgroup cg1 -route 1
```

Remember to monitor the effects of using the -route constraint from the Pre-placement Timing Snapshot report in the log file. A clock constrained by the -route option should target a slightly negative slack.

## 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 show you how to specify timing exceptions in the SCOPE GUI. For the equivalent Tcl syntax, see [Chapter 14, \*Tcl Commands and Scripts\*](#) in the *Reference Manual*.

- [Defining From/To/Through Points for Timing Exceptions](#), on page 231
- [Defining Multi-cycle Paths](#), on page 234
- [Defining False Paths](#), on page 235

For information about resolving timing exception conflicts, see [Conflict Resolution for Timing Exceptions](#), on page 451 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 451 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.

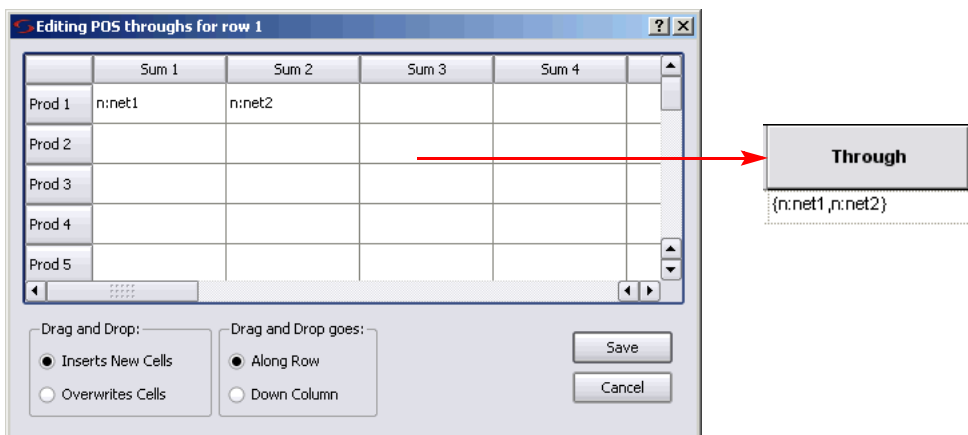
- In the From field, identify the starting point for the path. The starting point can be a clock (c:), register (i:), top-level input or bi-directional port (p:), or black box output (i:). To specify multiple starting points:
  - Such as the bits of a bus, enclose them in square brackets: A[15:0] or 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:

```
define_multicycle_path -from {i:aq i:bq} 2
define_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 (c:), register (i:), top-level output or bi-directional port (p:), or black box input (i:). 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 net (n:), hierarchical port (t:), or instantiated cell pin (t:). 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.

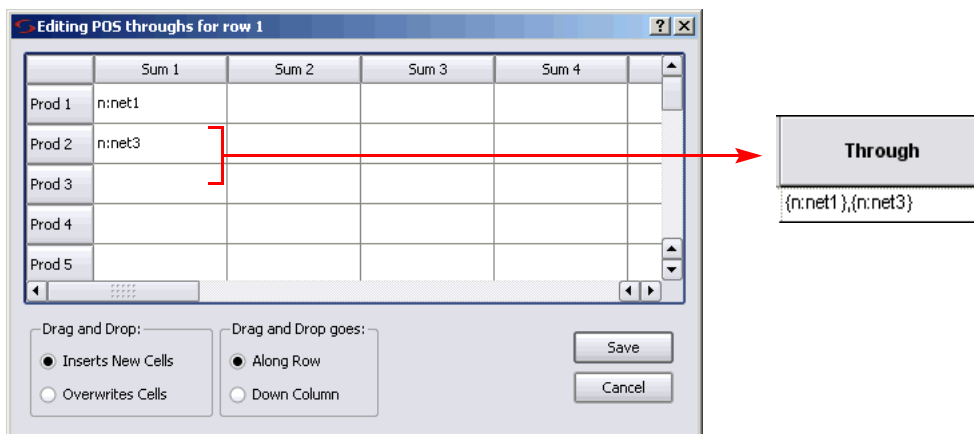


- Alternatively, select **Along Row** in the SCOPE POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names along the row. The nets in each row form an OR list.

- 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*.

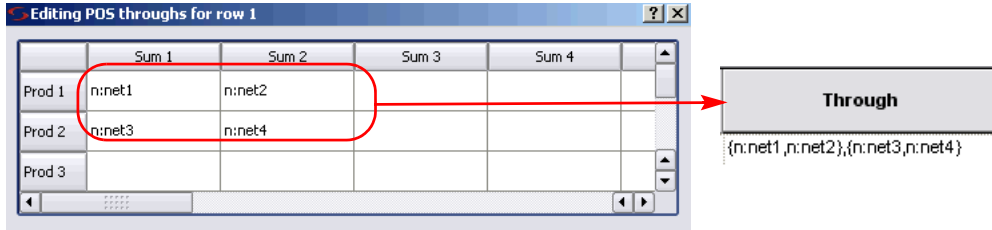
- To specify an AND when constraining a list of through points, type the names in the Through field or do the following:
  - Open the Product of Sums interface as described previously.
  - Either type the first net name in the first cell in a Sum column or drag the net from an HDL Analyst view into the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

To define a multi-cycle path constraint, use the Tcl `define_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 231](#) for more information.
3. Select another port or register if needed (From/To/Through).
4. Type the number of clock cycles (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 or Clock to Clock tabs. You can also define false paths with the corresponding `define_false_path`, `define_clock`, and `define_clock_delay` Tcl commands. See [Defining From/To/Through Points for Timing Exceptions, on page 231](#) 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 define a false path between two clock edges, select the SCOPE Clock to Clock tab, and do the following:
  - Specify one clock as the starting clock edge (From Clock Edge).
  - Specify the other clock as the ending clock edge (To Clock Edge).
  - Click in the Delay column, and select false.
  - Mark the Enabled check box.

Use this technique to specify a false path between any two clocks, regardless of clock groups. This constraint can be overridden by a maximum delay constraint on the same path.

- To override an implicit false path between any two clocks described previously, set an explicit constraint between the clocks by selecting the SCOPE Clock to Clock tab, and doing the following:
  - Specify the starting (From Clock Edge) and ending clock edges (To Clock Edge).
  - Specify a value in the Delay column.
  - Mark the Enabled check box.

The software treats this as an explicit constraint. You can use this method to constrain a path between any two clocks, regardless of whether they belong to the same clock group.

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



## Using Collections

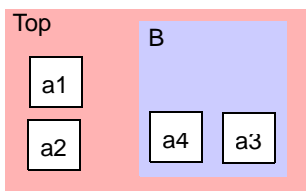
A collection is a group of objects. It can consist of just one object, or of other collections. You can set the same constraint for multiple objects if you group them together in a collection. You can either define collections in the SCOPE window or type the commands in the Tcl script window. The Synplify tool does not support collections.

- [Comparing Methods for Defining Collections](#), on page 237
- [Creating and Using Collections \(SCOPE Window\)](#), on page 238
- [Creating Collections \(Tcl Commands\)](#), on page 241
- [Using the Tcl Find Command to Define Collections](#), on page 244
- [Using the Expand Tcl Command to Define Collections](#), on page 246
- [Viewing and Manipulating Collections \(Tcl Commands\)](#), on page 247

## Comparing Methods for Defining Collections

The find and expand Tcl commands that are used to define collections in the Synplify Premier and Synplify Pro software can either be entered in the Tcl script window or in the SCOPE window. It is recommended that you use the SCOPE interface for two reasons:

- When you use the SCOPE interface, the software uses the top-level database to find objects, which is a good practice. The Tcl window commands are based on the current Analyst view. If you use the Tcl script window to type in a command after mapping, the search is based on the mapped database, which could have instances that have been renamed, replicated, or removed.



Similarly, the current Analyst view could be a lower-level view. In the design shown above, 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

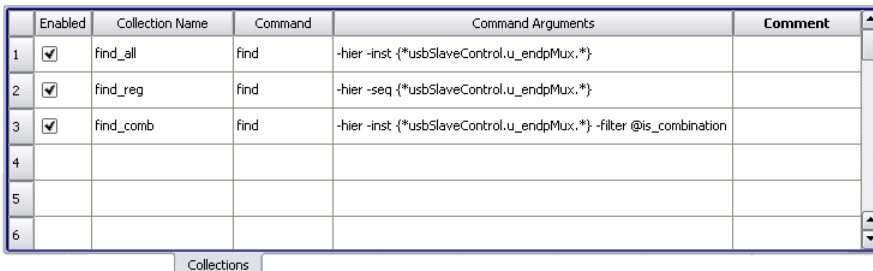
and paste the same command into the SCOPE Collections tab, your results would include a1, a2, a3, and a4, because the SCOPE interface uses the top-level database and searches the entire hierarchy.

- If you use the Tcl script window, you have to redefine the collection the next time you open the project. When you define a collection in the SCOPE window, the software saves the information in the constraint file for the project.
- You cannot apply constraints to collections defined in the Tcl script window, but you can apply constraints and attributes to SCOPE collections.

## Creating and Using Collections (SCOPE Window)

The following procedure shows you how to define collections in the Synplify Pro or Synplify Premier SCOPE window. You can also type the commands directly in the Tcl script window ([Creating Collections \(Tcl Commands\)](#), on page 241). See [Comparing Methods for Defining Collections](#), on page 237 for a comparison of the two methods.

1. Define a collection by doing the following:
  - Open the SCOPE window and click the Collections tab.
  - In the Collection Name column, type a name for the collection. This is equivalent to defining the collection with the set command, as described in [Creating Collections \(Tcl Commands\)](#), on page 241.



|   | 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 |                                     |                 |         |                                                                    |         |

- In the Commands column, select find or expand. For tips on using these commands, see [Using the Tcl Find Command to Define Collections](#), on page 244 and [Using the Expand Tcl Command to Define Collections](#), on page 246. For complete syntax details, see the *Reference Manual*.

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. See [Comparing Methods for Defining Collections, on page 237](#).

- In the Command Arguments column, type only the arguments to the command you set in the Commands column, so that you locate the objects you want. Do not repeat the command itself. For details of the syntax, see the *Reference Manual*. Objects in a collection do not have to be of the same type. The collections defined above do the following:

| Collection | Finds...                                   |
|------------|--------------------------------------------|
| find_all   | All components in the module endpMux       |
| find_reg   | All registers in the module endpMux        |
| find_comb  | All combinatorial components under endpMux |

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

- To crossprobe the objects selected by the find and expand commands, click Select in the Select in Analyst column. The schematic views highlight the objects located by these commands. For other viewing operations, see [Viewing and Manipulating Collections \(Tcl Commands\), on page 247](#).
2. 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 Collection Name column. Set Commands to one of the operator commands like c\_union or c\_diff. Type the appropriate arguments in Command Arguments. See [Creating Collections \(Tcl Commands\), on page 241](#) for a list of available commands and the *Reference Manual* for the complete syntax.
    - Click Run Commands. The software runs through the commands in sequence, so you must first define collections before doing any group or comparative operations.

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

3. 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 |
|-------------------------------------|------------|--------------------------|----|---------|-----------|--------|---------------|---------|
| <input checked="" type="checkbox"/> | False      | special_regs.status[7:0] |    |         |           |        |               |         |
|                                     |            | i:decode.opcode_goto     |    |         |           |        |               |         |
|                                     |            | i:decode.opcode_call     |    |         |           |        |               |         |
|                                     |            | i:decode.opcode_retlw    |    |         |           |        |               |         |
|                                     |            | i:decode.skip            |    |         |           |        |               |         |
|                                     |            | i:decode.decodes[13:0]   |    |         |           |        |               |         |
|                                     |            | i:prgmcntr..r_0_[10:0]   |    |         |           |        |               |         |
|                                     |            | i:prgmcntr..r_1_[10:0]   |    |         |           |        |               |         |
|                                     |            | i:prgmcntr..r_2_[10:0]   |    |         |           |        |               |         |
|                                     |            | i:prgmcntr..r_3_[10:0]   |    |         |           |        |               |         |
|                                     |            | i:prgmcntr..r_4_[10:0]   |    |         |           |        |               |         |

Delay Paths

- Specify the rest of the constraint as usual. The software applies the constraint to all the objects in the collection. See examples of constraints in [Example: Attribute Attached to a Collection, on page 240](#).

### Example: Attribute Attached to a Collection

The following example shows the `xc_area_group` attribute applied to `$find_reg`, which results in all the registers in this collection being placed in the same region. Check the `.sr` file, the netlist, and if you are using Synplify Premier, the Design Planner view to see that the attribute is honored.

| Enabled                             | Object Type | Object     | Attribute     | Value       | Val Type | Description                                          | Comment |
|-------------------------------------|-------------|------------|---------------|-------------|----------|------------------------------------------------------|---------|
| <input checked="" type="checkbox"/> | instance    | \$find_reg | xc_area_group | rr#cc#rr#cc | string   | Specifies the region where instance should be placed |         |
|                                     |             |            |               |             |          |                                                      |         |
|                                     |             |            |               |             |          |                                                      |         |
|                                     |             |            |               |             |          |                                                      |         |
|                                     |             |            |               |             |          |                                                      |         |
|                                     |             |            |               |             |          |                                                      |         |

Attributes

## Creating Collections (Tcl Commands)

This section describes how to type in and use the Tcl collection commands instead of the SCOPE window ([Creating and Using Collections \(SCOPE Window\), on page 238](#)). Although you can type these commands in the Tcl window (Synplify Premier and Synplify Pro) or put them in a Tcl script, it is recommended that you use the SCOPE window, for the reasons described in [Comparing Methods for Defining Collections, on page 237](#).

For details of the syntax for the commands described here, refer to [Tcl Collection Commands, on page 1251](#) in the *Reference Manual*.

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

A collection can consist of individual objects, Tcl lists (which can have single elements as arguments), or other collections. Use the Tcl find and expand commands to locate objects for the collection (see [Using the Tcl Find Command to Define Collections, on page 244](#) and [Using the Expand Tcl Command to Define Collections, on page 246](#)). The following example creates a collection called my\_collection which consists of all the modules (views) found by the 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 | c_union. See <a href="#">Examples: c_union Command, on page 242</a>   |
| Concatenate collections     | c_union. See <a href="#">Examples: c_union Command, on page 242</a> . |

| To...                                                        | Use this command...                                                                                      |
|--------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| Create a collection from the differences between collections | <code>c_diff</code> . See <a href="#">Examples: <code>c_diff</code> Command, on page 243</a> .           |
| Create a collection from common objects in collections       | <code>c_intersect</code> . See <a href="#">Examples: <code>c_intersect</code> Command, on page 243</a> . |
| Find objects that belong to just one collection              | <code>c_symdiff</code> . See <a href="#">Examples: <code>c_symdiff</code> Command, on page 243</a> .     |

- If your Tcl collection includes instances that have special characters make sure to use extra curly braces or use a backslash to escape the special character. See [Examples: Names with Special Characters, on page 244](#) for details.

Once you have created a collection, you can do various operations on the objects in the collection (see [Viewing and Manipulating Collections \(Tcl Commands\), on page 247](#)), 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_syndiff` 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_syndiff $collection1 {i:reg1 i:reg3}]
c_list $diffCollection
 {"i:reg2" "i:reg3"}
```

You can also use the command to compare two collections:

```
set syndiff_collection [c_syndiff $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\[*\}}
```

## 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 [Comparing Methods for Defining Collections, on page 237](#).

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 ([Using the Expand Tcl Command to Define Collections, on page 246](#)). This section lists some tips for using the Tcl find command.

1. Tcl find always searches at the top-level of your design, regardless of the current Analyst view.
2. Create a collection by typing the find 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 == FDE]
```

The result is a random number like s:49078472, which is the collection of objects found. For a list of some useful find commands, see [Examples: Useful Find Commands, on page 246](#).

3. The following table lists some usage tips for specifying the find command. For the full details of the syntax, refer to [Tcl find Command, on page 1260](#) 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 Synplify Pro and Synplify Premier mixed language designs, use the case rules for the parent module. This example finds any object in the current view that starts with either a or A:</p>                                                            |
| Pattern matching                                                 | <p>You have two 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 <code>*</code> and <code>?</code> wildcards for pattern matching.</li> </ul>                                                                                                                                                                                                                            |
| Restricting search by type of object                             | <p>Use the <code>-object_type</code> argument. The following command finds all nets that contain <code>syn</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Restricting search to hierarchical levels below the current view | <p>Use the <code>-hier</code> argument. The following example finds all objects below the current view that begin with <code>a</code>:</p>                                                                                                                                                                                                                                                                                                                                                                                                       |
| Restricting search by object property                            | <ul style="list-style-type: none"> <li>• Select Project-&gt;Implementation Options. On the Device tab, enable Annotated Properties for Analyst.</li> <li>• Compile or synthesize the design. After the compile stage, the tool annotates the design with properties like clock pins. You can find objects based on these annotated properties.</li> <li>• Use the <code>-filter</code> argument to the <code>find</code> command. The following example finds any register in the current view that is clocked by <code>myclk</code>.</li> </ul> |

```
find {a*} -nocase
```

```
find -net {*syn*}
```

```
find {a*} -hier
```

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

4. 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 \(Tcl Commands\)](#), on page 247:

- Select the collection in an HDL Analyst view (`select`).
  - Print the collection using the `-print` option to the `find` command.
  - Print the collection without carriage returns or properties (`c_list`).
  - Print collection in columns, with optional properties (`c_print`).
5. To manipulate the objects in the collection, use the commands described in [Viewing and Manipulating Collections \(Tcl Commands\)](#), on page 247.

## Examples: Useful Find Commands

| To find...                              | Use a command like this example...                                                                            |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------|
| Instances by slack value                | <code>set result [find -hier -inst {*} -filter @slack &lt;= {-1.000}]</code>                                  |
| Instance within a slack range           | <code>set result [find -hier -inst {*} -filter @slack &lt;= {-1.000} &amp;&amp; @slack &gt;= {+1.000}]</code> |
| Pins by fanin/fanout value              | <code>set result [find -hier -inst {*.D} -filter @fanin &lt;= {50}]</code>                                    |
| Sequential components by primitive type | <code>set result [find -hier -seq {*} -filter @view=={FDRSE}]</code>                                          |

## Using the Expand Tcl Command to Define Collections

The Tcl `expand` command returns a collection 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 [Tcl expand Command](#), on page 1257 in the *Reference Manual*.

- Specify at least one from, to, or through 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 through 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}
```

- 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 through 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
```

- 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}
```

- To print a list of the objects found, either use the `-print` argument to the `find` command, or use the `c_print` or `c_list` commands (see [Creating Collections \(Tcl Commands\)](#), on page 241).

## Viewing and Manipulating Collections (Tcl Commands)

The following section describes various operations you can do on the collections you defined. For full details of the syntax, see [Tcl Collection Commands](#), on page 1251 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 250](#).

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:EP0RxFifo.u_fifo.dataOut[0]}
{i:EP0RxFifo.u_fifo.dataOut[1]}
{i:EP0RxFifo.u_fifo.dataOut[2]}
{i:EP0RxFifo.u_fifo.dataOut[3]}
{i:EP0RxFifo.u_fifo.dataOut[4]}
{i:EP0RxFifo.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:EP0RxFifo.u_fifo.dataOut[0]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[1]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[2]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[3]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[4]} 0.3223 "FDE" clk
```

```
{i:EP0RxFifo.u_fifo.dataOut[5]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[6]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[7]} 0.3223 "FDE" clk
{i:EP0TxFifo.u_fifo.dataOut[0]} 0.1114 "FDE" clk
{i:EP0TxFifo.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 [Tcl Collection Commands](#), on page 1251 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> <pre><b>set my_collection_copy \$my_collection</b></pre>                                                                                                                                                                                       |
| 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> |

| To...                        | Do this...                                                                                                                                                                             |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Iterate through a collection | Use the <code>foreach</code> command. This example iterates through all the objects in the collection:<br><pre>foreach port [find -port *] {   define_false_path -from \$port } </pre> |

### 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"
```

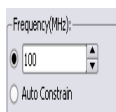
This returns the following, which shows that the terminal (pin) has been converted to the owning instance:

```
terminal owner is i:EP0RxFifo.u_fifo.dataOut[0]
i:EP0RxFifo.u_fifo.dataOut[1] i:EP0RxFifo.u_fifo.dataOut[2]
```

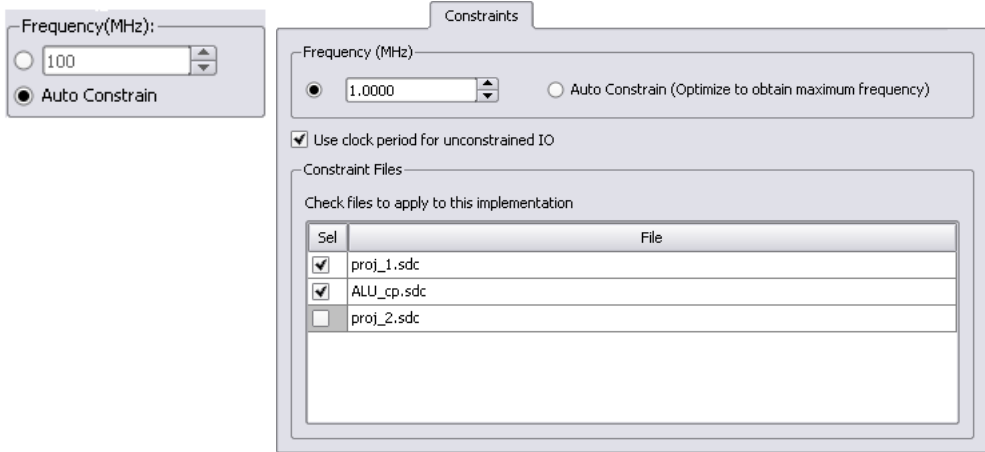
## Using Auto Constraints

Auto constraining is available for certain technologies in both Synplify Pro and Synplify Premier, however, the Physical Synthesis option must be disabled in the Synplify Premier tool. You can 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 [Auto Constraints, on page 386](#) in the *Reference Manual*.

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 `define_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_<design_name>.sdc` in the run directory. It also forward-annotates these constraints to the place-and-route tools.

5. Check the results in `AutoConstraint_<design_name>.sdc` and the log file. To open the `.sdc` file as a text file, right-click the file in the Implementation Results view and select Open as Text.



The flop-to-flop constraints use syntax like the following:

```
define_clock -name {b:leon|clk} -period 13.327 -clockgroup
Autoconstr_clkgroup_0 -rise 0.000 -fall 6.664 -route 0.000
```

6. You can now add the generated .sdc file to the project and rerun synthesis with these constraints.

## Translating Altera QSF Constraints

If you have an Altera Quartus Settings File (QSF) with timing or pad constraints, you can use the `qsf2sdc` translator to translate these constraints to the `sdc` format and use the translated constraints to drive synthesis.

1. Run the `qsf2sdc` utility.
  - Make sure the input QSF file has a `.qsf` extension.
  - From the command line, run the translator on the QSF file. The translator is in the `bin` directory: `install_dir/bin/qsf2sdc.exe`. Use the following syntax:

```
<install_dir>/bin/qsf2sdc -iqsf <constraints_file>.qsf
-osdc <constraints_file>.sdc
[-oqsf <residual_constraints_file>.qsf] [-all]
[-silent]
```

The translator generates a constraint file in the `sdc` format, which contains the timing-related constraints from the QSF file that are relevant to synthesis. It ignores the other backend constraints in the file. See [Altera `qsf2sdc` Utility, on page 541](#) in the *Reference Manual* for details of the syntax and a list of supported pin location and I/O constraints.

2. After translating the constraints, edit the new `.sdc` file.

The translator converts the most common timing and physical constraints. However, because of the diversity and complexity of QSF format, the resulting `.sdc` file requires manual intervention.

- Visually inspect the translated file.

The original QSF commands are written as comments in the new `.sdc` file so that you can validate the translated constraints. Constraints

which were successfully translated are specified as Supported. However, constraints which were unsuccessfully translated are specified as Unsupported. Use the `-silent` option to suppress all the `#Supported` and `#Unsupported` messages in the `.sdc` file.

- Manually edit the SDC file to complete the translation of constraints, as necessary.
  - Optionally, use the `-all` option to convert any instances with location assignments. By default, only pin location assignments and IO standards are automatically converted.
3. To run physical synthesis, create one `.sdc` file.
- Include timing constraints created previously into the `.sdc` file containing the translated physical constraints. Make sure that all of the following types of constraints are combined into the `.sdc` file

Timing Constraints:

Clock

Clock-to-clock

IO delays

IO standard, drive, slew and pull-up/pull-down

Multi-cycle and false paths

Max-delay paths

DCM parameters

Physical Constraints

SYN\_LOC on IO pins and pad types

Physical constraints applied to invariant objects (such as registers, instantiated macros and modules) can be safely translated to SDC constraints. Use the Design Planner tool for advanced physical constraints.

- Include any synthesis attributes from logic synthesis, such as `syn_ramstyle`, into the `.sdc` file.
4. Edit the original QSF file.
- Remove all translated constraints from the original `.qsf` file.
  - If there are any untranslated QSF commands left in the file, add the `.qsf` file to your project. The file must have the same base name as the `.vqm` netlist so that the Altera P&R tool can source the file.

5. Run a constraint check by selecting Run->Constraint Check.

This command generates a report that checks the syntax and applicability of the timing constraints in the .sdc file(s) for your project. The report is written to the *project\_name\_cck.rpt* file.

6. Add the generated .sdc file to the project, and use it to drive synthesis.

## Converting and Using Xilinx UCF Constraints

As you iterate through the flow, you might want to use Xilinx UCF constraints to guide synthesis. To do this, you must translate the UCF constraints into SDC constraints that the synthesis tools can use. The following procedures show you how convert the UCF constraints in logic and physical synthesis designs and then forward-annotate them for place-and-route. Although this utility is also documented here, it is recommended that you do not use this method, as it will not be available in future releases of the software.

- [Using Xilinx UCF Constraints in a Logic Synthesis Design](#), on page 255
- [Using Xilinx UCF Constraints in a Physical Synthesis Design](#), on page 258
- [Support for UCF Conversion](#), on page 260
- [Using the Legacy UCF2SDC Utility](#), on page 264.  
The process for the command-line `ucf2sdc_old` utility (see [Xilinx Legacy ucf2sdc Utility, on page 537](#) in the *Reference Manual*) is a little different from the other method described, as it does not use mapper information, run the constraint checker, or create a new project. The utility is still available, but will not be supported in future releases.

### Using Xilinx UCF Constraints in a Logic Synthesis Design

You can run logic synthesis in the Synplify Pro tool or in the Synplify Premier tool in logic synthesis mode. The following procedure shows you how to use Xilinx UCF constraints for a logic synthesis run with either of the tools.

1. Start with the Xilinx constraint files to be translated.

- You can use the following kinds of files:

---

UCF Top-level constraint file, with corresponding EDIF file (edf)

---

NCF Block-level constraint file, with corresponding EDIF file (edn, edf, ngc or ngo)

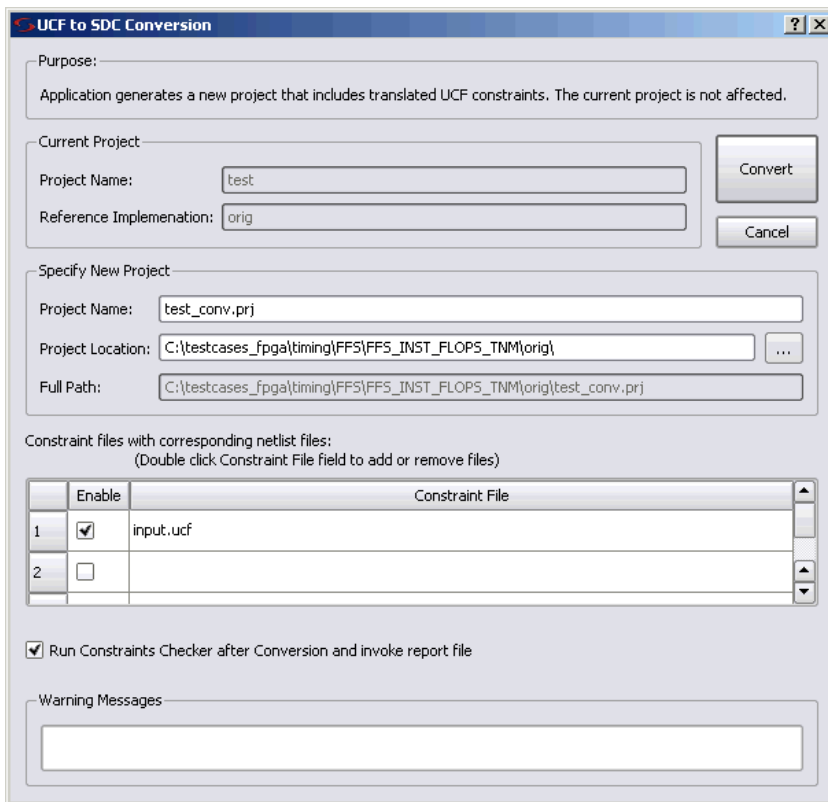
---

XCF Block-level constraint file, with corresponding EDIF file (ngc or ngo)

---

These files must refer to design objects in the mapped synthesis tool database so as to be consistent with subsequent synthesis runs. If you use a UCF file that refers to XST design objects, naming might be inconsistent. You can have multiple constraint files, one for the top-level, and others for blocks. See [Supported Input Files for UCF Conversion, on page 261](#) for details about the input files.

- Add all Xilinx constraint files to be converted to the logic synthesis project.
  - Add the corresponding netlist files to the project, along with the constraint file.
2. Do an initial synthesis run.
    - Set up a P&R implementation.
    - Synthesize the design and run P&R.
    - Check the log files for any constraint-related warnings and fix them before proceeding.
  3. Select Project->Convert Vendor Constraints to open the UCF to SDC Conversion dialog box.



4. Specify the translation options:

- Specify a name for the new project in Project Name.
- Set a location for the new project in Project Location.
- In the Constraint Files section, enable the files you want to use. This section lists the files you added to the project in step 2. If you do not have corresponding EDIF files for the constraint files you enable, you see warning messages in the box at the bottom of the dialog box.
- Enable Run Constraints Checker after Conversion and Invoke Report File.
- Click the Convert button in the upper right.

The tool uses information from the project .srd file and translates the constraints in the input files, using a separate process for the top level and for each block. It then creates a new project. Note that it does not delete the original project or files, but creates a new one. See

[Generated Files after UCF Conversion, on page 261](#) for names and descriptions of the files generated after conversion.

Finally, it runs the constraints checker and reports any Xilinx constraints that cannot be translated. See [Support for UCF Conversion, on page 260](#) for information about supported and unsupported constraints.

- Check the `ucf2sdc.log` file for any errors or warnings.
5. To use the generated `sdc` file to drive synthesis for the new project, do the following:
    - Open the `sdc` file and check it. Edit it if necessary. You can also rename this file.
    - Make sure the file is added to the project.
    - Run logic synthesis by clicking Run.
  6. After logic synthesis, you can do either or both of the following:
    - Use the newly-generated project and the `sdc` files with translated constraints for physical synthesis. See [Using Xilinx UCF Constraints in a Physical Synthesis Design, on page 258](#) for more information.
    - Use the `synplicity.ucf` and `unsupported.ucf` files for Xilinx P&R. You can use the `ucf2sdc.log` file and the `unsupported.ucf` file to manually translate any remaining constraints.

## Using Xilinx UCF Constraints in a Physical Synthesis Design

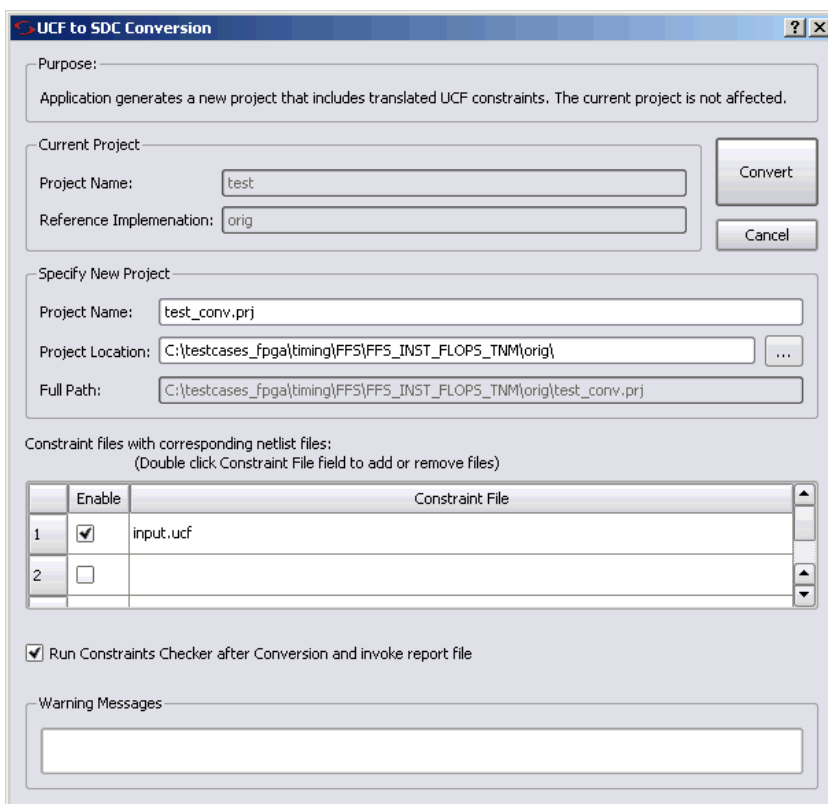
When you use UCF constraints in a physical synthesis design, it is assumed that you did a baseline logic synthesis run, and that you have certain files for the design. The following procedure shows you how to translate and use UCF constraints in a physical synthesis design with the Synplify Premier software.

1. Run through an initial logic synthesis and P&R run, with either the Synplify Pro or the Synplify Premier tool in logic synthesis mode. See step 1 of [Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 255](#).

For Synplify Premier designs, the conversion of UCF constraints requires the `.srd`, `.prj`, and `.ucf` files.

2. Do not enable physical synthesis.

3. Select Project->Convert Vendor Constraints to open the UCF to SDC Conversion dialog box.



4. Do the following:
  - Specify a name for the new project in Project Name.
  - Set a location for the new project in Project Location.
  - In the Constraint Files section, check that the files you want to translate are enabled. The list should include the files you added in step 2. If you do not have corresponding EDIF files for the constraint files you enable, you see warning messages in the box at the bottom of the dialog box.
  - Enable Run Constraints Checker after Conversion and Invoke Report File.
  - Click the Convert button in the upper right.

The tool uses information from the project `.srd` file, the `.prj` file, and the `.ucf` files. It translates the constraints in the Xilinx input constraint files and creates a new project. It does not overwrite or delete the files from the input project. Finally, it runs the constraints checker and reports any Xilinx constraints that cannot be translated. See [Supported UCF Constraints, on page 263](#) for a list of supported and unsupported constraints.

It generates the following output constraint files:

---

|                                       |                                                                                                                                                                  |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;ucfFile&gt;_conv.sdc</code> | Translated UCF constraints. Could have multiple files.                                                                                                           |
| <code>unsupported.ucf</code>          | Unsupported input Xilinx constraints, in the ucf format for forward-annotation. This includes any physical constraints that are not used by the synthesis tools. |

---

- To use the generated `sdc` file to drive further synthesis, do the following:
  - Open the `ucf2sdc.log` and check each `ucf` translation section in this file for errors or warnings.
  - Open the `sdc` file and check it. Edit it if necessary.
  - Make sure the file is added to the project.
  - Run physical synthesis.
- Use the `synplicity.ucf` and `.ncf` files for Xilinx P&R.

You can check the `ucf2sdc.log` file for messages and manually translate any untranslated constraints in the `synplify.ucf` file before running P&R.

## Support for UCF Conversion

For procedures on converting UCF constraints, see the methods listed [Converting and Using Xilinx UCF Constraints, on page 255](#). The following describe what the software supports when translating UCF constraints to SDC.

- [Supported Input Files for UCF Conversion, on page 261](#)
- [Generated Files after UCF Conversion, on page 261](#)
- [Supported UCF Constraints, on page 263](#)



## Supported Input Files for UCF Conversion

The synthesis software can translate Xilinx constraints from UCF, NCF, and XCF files with the Project->Convert Vendor Constraints command. The UCF file is for the top-level design, and the XCF and NCF files are for blocks. The following table lists support criteria for each of these formats:

---

|     |                                                                                                                                                                                                                                                                                                                      |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UCF | <ul style="list-style-type: none"><li>• You can only have UCF files for the top-level project.</li><li>• Paths referring to elements must start at the top level.</li><li>• The UCF file must be one written for the Synopsys FPGA synthesis netlist. If it is an XST netlist, object names may not match.</li></ul> |
| NCF | <ul style="list-style-type: none"><li>• You can only use block-level NCF files.</li><li>• A project can have multiple NCF files.</li><li>• Each NCF file must have a corresponding edn, edf, ngc, or ngo file with the same name.</li></ul>                                                                          |
| XCF | <ul style="list-style-type: none"><li>• You can only use block-level XCF files.</li><li>• A project can have multiple XCF files.</li><li>• Each XCF file must have a corresponding ngc or ngo file with the same name.</li></ul>                                                                                     |

---

## Generated Files after UCF Conversion

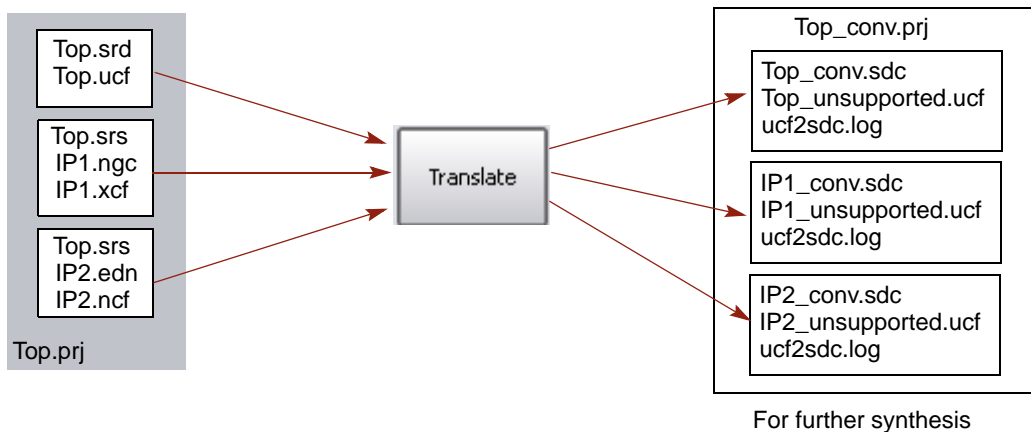
The tool creates these files after UCF conversion:

---

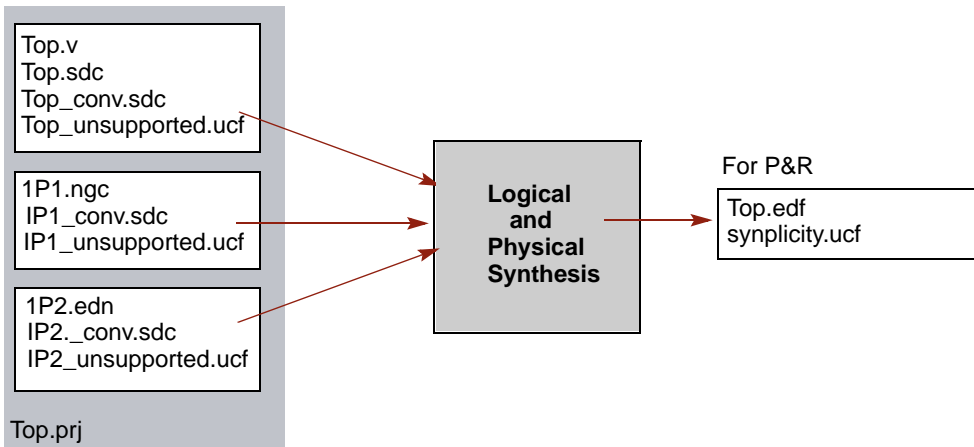
|                    |                                                                 |
|--------------------|-----------------------------------------------------------------|
| ucf2sdc.log        | Log file that contains messages after ucf conversion completes. |
| <prjFile>_conv.prj | The default name for the new project that was generated.        |

---

|                                       |                                                                                                                                                                                                                                                                                                         |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;ucfFile&gt;_conv.sdc</code> | Contains converted Xilinx constraints for logic synthesis. The tool generates a corresponding <code>sdc</code> file for each input <code>ucf</code> file. The name for this file is derived from the input UCF file name.                                                                               |
| <code>synplify.ucf</code>             | After logic synthesis, this one file contains all the supported input Xilinx constraints in the <code>ucf</code> format. Unsupported constraints are in a separate file.<br>After physical synthesis, this one file contains both supported and unsupported constraints in the <code>ucf</code> format. |
| <code>unsupported.ucf</code>          | File that contains all the unsupported Xilinx constraints in the <code>ucf</code> format after logic synthesis. This can include any physical constraints not used for synthesis.                                                                                                                       |



The next figure shows how the project-level input files are handled in a post-translation synthesis run:



## Supported UCF Constraints

The UCF converter supports the following types of constraints:

|               | FF  | RAM | ROM | DSP | Net | Inst | View | Collection | Port | Pin |
|---------------|-----|-----|-----|-----|-----|------|------|------------|------|-----|
| PERIOD        |     |     |     |     | Yes |      |      | Yes        | Yes  | Yes |
| FROM/TO       | Yes | Yes | Yes | Yes | Yes | Yes  |      | Yes        | Yes  | Yes |
| TIG           | Yes | Yes | Yes | Yes | Yes | Yes  |      | Yes        | Yes  | Yes |
| OFFSET        | Yes | Yes | Yes | Yes | Yes | Yes  |      | Yes        | Yes  | Yes |
| TNM           | Yes | Yes | Yes | Yes | Yes | Yes  | Yes  | Yes        | Yes  | Yes |
| TNM_NET       |     |     |     |     |     |      |      |            |      |     |
| TIMEGRP       |     |     |     |     |     |      |      |            |      |     |
| LOC           | Yes | Yes | Yes | Yes | Yes | Yes  |      | Yes        | Yes  | Yes |
| IO PROPS      |     |     |     |     | Yes | Yes  |      | Yes        | Yes  |     |
| General PROPS | Yes | Yes | Yes | Yes | Yes | Yes  | Yes  | Yes        | Yes  | Yes |

## Unsupported UCF Constraints

Currently, the UCF converter does not handle the following:

- Back-annotated netlists from the physical synthesis flow.
- Case-sensitive matching on instance and net names. For example: aBc.
- Nets driven by LUTs, except for nets that source OPADs.
- Collections that include inferred RAMs or DSPs. The tool cannot guarantee that inferred components match.
- The MAXDELAY constraint.

The UCF converter does not currently convert the following keywords:

| Unsupported Keywords                 | Description           |
|--------------------------------------|-----------------------|
| BRAMS_PORT[A/B]                      | Predefined keyword    |
| INPUT_JITTER, PRIORITY, DATAPATHONLY | TIMESPEC constraint   |
| RISING, FALLING                      | TIMEGRP constraint    |
| CLOSED, OPEN                         | AREA_GROUP constraint |
| HIGH, LOW, VALID                     | OFFSET constraint     |

## Using the Legacy UCF2SDC Utility

If you have a Xilinx User Constraint File (UCF) with timing or pad constraints, you can also use the `ucf2sdc_old` command to translate these constraints to the `sdc` format and use the translated constraints to drive synthesis. However, it is recommended that you use the methodology described in [Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 255](#) or [Using Xilinx UCF Constraints in a Physical Synthesis Design, on page 258](#), as the `ucf2sdc_old` command will be phased out in future releases.

1. Run this utility.
  - Make sure the input UCF file has a `.ucf` extension.
  - From the command line, run the translator on the UCF file. The translator is in the bin directory: `install_dir/bin/ucf2sdc_old.exe`. Use the following syntax:

```
<install_dir>/bin/ucf2sdc_old -iucf <constraints_file>.ucf
 -osdc <constraints_file>.sdc
```

The translator generates a file in the .sdc format, with the translated timing-related constraints from the UCF file. It ignores the other backend constraints in the UCF file. See [Xilinx Legacy ucf2sdc Utility](#), on page 537 in the *Reference Manual* for details.

The following table shows which UCF constraints are translated:

| Supported on INST, NET, PIN, SET |              | Supported on NET |
|----------------------------------|--------------|------------------|
| AREA_GROUP                       | PHASE_SHIFT  | COLLAPSE         |
| BLKNM                            | REG          | MAXDELAY         |
| BUFG                             | RLOC         | MAXSKEW          |
| DRIVE                            | RLOC_ORIGIN  | OPEN_DRAIN       |
| FAST                             | SLEW         | PULLDOWN         |
| HBLKNM                           | SLOW         | PULLUP           |
| HU_SET                           | STARTUP_WAIT | USELOWSKEWLINES  |
| IOB                              | TIMEGRP      | WIREAND          |
| IOBDELAY                         | TIMESPEC     |                  |
| IOSTANDARD                       | TNM          |                  |
| KEEP_HIERARCHY                   | TNM_NET      |                  |
| LOC                              | TPSYNC       |                  |
| MAP                              | TPTHRU       |                  |
| OPT_EFFORT                       | U_SET        |                  |
| OPTIMIZE                         | USE_RLOC     |                  |
| PERIOD                           | XBLKNM       |                  |

2. After translating the constraints, edit the new .sdc file.

The translator converts the most common timing and physical constraints. However, you still need to check it manually.

- Visually inspect the translated file. The original UCF commands are written as comments in the new .sdc file so that you can validate the translated constraints.
- Manually edit the SDC file to complete the translation of constraints, as necessary.

3. To run physical synthesis, create one .sdc file.

- Include the timing constraints created previously for logic synthesis into the .sdc file containing the translated physical constraints. Make sure that the following constraints are combined into one .sdc file:

### Timing Constraints

- Clock
- Clock-to-clock
- IO delays
- IO standard, drive, slew and pull-up/pull-down
- Multi-cycle and false paths
- Max-delay paths
- DCM parameters

### Physical Constraints

- Register packing into IOB
- LOC on IO pads
- Macro LOC/RLOC constraints (BUFG, DCM, DSP, MULT, etc.)
- Instance LOC/RLOCs (Register, LUT, SRL, RAMS, RAMD, etc.)
- AREA\_GROUP constraints

Physical constraints on invariant objects like registers, instantiated macros and modules, can be safely translated to SDC constraints. Use the Design Planner tool for advanced physical constraints.

- Also, include any synthesis attributes, from logic synthesis, such as `syn_ramstyle`, in the `.sdc` file.
4. Make sure all UCF constraints are forward-annotated to the P&R tool.
    - Edit the original input `.ucf` file and delete all constraints that were successfully translated to `sdc`.
    - If any untranslated UCF commands remain in the original file (such as the PROHIBIT constraint), add the edited `ucf` file to the project. This file now only contains the untranslated commands. These UCF constraints are not used for synthesis, but after synthesis, the tool copies the untranslated UCF commands to the `synplicity.ucf` file and appends the other `sdc` timing constraints to the same file. After synthesis, all timing constraints are in the `synplicity.ucf` file.
  5. Save the project file.
  6. Use the `.sdc` file to drive synthesis.

The tool generates two output constraint files after synthesis:

|                             |                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>synplicity.ucf</code> | Contains all timing constraints, whether user-specified <code>sdc</code> constraints, <code>ucf</code> constraints translated to <code>sdc</code> , or unsupported <code>ucf</code> constraints that are passed without translation. Physical constraints that are not used for synthesis are included in this file, along with other unsupported constraints. |
| <code>design.ncf</code>     | Contains all physical constraints for forward-annotation.                                                                                                                                                                                                                                                                                                      |

---

See [Specifying Xilinx Timing Constraints, on page 228](#) for an illustration.

7. Make sure to use the updated `synplicity.ucf` file and the `design.ncf` file to drive the Xilinx place-and-route tool. If necessary, update any scripts or `par_opt` files generated with older versions of the synthesis tools.

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.



---

## CHAPTER 6

# 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 270
- [Project File Hierarchy Management](#), on page 279
- [Setting Up Implementations and Workspaces](#), on page 285
- [Setting Logic Synthesis Implementation Options](#), on page 289
- [Entering Attributes and Directives](#), on page 304
- [Searching Files](#), on page 312
- [Archiving Files and Projects](#), on page 315

To set up a physical synthesis project, you follow the steps for setting up a logic synthesis project, and then additionally follow the procedures described in [Chapter 7, \*Setting up a Physical Synthesis Project\*](#).

## Setting Up Project Files

For a specific example on setting up a project file, refer to the Synplify and Synplify Pro tutorial. This section describes the following:

- [Creating a Project File](#), on page 270
- [Opening an Existing Project File](#), on page 273
- [Making Changes to a Project](#), on page 274
- [Setting Project View Display Preferences](#), on page 275
- [Updating Verilog Include Paths in Older Project Files](#), on page 277

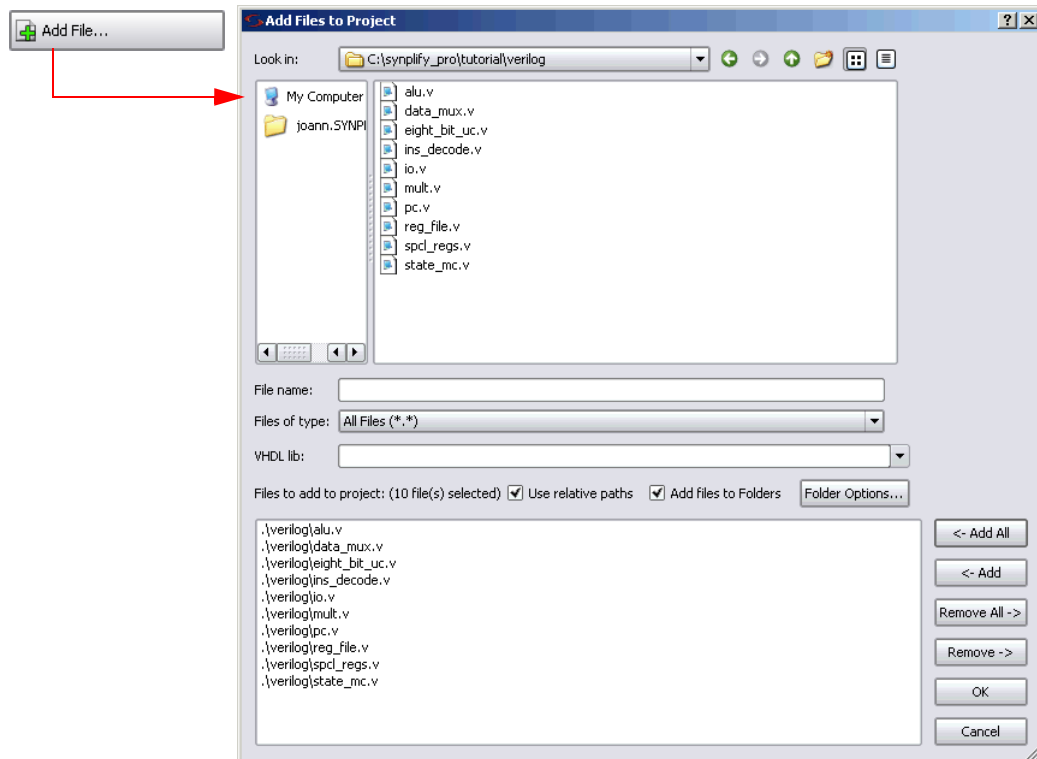
### Creating a Project File

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 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 95.



- 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.

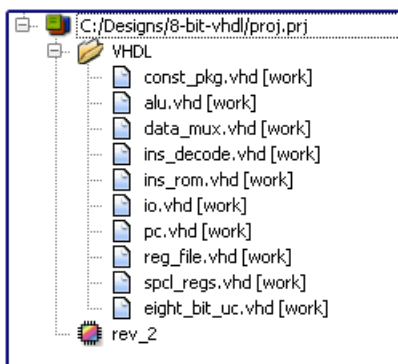
You can add all the files in the directory and then remove the ones you do not need with the Remove button.

If you are adding VHDL files, select the appropriate library from the 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 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. You can have multiple implementations in Synplify Premier and Synplify Pro, but not in Synplify. 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 vendor-specific 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 298](#). You can also set these file options later, before running synthesis.

For additional vendor-specific information about using vendor macro libraries and black boxes, see [Optimizing Actel Designs, on page 760](#), [Optimizing Altera Designs, on page 764](#), [Optimizing Lattice Designs, on page 785](#), and [Optimizing Xilinx Designs, on page 797](#).

- 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 especially important for VHDL files.
    - For VHDL files, you can automatically order the files by selecting Run->Arrange VHDL Files. Alternatively, manually move the files in the Project view. Package files must be first on the list because they are compiled before they are used. If you have design blocks spread over many files, make sure you have the following file order: the file containing the entity must be first, followed by the architecture file, and finally the file with the configuration.
    - 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.
  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 <b>Command</b>                                                                                                                                                                                                                                                                                                                       | File->Open <b>Command</b>                                                                                                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Select File->Open Project, click the Open Project button on the left side of the Project window (Synplify Pro and Synplify Premier only), or click the P icon.<br>To open a recent project, double-click it from the list of recent projects.<br>Otherwise, click the Existing Project button to open the Open dialog box and select the project. | Select File->Open.<br>Specify the correct directory in the Look In: field.<br>Set File of Type to Project Files (*.prj). The box lists the project files.<br>Double-click on 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 270](#) 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.

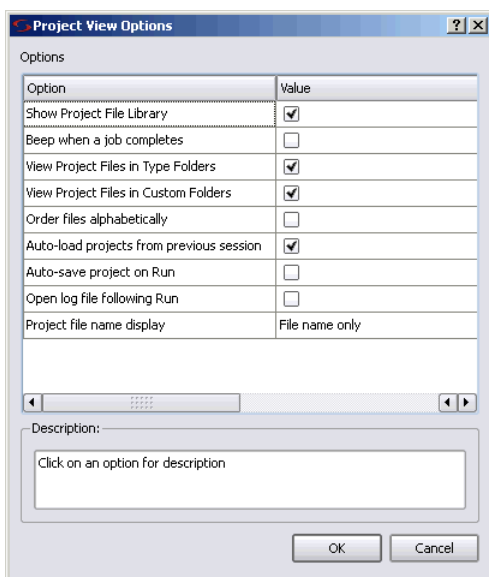
4. To specify how project files are saved in the project, right click on a file in the Project view and select File Options. Set the Save File option to either Relative to Project or Absolute Path.
5. To check the time stamp on a file, right click on a file in the Project view and select File Options. Check the time that the file was last modified. Click OK.

## Setting Project View Display Preferences

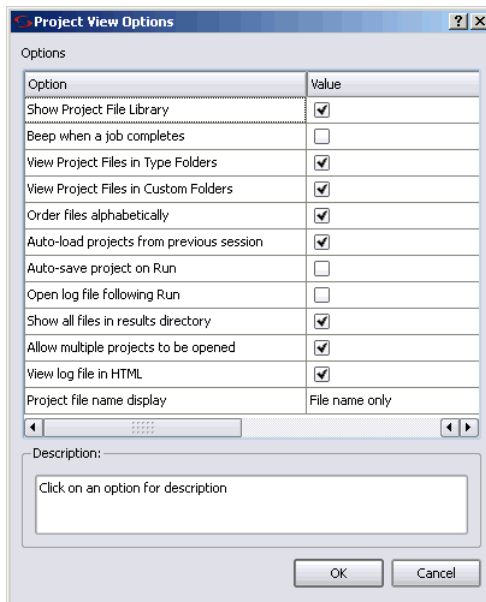
You can customize the organization and display of project files.

1. Select Options->Project View Options.

The Project View Options form opens. Available options vary, depending on the tool. The Synplify Premier and Synplify Pro options are the same.



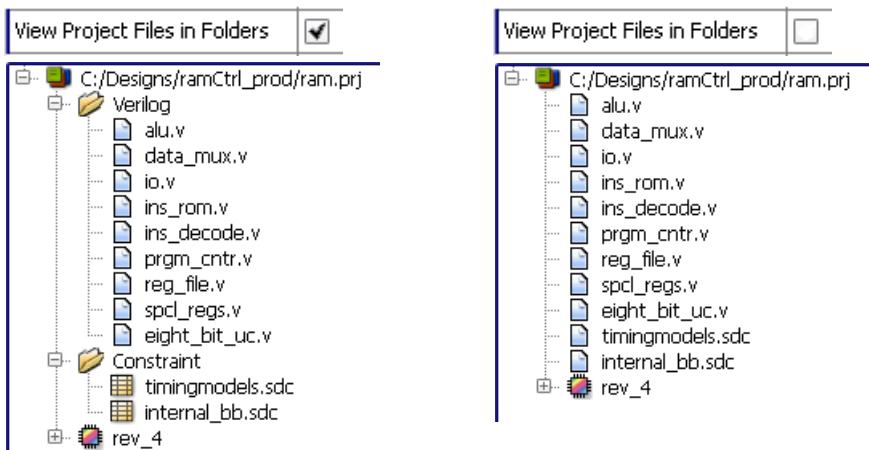
Synplify Options



Synplify Pro and Synplify Premier Options

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.

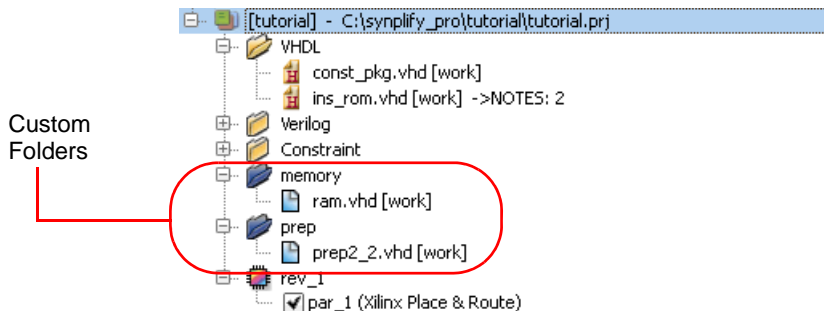


3. Control file display with the following:

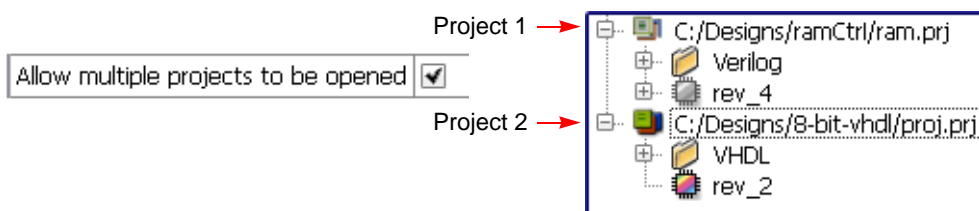
- Automatically display all the files, by checking Show Project Library. If this is unchecked, the Project view does not display files until you click on the plus symbol and expand the files in a folder.
- Check one of the boxes in the Project File Name Display section of the form to determine how filenames 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 279](#). 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. You can only use multiple implementations with the Synplify Pro and Synplify Premier tools.



- 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.

## 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 dir 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.

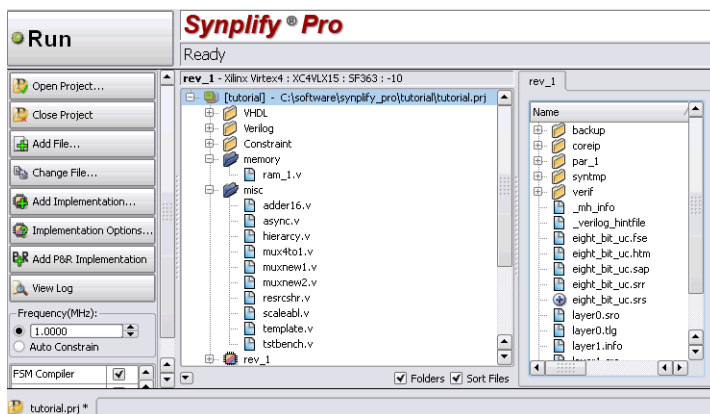
# Project File Hierarchy Management

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

- [Creating Custom Folders](#)
- [Other Custom Folder Operations](#)
- [Other Custom File Operations](#)

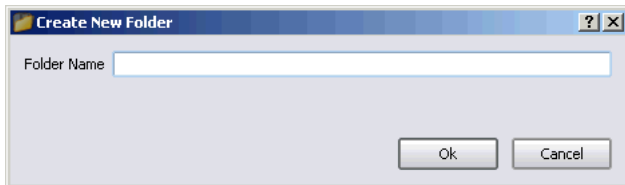
## Creating Custom Folders

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.



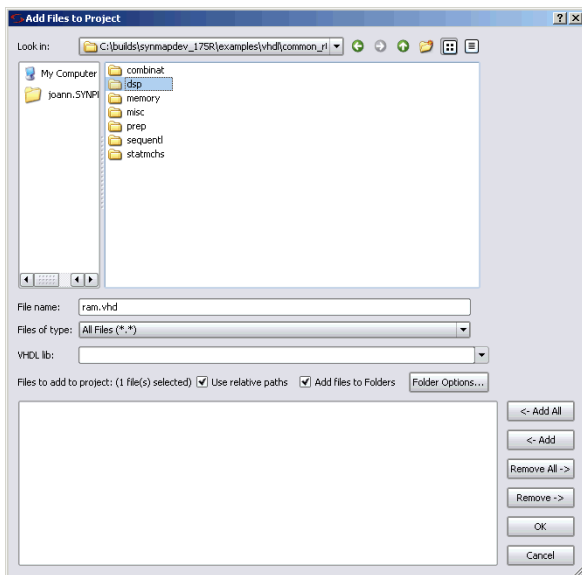
There are several ways to create custom folders and then add files to them in a project. Use one of the following methods:

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.



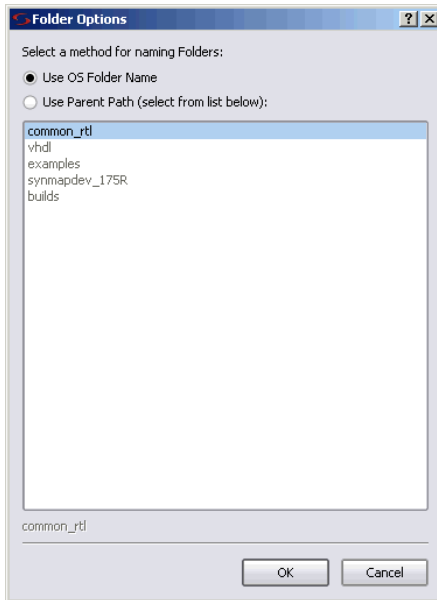
Note that you can arbitrarily name the folder, however do not use the character (/) because this is a hierarchy separator symbol.

- 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 the requested folder(s) 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 or Solaris 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.

## Other Custom Folder Operations

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:

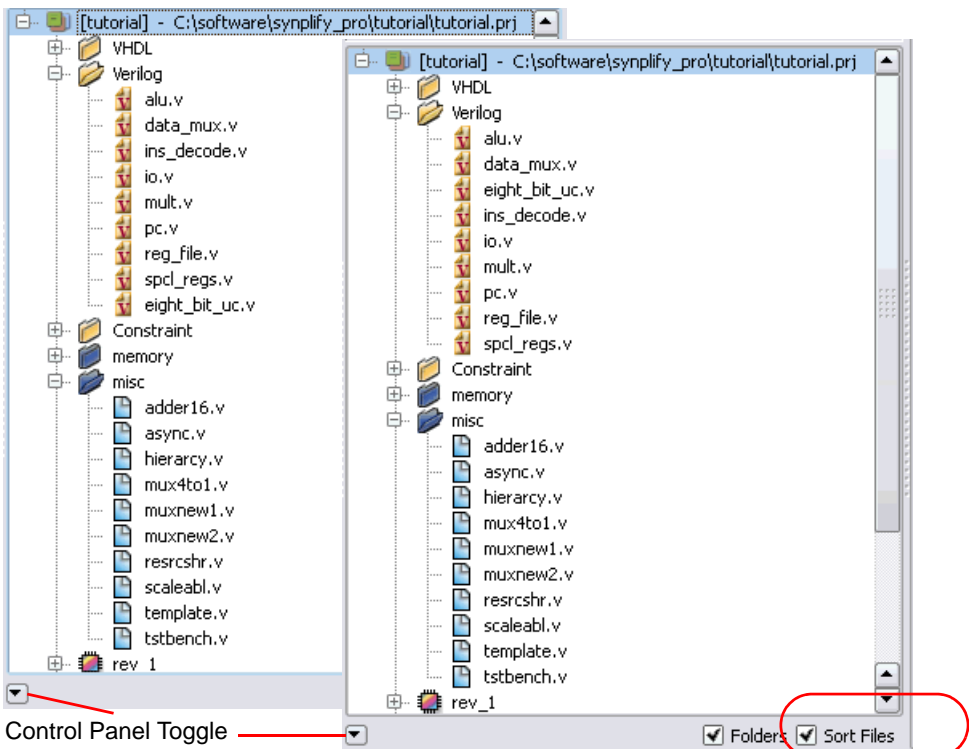
```
/Examples/Verilog/RTL
```

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.

## Other Custom File Operations

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 and Workspaces

Workspaces and implementations are extensions of the project metaphor used in the Synplify Pro and Synplify Premier synthesis software. The Synplify software does not support multiple implementations or workspaces.

This section describes the following:

- [Working with Multiple Implementations](#), on page 285
- [Creating Workspaces](#), on page 287
- [Using Workspaces](#), on page 288

## Working with Multiple Implementations

The Synplify Premier and Synplify Pro 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 .tlg log file, the .srs RTL netlist file, and the <design>\_fsm.sdc file generated by FSM Explorer. 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 289](#) 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 Log 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 Log watch window to compare selected criteria. Make sure to set the implementations you want to compare with the Configure Watch command. See [Using the Log Watch Window, on page 600](#) 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.

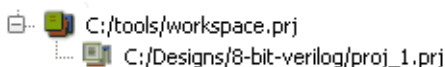
6. To delete an implementation, click the right mouse button on the implementation name in the project view, and select Remove Implementation from the popup menu.

## Creating Workspaces

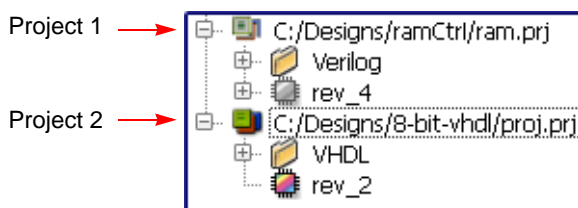
The Synplify Premier and Synplify Pro tools let you group projects together into workspaces. A workspace is like a container for a number of projects.

1. To create a new workspace, select File->New Workspace or right-click in the Project view and select Build Workspace.
2. In the dialog box,
  - Select the project files (.prj) of the projects you want to add to the workspace.
  - Name the workspace and click OK.

The Project view displays the workspace and the associated projects under it. The workspace file is also a .prj file.



3. To open more than one project in the same Project view, check Allow Multiple Projects to be Opened. After you set up the new project, you can see it in the Project view.



## Using Workspaces

You can use your workspace to simplify your work flow. For example, you can set up dependencies between projects in the same workspace. The Synplify software does not support workspaces.

1. To add a project to a workspace, right-click the workspace and select **Insert Project**. Select the project file you want to add, and click **OK**.
2. To remove a project from a workspace, right-click on the project and select **Remove Project from Workspace**.
3. To synthesize a single project in a workspace, click **Run**.

The software synthesizes the current project.

4. To run all the projects in a workspace, do the following:
  - If you have multiple implementations within a project, check that the correct implementation is active. To make an implementation active, click on the implementation in the **Project** view.
  - Select the workspace in the **Project** view, right-click, and select **Run all Projects**.

The software synthesizes the active implementations of all the projects in the workspace.

# 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 Timing Constraints, on page 219](#). For information about overriding global settings with individual attributes or directives, see [Entering Attributes and Directives, on page 304](#).

This section discusses the following topics:

- [Setting Device Options, on page 289](#)
- [Setting Optimization Options, on page 292](#)
- [Specifying Global Frequency and Constraint Files, on page 294](#)
- [Specifying Result Options, on page 296](#)
- [Specifying Timing Report Output, on page 297](#)
- [Setting Verilog and VHDL Options, on page 298](#)

## 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.

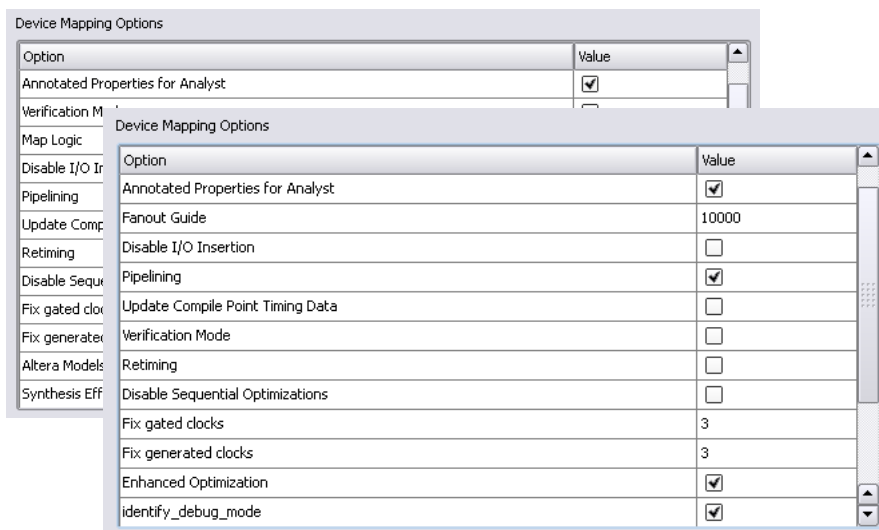
1. Open the Implementation Options form by clicking the Implementation Options button or selecting Project->Implementation Options, and click the Device tab at the top if it is not already selected.
2. Select the technology, part, package, and speed. Available options vary, depending on the technology you choose. Also, the Synplify Premier software does not support as many technologies as do the Synplify and Synplify Pro tools.

The image shows two examples of the 'Device' configuration window. The top window is for Xilinx Virtex4, with fields for Technology (Xilinx Virtex4), Part (XC4VLX15), Speed (-10), and Package (SF363). The bottom window is for Actel ProASIC3, with fields for Technology (Actel ProASIC3), Part (A3P060), and Speed (-2).

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 on 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 446](#), [Pipelining, on page 429](#), and [Retiming, on page 433](#), respectively. For details about other vendor-specific options, refer to the appropriate vendor chapter and technology family in the *Reference Manual*. Note that the Synplify tool does not support all these optimization options.

## Device Mapping Options Vary by Technology



4. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options](#), on page 289 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 will not apply to your design. 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 typical 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>   |

| Option                           | Tcl Command ( <code>set_option...</code> )     |
|----------------------------------|------------------------------------------------|
| Disable Sequential Optimizations | <code>-no_sequential_opt {1 0}</code>          |
| Enhanced Optimization            | <code>-enhanced_optimization {0 1}</code>      |
| Fanout Guide                     | <code>-fanout_guide <i>fanout_value</i></code> |
| Fix Gated Clocks                 | <code>-fixgatedclocks {0 1 2 3}</code>         |
| Fix Generated Clocks             | <code>-fixgeneratedclocks {0 1 2 3}</code>     |
| Package                          | <code>-package <i>pkg_name</i></code>          |
| Part                             | <code>-part <i>part_name</i></code>            |
| Pipelining                       | <code>-pipe {0 1}</code>                       |
| Retiming                         | <code>-retiming {0 1}</code>                   |
| Speed                            | <code>-speed_grade <i>speed_grade</i></code>   |
| Technology                       | <code>-technology <i>keyword</i></code>        |
| Update Compile Point Timing Data | <code>-update_models_cp {0 1}</code>           |
| Verification Mode                | <code>-verification_mode {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 grayed out. Setting the option in one place automatically updates it in the other. The Synplify software does not support all the options shown below.

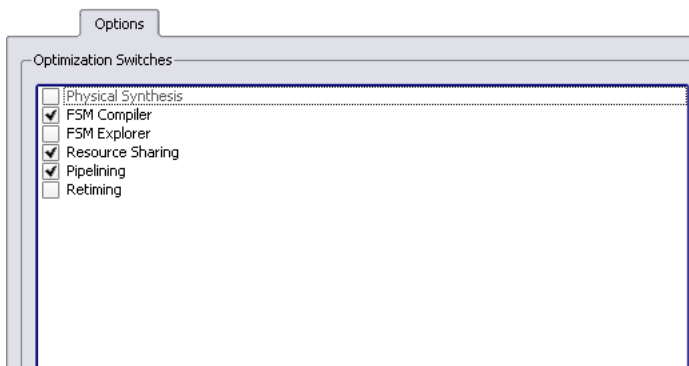


## Optimization Options

Project View

|                  |                                     |
|------------------|-------------------------------------|
| FSM Compiler     | <input checked="" type="checkbox"/> |
| FSM Explorer     | <input type="checkbox"/>            |
| Resource Sharing | <input checked="" type="checkbox"/> |
| Pipelining       | <input checked="" type="checkbox"/> |
| Retiming         | <input type="checkbox"/>            |

Implementation Options->Options



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

FSM Compiler      [Optimizing State Machines, on page 453](#)

FSM Explorer      [Running the FSM Explorer, on page 458](#)

*Note:* Only a subset of the Xilinx, Altera, and Actel technologies support the FSM Explorer option. Use the Project->Implementation Options->Options panel to determine if this option is supported for the device you specify in your tool.

Resource Sharing      [Sharing Resources, on page 450](#)

Pipelining      [Pipelining, on page 429](#)

Retiming      [Retiming, on page 433](#)

The equivalent Tcl set\_option command options are -symbolic\_fsm\_compiler, -use\_fsm\_explorer, -resource\_sharing, -pipe, and -retiming.

3. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 289](#) 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

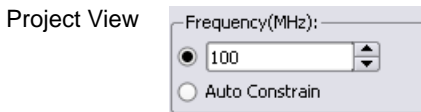
This procedure tells you how to set the global frequency and specify the constraint files for the implementation.

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.

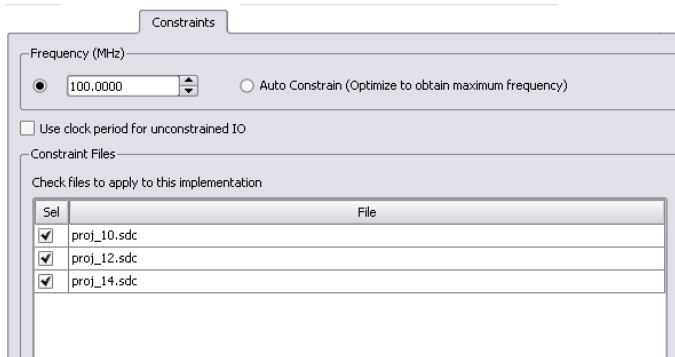
The equivalent Tcl set\_option commands is `-frequency frequency_value`.

You can override the global frequency with local constraints, as described in [Specifying Timing Constraints, on page 219](#). In the Synplify Pro tool, you can automatically generate clock constraints for your design instead of setting a global frequency. See [Using Auto Constraints, on page 251](#) for details.

### Global Frequency and Constraints



### Implementation Options->Constraints



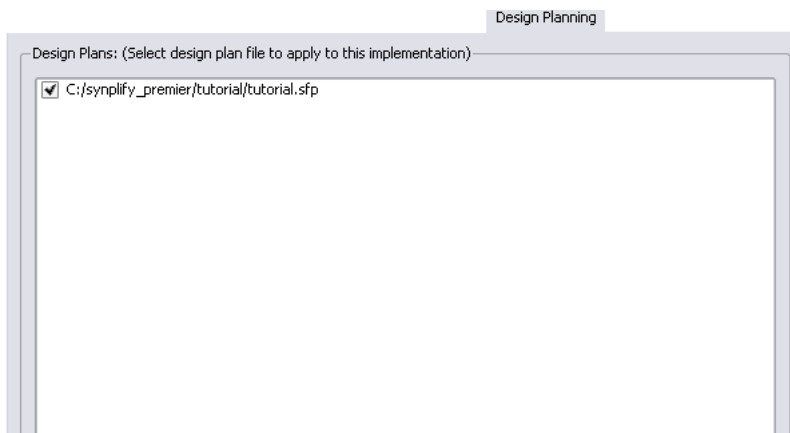
2. To specify constraint files for an implementation, do one of the following:
  - Select Project->Implementation Options->Constraints. Check the constraint (.sdc) files you want to use in the project.
  - With the implementation you want to use selected, click Add File in the Project view, and add the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

To create constraint files, see [Specifying Timing Constraints, on page 219](#).

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. To specify or remove a Synplify Premier design plan (.sfp), use the techniques described in steps 2 and 3, or do the following:
  - Select Project->Implementation Options->Design Planning. Check the box next to the file you want.
  - To delete a file, disable the check box next to the file name on the Design Planning tab.



When the implementation is synthesized, the Synplify Premier tool uses the region assignments in this file for the second phase of optimization to perform physical synthesis.

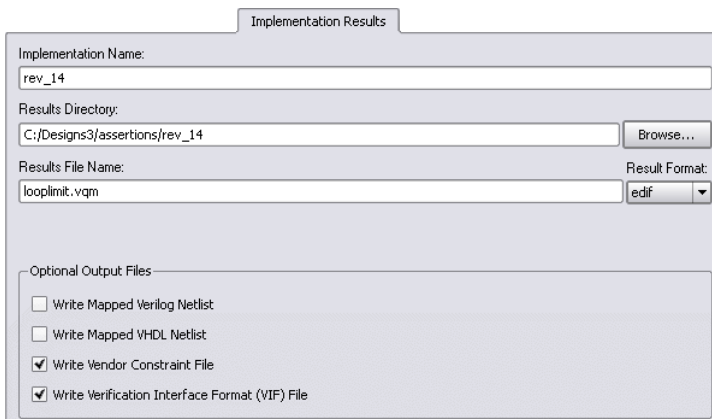
5. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 289](#) for a list of choices). Click OK.

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.



Implementation Results

Implementation Name: rev\_14

Results Directory: C:/Designs3/assertions/rev\_14 Browse...

Results File Name: looplimit.vqm Result Format: edif

Optional Output Files

- Write Mapped Verilog Netlist
- Write Mapped VHDL Netlist
- Write Vendor Constraint File
- Write Verification Interface Format (VIF) File

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. See [Generating Constraint Files for Forward Annotation, on page 105](#) for more information.
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*.

For certain Altera technologies (see [Generating Vendor-Specific Output, on page 836](#)), the .vqm result format allows you to also select the version of Quartus II you are using from the pop-up menu.



5. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 289](#) 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.

In the Synplify Premier tool, you can also use this tab to generate hierarchical-based island timing reports for certain technologies like Xilinx Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, Spartan-3, and Altera Stratix technologies. For more information about generating island timing reports, see [Generating the Island Timing Report Automatically, on page 745](#) and [Generating the Island Timing Report Interactively, on page 747](#).

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



The image shows a dialog box titled "Timing Report". It contains two input fields. The first field is labeled "Number of Critical Paths:" and contains the value "32". The second field is labeled "Number of Start/End Points:" and contains the value "8".

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 289](#) for a list of choices). Click OK.

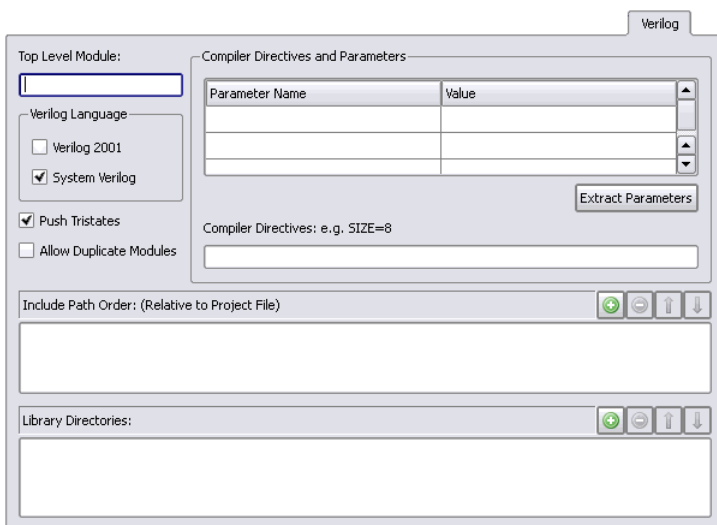
When you synthesize the design, the software compiles and maps the design using the options you set.

## 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.

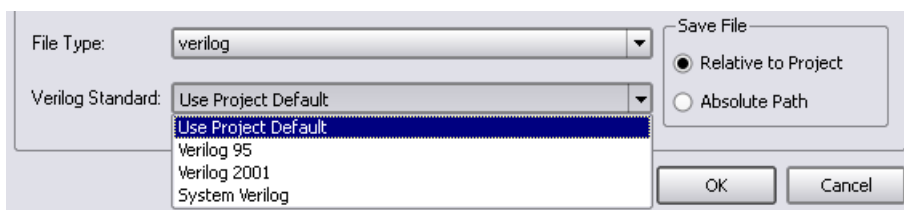
### Setting Verilog File Options

You set Verilog file options by selecting either Project->Implementation Options->Verilog, or Options->Configure Verilog Compiler. For information about creating always block hierarchy for Synplify Premier, see [Setting Synplify Premier Netlist Restructuring Optimizations, on page 330](#).



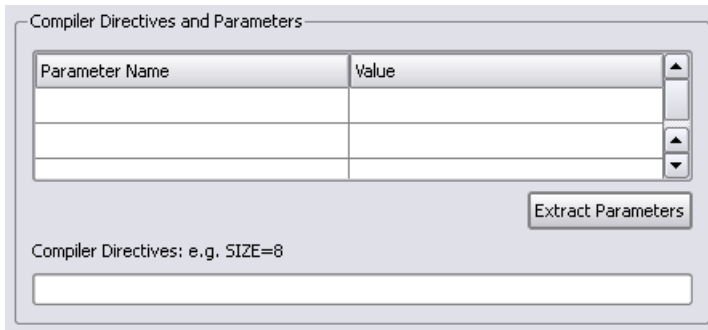
1. Specify the Verilog format to use.

- 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 *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 is Verilog 2001.



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.

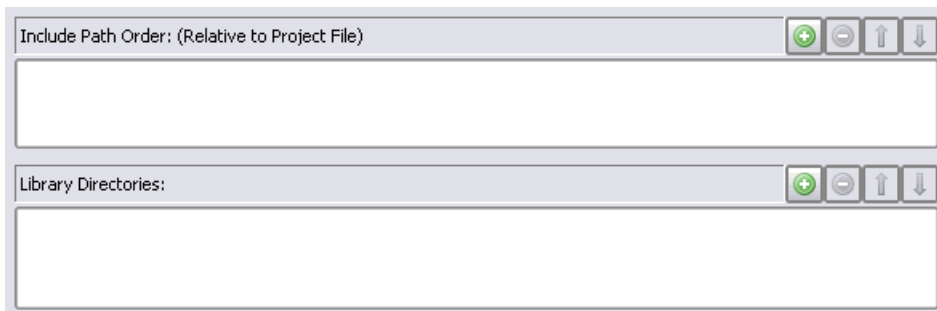
The software uses the new value for the current implementation only. Note that parameter extraction is not supported for mixed designs.



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, `SIZE=32 TEST_IMPL` results in the software writing the following statements to the project file:

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



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, 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.
7. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 289](#) 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.

| Generic Name | Value |
|--------------|-------|
|              |       |
|              |       |
|              |       |
|              |       |
|              |       |
|              |       |

For VHDL source, you can specify the options described below. For information about creating process hierarchy for Synplify Premier, see [Setting Synplify Premier Netlist Restructuring Optimizations](#), on page 330.

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 148.

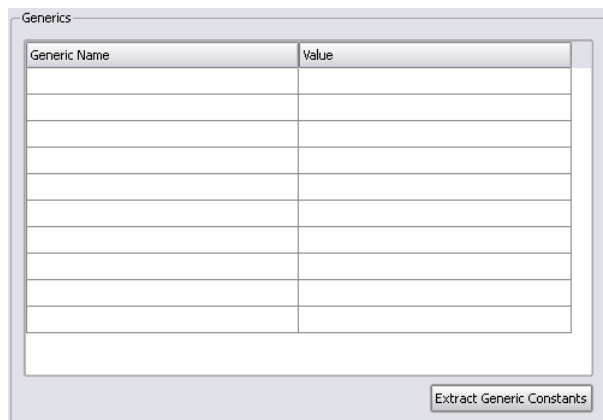
You can also use this option for mixed language designs or when you want to specify a module that is not the actual top-level entity for HDL Analyst displaying and debugging in the schematic views.

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 369.

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 153 in the *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 Directive](#), on page 1151 in the *Reference Manual* for more information.

6. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 289](#) for a list of choices). Click OK.

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

## Entering Attributes and Directives

Attributes and directives are pieces of information that you attach to design objects to control the way in which your design is analyzed, optimized, and mapped. For further details, refer to these subtopics:

- [Specifying Attributes and Directives](#), on page 304
- [Specifying Attributes and Directives in VHDL](#), on page 305
- [Specifying Attributes and Directives in Verilog](#), on page 307
- [Specifying Attributes Using the SCOPE Editor](#), on page 308
- [Specifying Attributes in the Constraints File \(.sdc\)](#), on page 310

## Specifying Attributes and Directives

Attributes control mapping optimizations and directives control compiler optimizations. Because of this difference, you must specify directives in the source code. However, attributes can be added to the constraint file or the source code.

### HDL Source Code

This is the only way to specify directives. You can also specify attributes in the source code, but if you do so, you must recompile the design whenever you change an attribute value. This can be very time-consuming for large designs, so the preferred way is to specify attributes in the constraint file, as described above.

For information about procedures to add attributes and directives, see the following:

- [Specifying Attributes and Directives in VHDL](#), on page 305
- [Specifying Attributes and Directives in Verilog](#), on page 307

### Constraint File (.sdc)

This is the preferred method for specifying attributes because it is more flexible; you do not have to recompile the design. You can add attributes to a constraint file using one of the ways described here:

| Enter them...                           | Description                                                  | Details                                                                                                                                                                                                                                                             |
|-----------------------------------------|--------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| In the SCOPE Attributes tab             | A graphical interface for generating or editing an .sdc file | <ul style="list-style-type: none"> <li>• <a href="#">Specifying Attributes Using the SCOPE Editor, on page 308</a></li> <li>• <a href="#">How Attributes and Directives are Specified, on page 896 of the Reference Manual</a></li> </ul>                           |
| By manually editing the constraint file | Tcl constraint file (.sdc)                                   | <ul style="list-style-type: none"> <li>• <a href="#">Specifying Attributes in the Constraints File (.sdc), on page 310</a></li> <li>• Syntax for individual attributes is included in the descriptions of the attributes in the <i>Reference Manual</i>.</li> </ul> |

## Specifying Attributes and Directives in VHDL

You can use other methods to add attributes to objects, as listed in [Entering Attributes and Directives, on page 304](#). 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 881](#) 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 `product_installation_dir/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** *object\_name* : *object\_kind* **is** *value* ;

For example:

```
entity simpledff is
 port (q: out bit_vector(7 downto 0);
 d : in bit_vector(7 downto 0);
 clk : in bit);
 attribute syn_noclockbuf of clk : signal is true;
```

For details of the syntax conventions, see [VHDL Attribute and Directive Syntax, on page 881](#) 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** *attribute\_name* : *data\_type* ;**attribute** *attribute\_name* **of** *object\_name* : *object\_kind* **is** *value* ;

For example:

```
entity simpledff is
 port (q: out bit_vector(7 downto 0);
 d : in bit_vector(7 downto 0);
 clk : in bit);
 attribute syn_noclockbuf : boolean;
 attribute syn_noclockbuf of clk :signal is true;
```

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 [Entering Attributes and Directives, on page 304](#). 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 717](#) 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 attribute_name = value */
/* synthesis directory_name = value */
```

### Verilog Line Comment Syntax

```
// synthesis attribute_name = value
// synthesis directory_name = value
```

For details of the syntax rules, see [Verilog Attribute and Directive Syntax, on page 717](#) in the *Reference Manual*. The following are examples:

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

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 */;
```

## 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 305](#) or [Specifying Attributes and Directives in Verilog, on page 307](#)). 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).

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:<module_name>` in Verilog, or `v:<library>.<module_name>` 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.

4. 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.



|   | Enabled                             | Object Type | Object   | Attribute               | Value | Val Type | Description | mme |
|---|-------------------------------------|-------------|----------|-------------------------|-------|----------|-------------|-----|
| 1 | <input checked="" type="checkbox"/> | output_port | <global> | syn_noclockbuf          |       |          |             |     |
| 2 | <input checked="" type="checkbox"/> |             |          | syn_clean_reset         |       |          |             |     |
| 3 | <input checked="" type="checkbox"/> |             |          | syn_dspstyle            |       |          |             |     |
| 4 | <input checked="" type="checkbox"/> |             |          | syn_edif_bit_format     |       |          |             |     |
| 5 | <input checked="" type="checkbox"/> |             |          | syn_edif_scalar_format  |       |          |             |     |
|   |                                     |             |          | syn_forwar...onstraints |       |          |             |     |
|   |                                     |             |          | syn_multstyle           |       |          |             |     |
|   |                                     |             |          | syn_netlist_hierarchy   |       |          |             |     |
|   |                                     |             |          | syn_noarrayports        |       |          |             |     |
|   |                                     |             |          | syn_noclockbuf          |       |          |             |     |
|   |                                     |             |          | syn_ramstyle            |       |          |             |     |

Attributes

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.

5. 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.

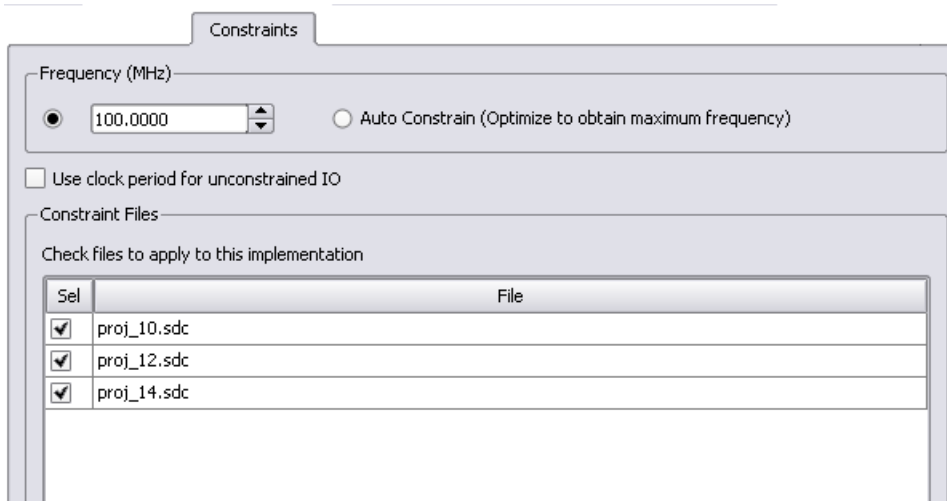
If you manually type an attribute the software does not recognize, or select an incompatible attribute/object combination, the attribute cell is shaded in red.

6. 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 896](#) of the *Reference Manual* for the syntax description.

7. 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 (.sdc)

When you use the SCOPE window ([Specifying Attributes Using the SCOPE Editor, on page 308](#)), the attributes are automatically written to the .sdc constraint file using the Tcl `define_attribute` syntax. The following procedure explains how to specify attributes directly in the constraint file. For information about editing constraints in the file, see [Using a Text Editor for Constraint Files, on page 100](#).

1. In the .sdc constraint file, enter the attribute and specify the value you want, using the `define_attribute` syntax. For example,

```
define_attribute {object_name} attribute_name value
```

Check the attribute descriptions in the *Reference Manual* for the exact syntax and values of the attribute.

The following code excerpt provides an example of attributes defined in the .sdc file. (Some of these attributes are specific to Xilinx devices):

```
Assign a location for scalar port "sel".
 define_attribute {sel} xc_loc "P139"

Assign a pad location to all bits of a bus.
 define_attribute {b[7:0]} xc_loc "P14, P12, P11, P5, P21,
 P18, P16, P15"

Assign a fast output type to the pad.
 define_attribute {a[5]} xc_fast 1

Use a regular buffer instead of a clock buffer for clock "clk_slow".
 define_attribute {clk_slow} syn_noclockbuf 1

Relax timing by not buffering "clk_slow", because it is the slow clock
Set the maximum fanout to 10000.
 define_attribute {clk_slow} syn_maxfan 10000
```

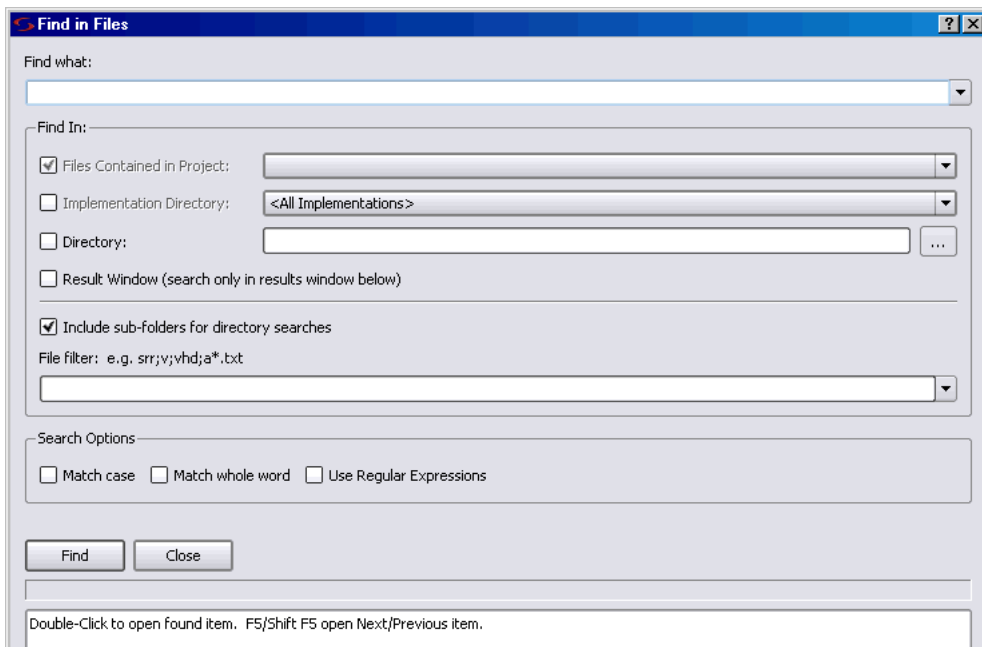
# Searching Files

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 crossprobe 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 subpattern.

## 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, sdc 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 on 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](#)

### Archive a Project

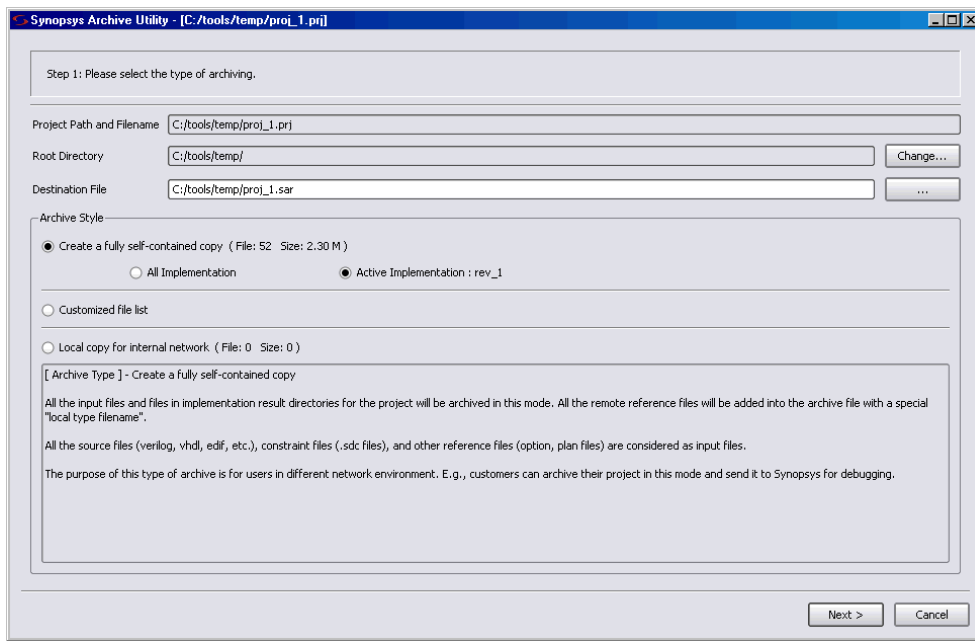
Use the archive utility to store the files for a design project into a single archive file in a proprietary format (.sar). You can archive an entire project or selected files from a project. If you want to create a copy of a project without archiving the files, see [Copy a Project, on page 323](#).

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 1215](#) of the *Reference Manual*.

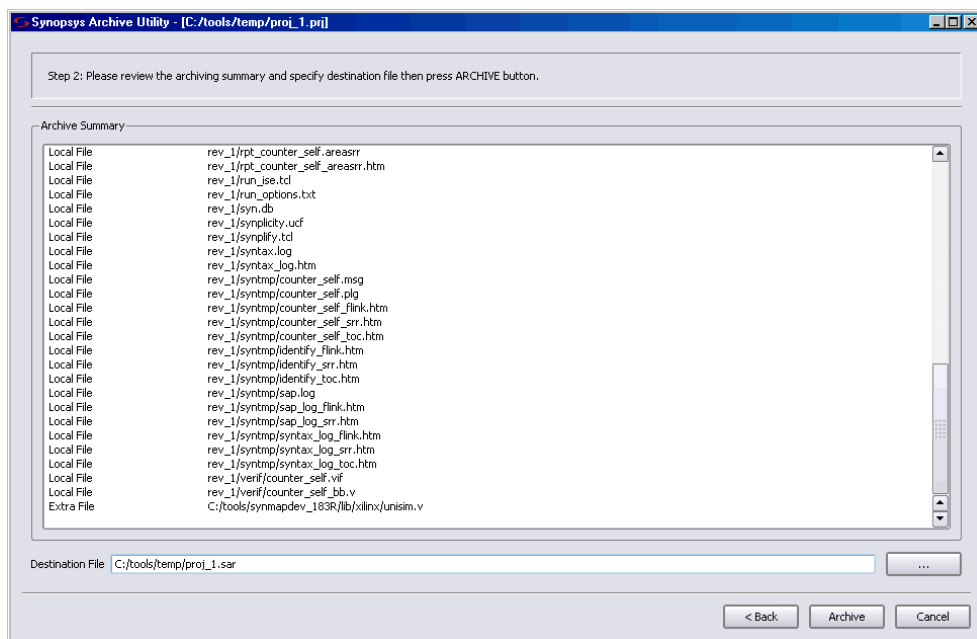
The archive utility automatically runs a syntax check on the active project (Run->Syntax Check command) to ensure that a complete list of project files is generated. If you have Verilog 'include files in your project, the utility includes the complete list of Verilog files. It also checks the syntax 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:

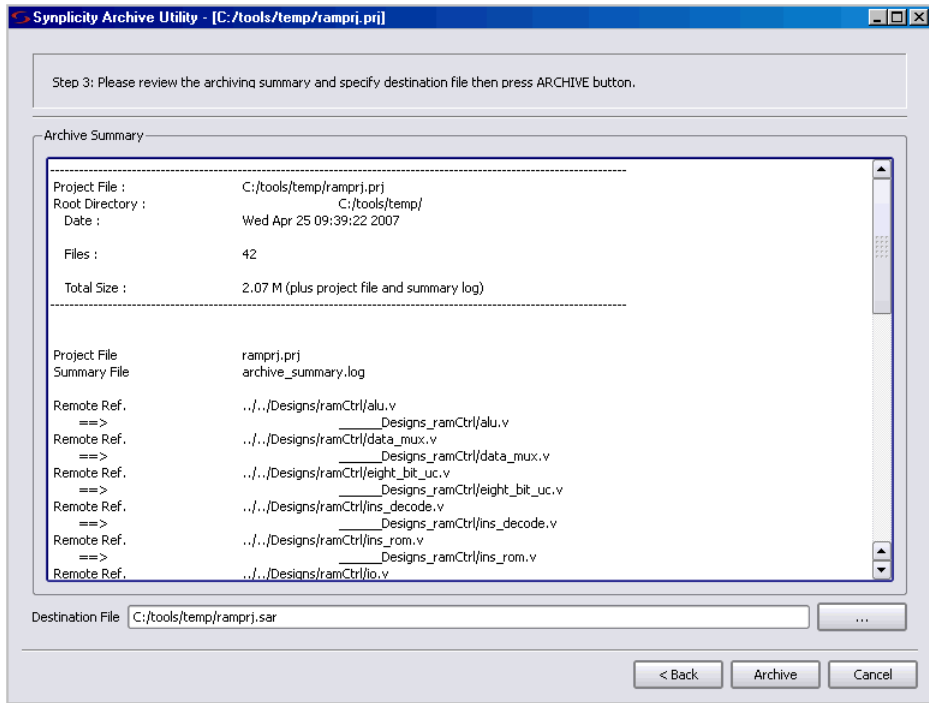
- Fill in Destination File with a location for the archive file.
- Set Archive Style. You can archive all the project files with all the implementations or selectively archive files and implementations
- To archive only the active implementation, enable Active Implementation.
- To selectively archive files, enable Customized file list, and use the check boxes to include files in or exclude files from the archive. Use the Add Extra Files button on the second page to include additional files in the project.



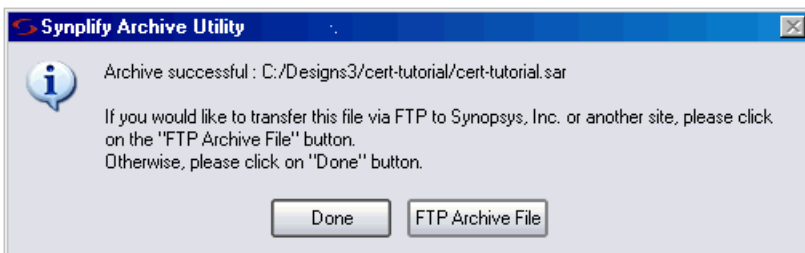


- Click Next.

The tool summary displays all the files in the archive and shows the full uncompressed file size. The actual size is smaller after the archiving operation as there is no duplication of files.



3. Use the Back button to correct directory or file information and/or follow-up on any missing files, as appropriate.
4. Verify that the current archive contains the files that you want, then click Archive which creates the project archive .sar file and displays the following prompt:

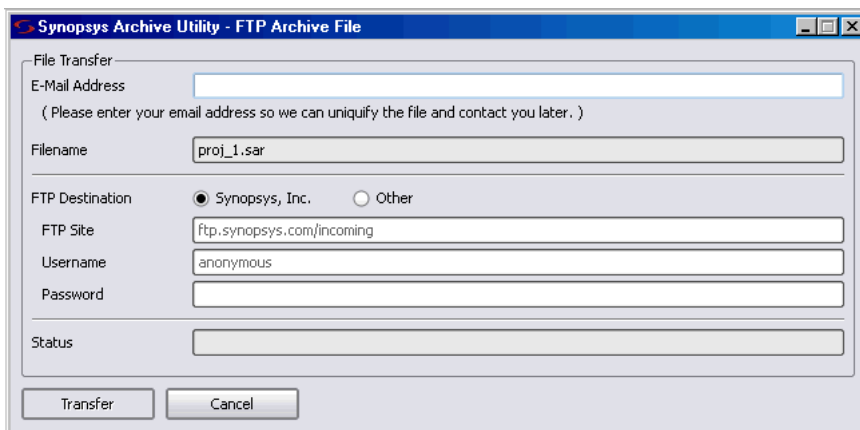


5. Click Done if you are finished.

If you want to send the archive to another site, go on to the next step. For example, you can send the design project to the Synopsys FTP site.

6. To send the archive to another site, do the following:

- Click FTP Archive File.



The screenshot shows a dialog box titled "Synopsys Archive Utility - FTP Archive File". It contains the following fields and controls:

- File Transfer** section:
- E-Mail Address**: A text input field with a placeholder text "( Please enter your email address so we can unifiy the file and contact you later. )".
- Filename**: A text input field containing "proj\_1.sar".
- FTP Destination**: Two radio buttons, "Synopsys, Inc." (selected) and "Other".
- FTP Site**: A text input field containing "ftp.synopsys.com/incoming".
- Username**: A text input field containing "anonymous".
- Password**: An empty text input field.
- Status**: An empty text input field.
- At the bottom, there are two buttons: "Transfer" and "Cancel".

- Fill in your email address. At the Synopsys web site, this email address, plus a date and time stamp are prepended to the .sar file name to uniquely identify your archive file.
- Fill in the other details about the FTP site destination, including username and password if you are sending it to sites other than the Synopsys one.

7. Click Transfer.

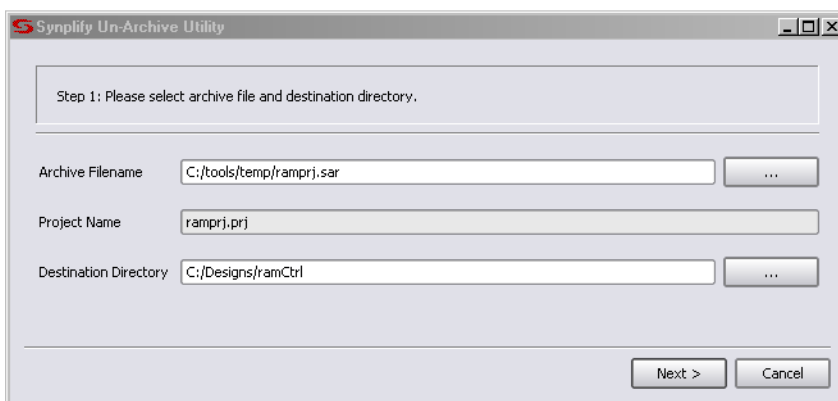
This completes the archive transfer. The Status field in the dialog box displays the transfer results.

## Un-Archive a Project

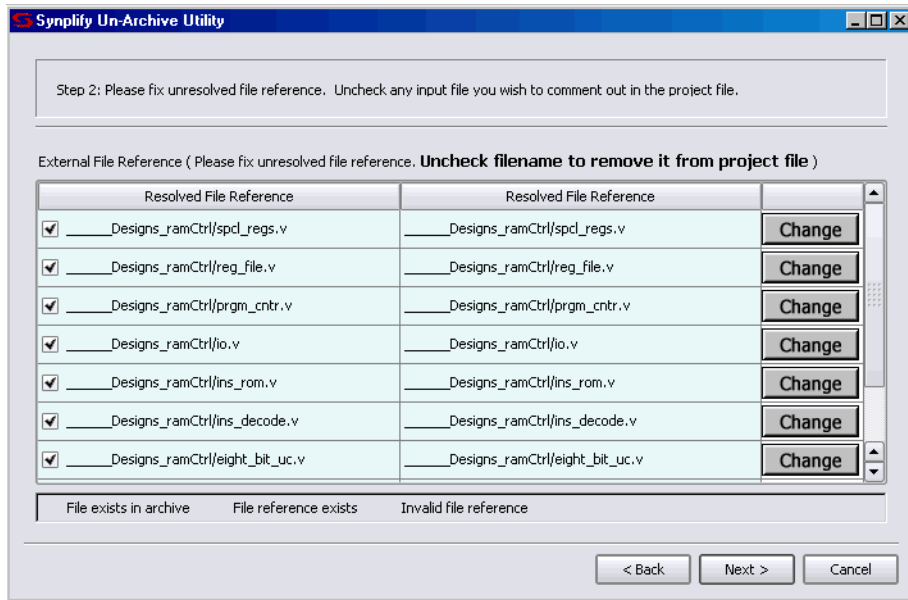
Use this procedure to extract design project files from an archive file (.sar).

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

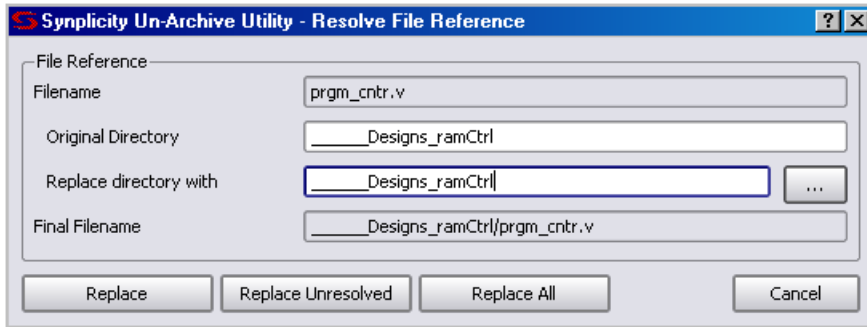
The Tcl command equivalent is project **-unarchive**. For a complete description of the project Tcl command options for archiving, see [project](#), on page 1215 of the *Reference Manual*.



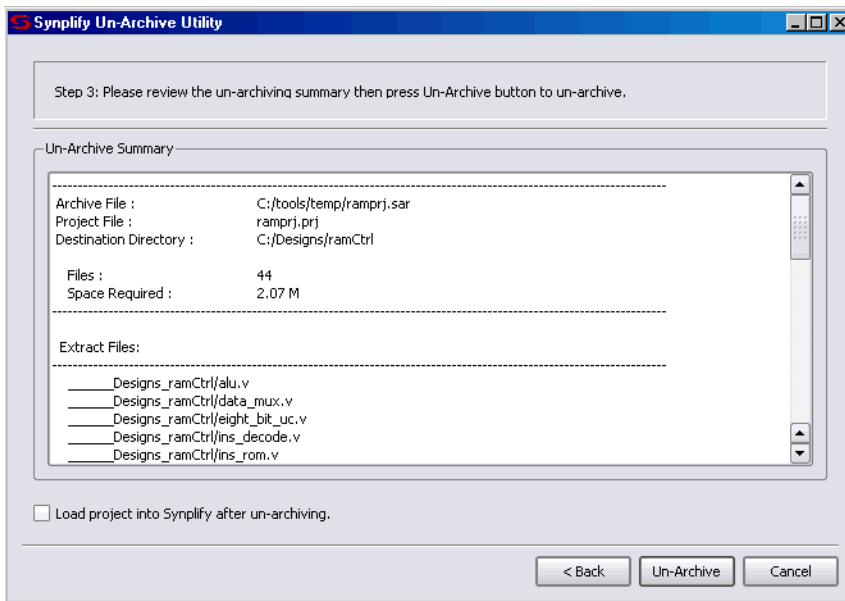
2. In the wizard, enter the following:
  - Name of the .sar file containing the project files.
  - Name of project to extract (un-archive). This field is automatically extracted from the .sar file and cannot be changed.
  - Pathname of directory in which to write the project files (destination).
  - 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.



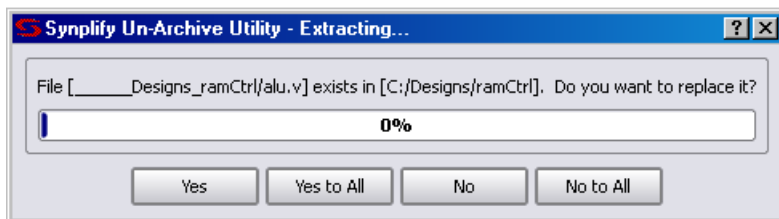
- Click Next and verify that the project files you want are displayed in the Un-Archive Summary.



- If you want to load this project in the UI after files have been extracted, enable the Load project into Synplicity after un-archiving option.
- Click Un-Archive.

A message dialog box is displayed while the files are being extracted.

7. 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

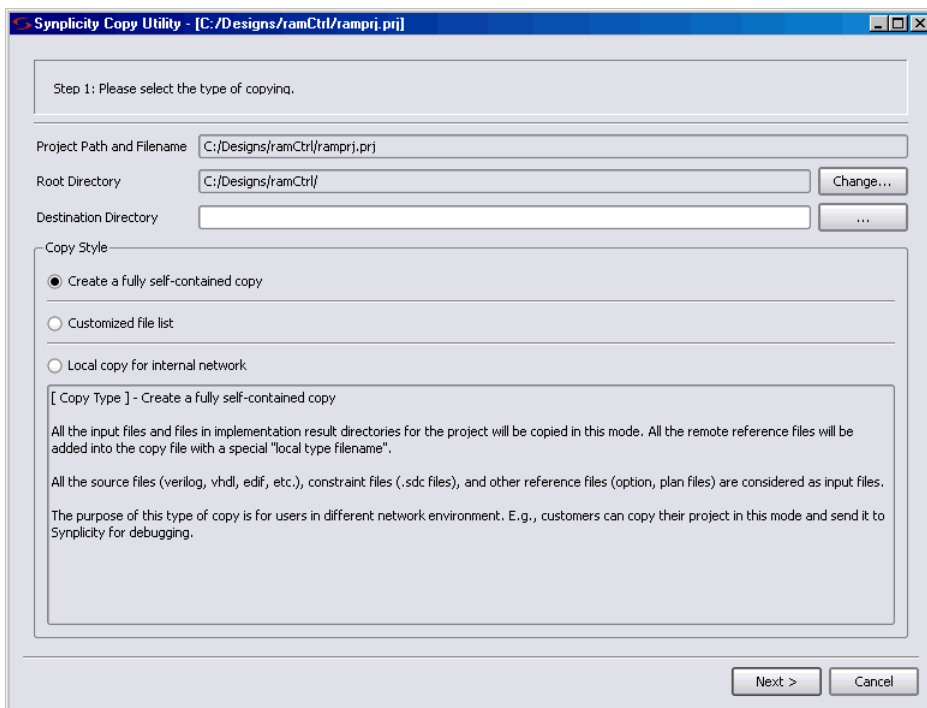
Use this utility to create an unarchived copy of a design project. You can copy an entire project or just selected files from the project. However, if you want to create an archive of the project, where the entire project is stored as a single file, see [Archive a Project, on page 315](#).

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 1215](#) of the *Reference Manual*.

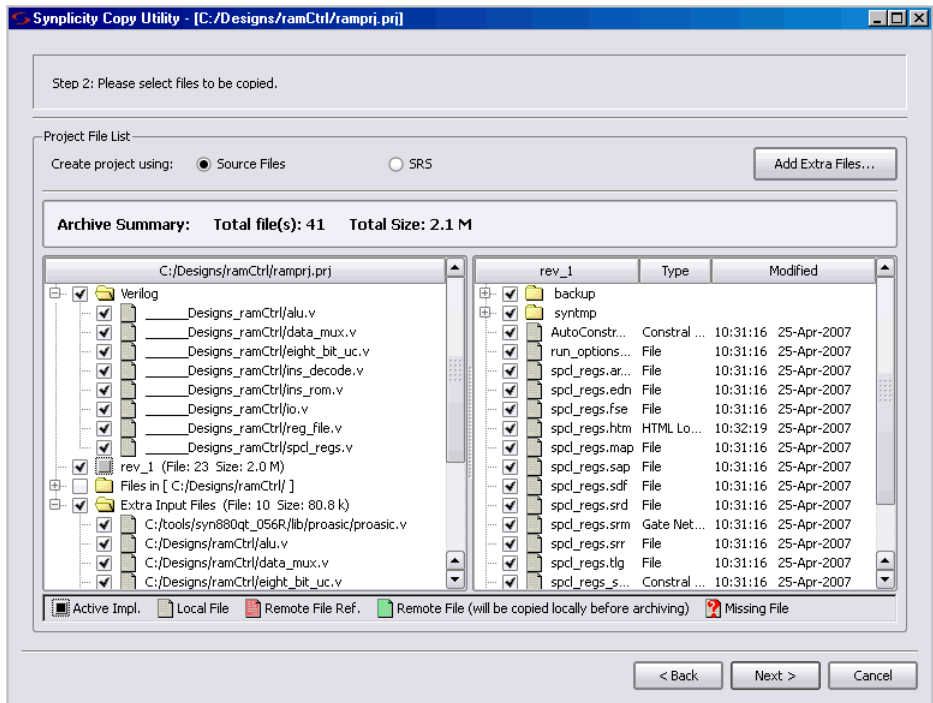
This command automatically runs a syntax check on the active project (Run->Syntax Check command) to ensure that a complete list of project files is generated. If you have Verilog include files in your project, they are included. The utility runs this check for each implementation in the project to ensure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



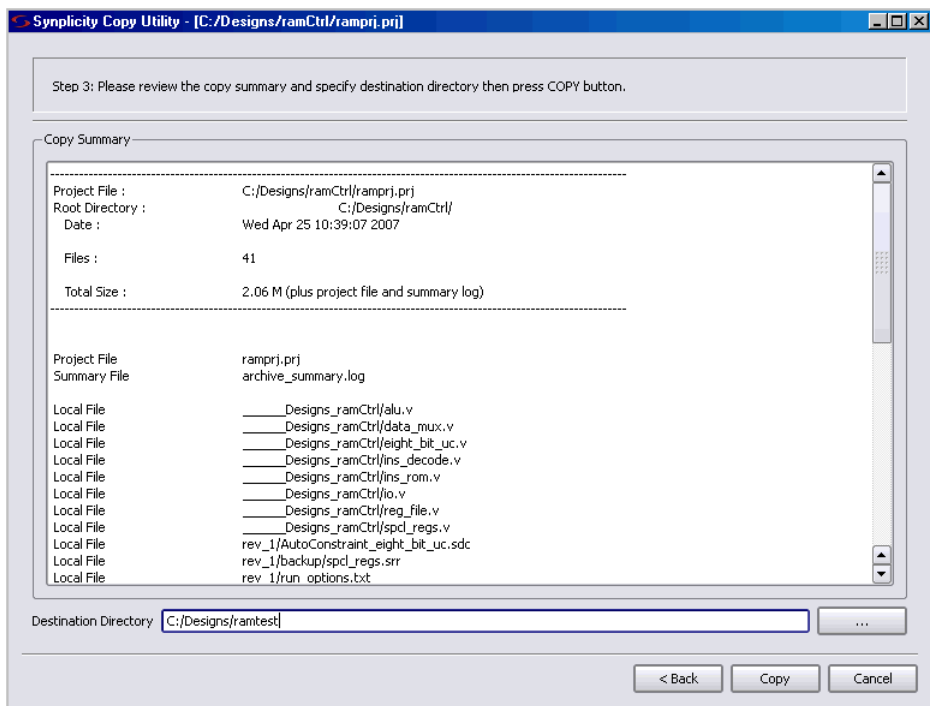
## 2. Do the following in the wizard

- 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.



#### Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA  
 Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Conforma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 7

# Setting up a Physical Synthesis Project

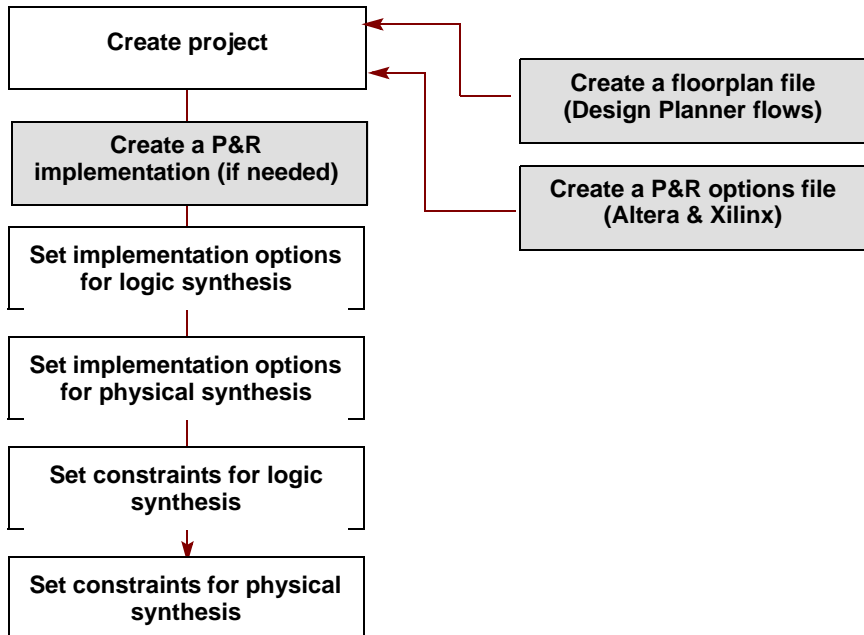
---

The process to set up a physical synthesis project is the same as for a logic synthesis project, but it requires a few additional steps. The following describe the additional steps needed to set up a physical synthesis project:

- [Setting up for Physical Synthesis](#), on page 328
- [Setting Options for Physical Synthesis](#), on page 330
- [Setting Constraints for Physical Synthesis](#), on page 347
- [Forward-Annotating Physical Synthesis Constraints](#), on page 351
- [Backannotating Physical Synthesis Constraints](#), on page 353

## Setting up for Physical Synthesis

This figure summarizes the steps for setting up a physical synthesis project. The shaded boxes are either optional steps, or steps that are specific to a flow or certain technologies. Although the figure makes a distinction between setting options or constraints for logic synthesis and physical synthesis, you can actually define both types of options at the same time.



See the following for details about physical synthesis setup:

| For information about... | See...                                                                   |
|--------------------------|--------------------------------------------------------------------------|
| Implementation options   | <a href="#">Setting Options for Physical Synthesis</a> , on page 330     |
| Constraints              | <a href="#">Setting Constraints for Physical Synthesis</a> , on page 347 |

| <b>For information about...</b> | <b>See...</b>                                                                                                                                                    |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Creating floorplan files        | <a href="#">Using Design Planner Floorplan Constraints</a> , on page 347                                                                                         |
| Creating P&R implementations    | <a href="#">Creating a Place and Route Implementation</a> , on page 332                                                                                          |
| Creating P&R options files      | <a href="#">Specifying Altera Place-and-Route Options</a> , on page 337<br><a href="#">Specifying Xilinx Place-and-Route Options in a Tcl File</a> , on page 340 |

---

## Setting Options for Physical Synthesis

After you have set up logic synthesis options for the implementation with the Implementation Options command (see [Setting Options for Physical Synthesis, on page 330](#)), you can set other options specific to physical synthesis.

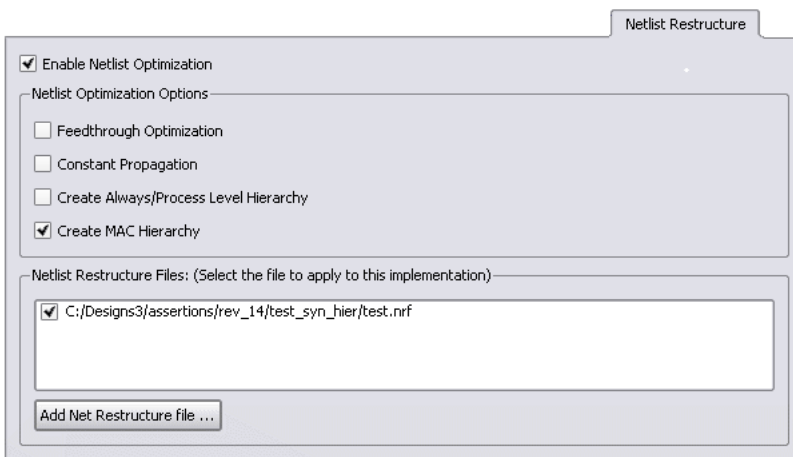
The following describe them :

- [Setting Synplify Premier Netlist Restructuring Optimizations, on page 330](#)
- [Creating a Place and Route Implementation, on page 332](#)
- [Specifying Altera Place-and-Route Options, on page 337](#)
- [Specifying Xilinx Place-and-Route Options in a Tcl File, on page 340](#)
- [Specifying Xilinx Place-and-Route Options in an .opt File, on page 341](#)
- [Specifying Xilinx Global Placement Options, on page 346](#)

### Setting Synplify Premier Netlist Restructuring Optimizations

You can set these options if you have zippered or bit-sliced your design or for other restructuring operations. The netlist restructuring options are only available in the Synplify Premier tool.

1. Select Project->Implementation Options, and click on the Netlist Restructure tab.



2. To reduce the number of ports, eliminate feedthrough ports by enabling Feedthrough Optimization. This can improve routability in the place-and-route tool.
3. To reduce area, enable Constant Propagation.

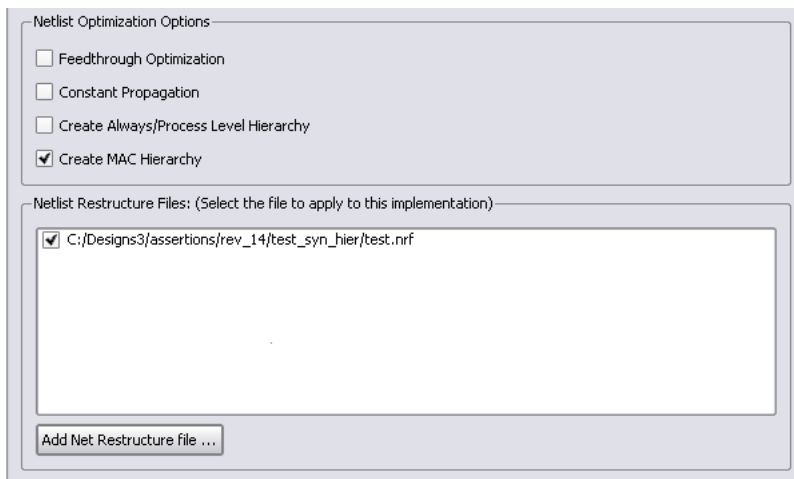
Where possible, this option eliminates the logic used when constant inputs to logic cause their outputs to be constant. It is sometimes possible to eliminate this type of logic altogether during optimization.

4. To provide more granularity for applying a design plan to large modules at the always block or process level, enable Create Always/Process Level Hierarchy.

Currently a design plan can be applied to either modules or to individual gates, registers, and so on. For a module that is too large to fit in a row or defined region, you might need an extra level of granularity which is not as detailed as a gate-level description. This option creates an additional, intermediate level of hierarchy to which you can apply a design plan.

For example, in Verilog, the always block becomes a module with the signals in the sensitivity list becoming inputs of the module and the signals that get their values set becoming outputs of the modules. Similarly, in VHDL, a process becomes a module. You might find that it is easier to apply a design plan to these always blocks/processes.

5. To group Altera Stratix MAC configurations together into one MAC block, enable Create MAC Hierarchy.
6. To add or delete netlist restructure files, such as the files created for bit-slicing or zippering, do the following:
  - On the Project->Implementation Options->Netlist Restructuring tab, check the box next to the file you want to add.
  - To remove a file, disable the check box next to the file name.



You can add or delete the files from the Project view. When the implementation is synthesized, the Synplify Premier tool uses the specified netlist restructure files for physical synthesis.

7. Set other implementation options as needed (see [Setting Logic Synthesis Implementation Options, on page 289](#)). Click OK.

## Creating a Place and Route Implementation

For Altera, Actel, and Xilinx technologies, the Synplify Pro and Synplify Premier tools automatically create a place-and-route implementation after synthesis completes. The following steps show you how to manually create a place-and-route implementation.

1. Make sure you have the correct version of the P&R tool installed, and that all variables for the tool have been set.
  - Check the release notes for version information. Select Help->Online Documents->release\_notes.pdf and go to *Third Party Tool Versions*.
  - For Actel technologies, set the Actel ALSDIR and PATH environment variables to point to a valid installation of the place and route tool.
  - For Altera technologies, set the QUARTUS\_ROOTDIR and PATH environment variables to point to a valid installation of the place and route tool.

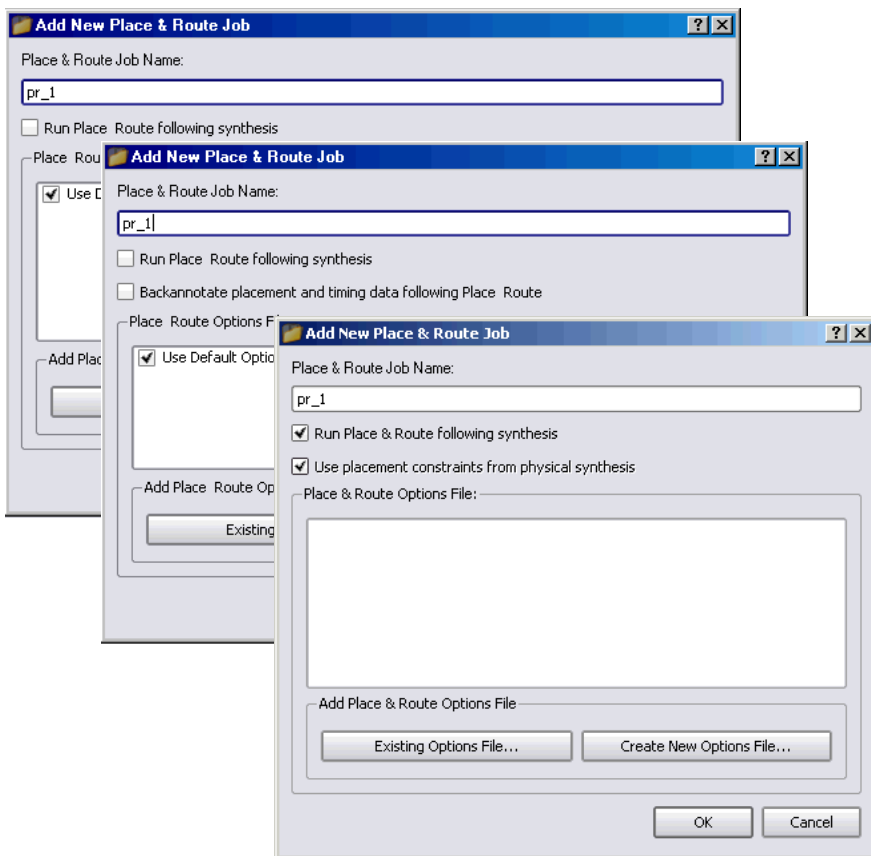


- For Xilinx technologies, set the XILINX and PATH environment variables to point to a valid installation of the place and route tool.

The Synplify Premier UI automatically sets the PAR\_BELDLYRPT environment variable to 1. This environment variable is set so that the Xilinx place-and-route placement file (.xdl) is generated with a particular format. The software uses this .xdl file to backannotate the placement information

2. To create a place-and-route implementation, do one of the following:
  - Click on the Add P& R Implementation button from the Project view.
  - Select an implementation in the Project view, then right-click and select Add New Place & Route Job.

The Add New Place and Route Job dialog box opens. The available options differ slightly, depending on the technology.



3. Set the options you need. Available place-and-route options vary depending on the synthesis tool and technology.
  - Specify the Place & Route Job Name. The default is `pr_n`. Avoid using spaces in the implementation name.
  - Enable Run Place and Route following synthesis.
  - Select a place-and-route options file. If you do not specify one, the tool uses the settings in the default file.

---

|        |                                         |
|--------|-----------------------------------------|
| Actel  | <install_dir>\lib\Actel\Actel_par.opt   |
| Altera | <install_dir>\lib\Altera\Altera_par.tcl |
| Xilinx | <install_dir>\lib\Xilinx\Xilinx_par.opt |

---

For Altera and Xilinx designs, refer to the information in [Specifying Altera Place-and-Route Options, on page 337](#) or [Specifying Xilinx Place-and-Route Options in a Tcl File, on page 340](#) for details. For Xilinx designs, you can also override global placement options with an environment variable, as described in [Specifying Xilinx Global Placement Options, on page 346](#).

- If you are going to run Synplify Premier physical synthesis, you can choose to backannotate data for certain Altera and Xilinx technologies. See [Backannotating Place-and-Route Data, on page 353](#) for details.
  - For Xilinx designs, you can automatically generate a coreloc file with backannotated data after place-and-route. See [Generating a Xilinx Coreloc Placement File, on page 354](#) for more information.
  - For physical synthesis with Altera Stratix III, Stratix II GX, or Stratix II devices you can select the Use placement constraints from physical synthesis. See [Forward Annotating Altera Physical Constraints, on page 351](#).
4. Enable the Run Place & Route following synthesis option. Click OK if you are not setting the options described in the next step.

This creates the place-and-route implementation under the current synthesis implementation. Currently, you cannot change the location of the P&R directory.

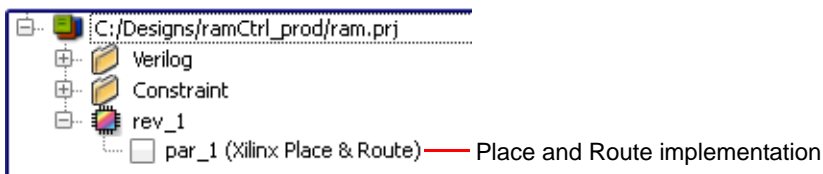
Conversely, if you do not want to create a place-and-route implementation, disable the Run Place & Route following synthesis option.

5. For Xilinx and Altera technologies, you can also do the following:
- Specify a place-and-route options file. If you do not specify one, the tool automatically uses the default options in these files:

```
<install_directory>\lib\altera\altera_par.tcl
<install_directory>\lib\xilinx\xilinx_par.tcl
```

You can change or override the default options. See [Specifying Altera Place-and-Route Options](#), on page 337 and [Specifying Xilinx Place-and-Route Options in a Tcl File](#), on page 340 for details.

- Backannotate constraints to the P&R tool. See [Backannotating Place-and-Route Data](#), on page 353 for details.
  - Forward-annotate constraints from the P&R tool. See [Forward Annotating Altera Physical Constraints](#), on page 351.
  - Click OK.
6. Select the implementation in the Project view to see the place-and-route implementation.



To create subsequent place-and-route implementations, select the place-and-route implementation, right-click, and select Add Place & Route Job. You can repeat the preceding steps to add as many P&R implementations as you need.

7. Synthesize the design.
- Enable the P&R implementation you want to use, if you have not already done so (Implementation Options->Place and Route tab).



- Click the Run button, or right-click in the Project view and select Run Place & Route Job from the popup menu.

If the synthesis implementation associated with the place-and-route implementation has not been synthesized, Run Place & Route Job invokes synthesis as well. After synthesis, the place-and-route tool is automatically run. If you have a Xilinx design and have specified an options file, the software uses these options during place-and-route.

8. To run in batch mode, do this:
  - Create a place-and-route implementation, as described previously.
  - Use the `-run all` command. If the synthesis implementation is selected the software only runs synthesis; you must run place-and-route separately. Otherwise, make the current implementation the place-and-route implementation before issuing the batch command.

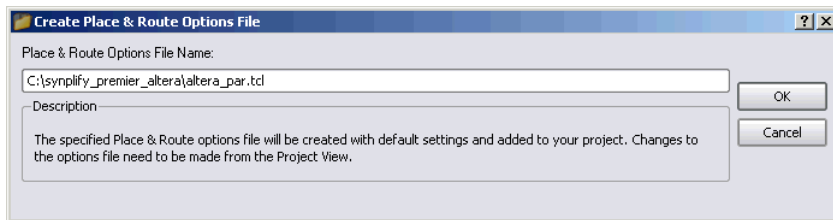
## Specifying Altera Place-and-Route Options

This section shows you how to customize your Altera place-and-route run by specifying a place-and-route options file or `.tcl` script. You can use either the default file or create a custom file.

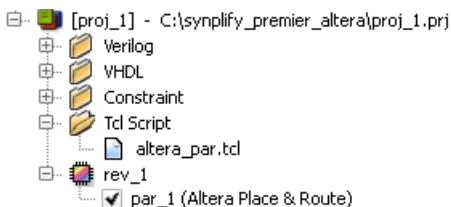
1. To use the default place-and-route options, click the Add P&R Implementation button in the Project view and select Use Default Options File in the dialog box. Click OK.

The software uses the options in the `altera_par.tcl` file which is located in the installation directory.

2. To use an existing options file (.tcl script):
  - Click the Add P&R Implementation button in the Project view.
  - Click Existing Options File. Select the file name in the next dialog box, and click Open.
  - Return to the Add New Place & Route Job dialog box and make sure the correct options file is selected. Click OK.
3. To create a new place-and-route options file:
  - Click the Add P&R Implementation button in the Project view. In the dialog box, click Create New Options File. Specify the file name in the next dialog box, and click OK.



A text window opens with the default options file. This file is automatically added to the project.



- Edit the default options to customize this options file. For more information about the contents of this file, see [Options in the Altera Place-and-Route Options File](#), on page 339.

- Save the file.
- Return to the Add New Place & Route Job dialog box, and make sure the options file you created is selected.
- Select Run Place & Route following synthesis. Click OK.

The software uses the options file to place and route the design after synthesis.

#### 4. View the results.

- Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
- View the log file quartus.log for information about the run.

## Options in the Altera Place-and-Route Options File

To customize the Altera place-and-route options file, you can edit the default options file (`altera_par.tcl`). This file contains the options for the following place-and-route processes:

- [Fitter Options](#)
- [Timing Analyzer Options](#)
- [Analysis & Synthesis Options](#)

### Fitter Options

Edit the following default fitter options for the Quartus process shown below.

```
#Fitter Options
set_global_assignment -name OPTIMIZE_HOLD_TIMING "IO PATHS AND MINIMUM TPD PATHS"
set_global_assignment -name OPTIMIZE_FAST_CORNER_TIMING OFF
set_global_assignment -name OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING ON
set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_RETIMING OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT NORMAL
set_global_assignment -name FITTER_EFFORT "AUTO FIT"
set_global_assignment -name SEED 1
```

### Timing Analyzer Options

Edit the following default timing analyzer options for the Quartus process shown below.

```
#Timing Analyzer Options
set_global_assignment -name DO_MIN_TIMING OFF
set_global_assignment -name REPORT_IO_PATHS_SEPARATELY OFF
set_global_assignment -name CUT_OFF_PATHS_BETWEEN_CLOCK_DOMAINS ON
set_global_assignment -name CUT_OFF_READ_DURING_WRITE_PATHS ON
set_global_assignment -name CUT_OFF_IO_PIN_FEEDBACK ON
set_global_assignment -name DO_COMBINED_ANALYSIS OFF
set_global_assignment -name ANALYZE_LATCHES_AS_SYNCHRONOUS_ELEMENTS ON
set_global_assignment -name ENABLE_RECOVERY_REMOVAL_ANALYSIS OFF
set_global_assignment -name ENABLE_CLOCK_LATENCY OFF
```

## Analysis & Synthesis Options

Edit the following default analysis and synthesis options for the Quartus process shown below.

```
#Analysis & Synthesis Options
set_global_assignment -name CYCLONE_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name CYCLONEII_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name STRATIX_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name MAXII_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name APEX20K_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name STRATIXII_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name TRUE_WYSIWYG_FLOW OFF
set_global_assignment -name ADV_NETLIST_OPT_SYNTH_GATE_RETIME OFF
set_global_assignment -name ADV_NETLIST_OPT_RETIME_CORE_AND_IO ON
```

## Specifying Xilinx Place-and-Route Options in a Tcl File

This section shows you how to customize your Xilinx place-and-route run by specifying a place-and-route Tcl file. This file uses the Xilinx xtclsh flow. You can use either the default Tcl file or create a custom file.

This is the recommended methodology, but if you must use the old Xilinx xflow, refer to the procedure in [Specifying Xilinx Place-and-Route Options in an .opt File](#), on page 341.

1. To use the default place-and-route options, do the following:
  - Click the Add P&R Implementation button in the Project view and select Use Default Options File in the dialog box.
  - Click OK.

By default, the software uses the Tcl file located in the installation directory. This file is used by the Xilinx xtclsh executable to run the P&R tool.

2. To use an existing Tcl options file, do the following:
  - Click the Add P&R Implementation button in the Project view.



- Click Existing Options File. Select the file you want, and click Open.
  - Right-click the implementation, select Add New Place & Route Job, and make sure the correct options file is selected. Click OK.
3. To create a new place-and-route Tcl file, do this:
- Click the Add P&R Implementation button in the Project view.
  - Click Create New Options File. Specify a file name and click Open. The tool generates a default Tcl file and automatically adds it to the project.

A text window opens with the default Tcl file options.

4. Edit the file.
- Edit the default options to customize this file.
  - Save the file.
5. Synthesize, place, and route the design.
- Right-click the implementation, select Add New Place & Route Job, and make sure the Tcl file is selected.
  - Select Run Place & Route following synthesis. Click OK.

The software uses the Tcl file to place and route the design after synthesis.

6. View the results.
- Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
  - View the log file xflow.log for information about the run.

## Specifying Xilinx Place-and-Route Options in an .opt File

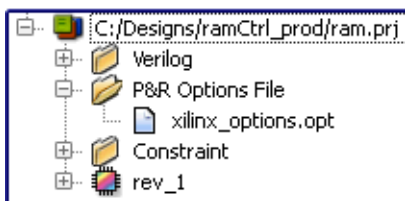
If you are using the old Xilinx xflow, you specify P&R options in a .opt file instead of a Tcl file, which is the recommended methodology (see [Specifying Xilinx Place-and-Route Options in a Tcl File, on page 340](#).)

1. To use the default place-and-route options, do the following:
- Click the Add P&R Implementation button in the Project view and select Use Default Options File in the dialog box.

- Click OK.

By default, the software uses the `opt` file located in the installation directory, which is used by the `xflow` executable to run the P&R tool.

2. To use an existing options file, do the following:
  - Click Implementation Options, go to the Device tab, and enable Use Xilinx Xflow. By default, the synthesis tools use the `xtclsh` executable and the corresponding Tcl options file, so you must explicitly turn on the Use Xilinx Xflow option to use Xflow.
  - Click the Add P&R Implementation button in the Project view.
  - Click Existing Options File. Select the `opt` file you want to use, and click Open.
  - Select the implementation in the project view, right-click, and select Add Place & Route Job. In the dialog box, and sure that the correct options file is selected. Click OK.
3. To create a new place-and-route options file, do this:
  - Click the Add P&R Implementation button in the Project view.
  - Select File->New. Set the file type to Xilinx Option File, type a file name., enable the Add to Project option, and click OK. This file is automatically added to the project.



A text window opens with the options file.

4. Edit the file.
  - Edit the default options to customize this options file. For more information about the contents of this file, see [Options in the Xilinx Place-and-Route Options File](#), on page 343.
  - Save the file.

5. Synthesize, place, and route the design.
  - Return to the Add New Place & Route Job dialog box, and make sure the options file you created is selected.
  - Select Run Place & Route following synthesis. Click OK.

The software uses the options file to place and route the design after synthesis.

6. View the results.
  - Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
  - View the log file xflow.log for information about the run.

## Options in the Xilinx Place-and-Route Options File

To customize the Xilinx place-and-route options file, you can edit the default options file (xilinx\_par.opt). This file contains the options for the following place-and-route processes:

- [Translator Options](#)
- [Mapper Options](#)
- [Place-and-Route Options](#)
- [Post Place-and-Route Timing Report Options](#)
- [Bitgen Generation Options](#)

## Translator Options

Edit the following default translator options for the ngdbuild command as shown below.

```

#####
Translator Options
#####
Type "ngdbuild -h" for a detailed list of ngdbuild command line options
#####
Program ngdbuild
-intstyle xflow; # Message Reporting Style: ise, xflow, or silent
-nt timestamp; # NGO File generation. Regenerate only when
 # source netlist is newer than existing
 # NGO file (default)
<userdesign>; # User design - pick from xflow command line
<design>.ngd; # Name of NGD file. Filebase same as design filebase
##-p <partname>; # Partname to use - picked from xflow commandline
##-sd <source_dir>; #Add "source_dir" to the list of directories
#to search when resolving netlist file references
##-uc <ucf_file>; #Use specified "User Constraint File".
#The file <design_name>.ucf is used by default
#if it is found in the local directory.
##-insert_keep_hierarchy; # Retain hierarchy identified by individual source input netlists
End Program ngdbuild

```

## Mapper Options

Edit the following default mapper options for the map command as shown below.

```

#####
Mapper Options
#####
Type "map -h <architecture>" for a detailed list of map command line options
#####
Program map
-intstyle xflow; # Message Reporting Style: ise, xflow, or silent
-cm area; # Cover mode.
-pr b; # Pack internal flops/latches into input (i), output (o),
 # or both (b) types of IOBs.
-k 4; # Function size for covering combinational logic.
-c 100; # Pack unrelated logic into clbs.
-o <design>_map.ncd; # Output Mapped ncd file
<inputdir><design>.ngd; # Input NGD file
<inputdir><design>.pcf; # Physical constraints file
END Program map

```

## Place-and-Route Options

Edit the following default place-and-route options for the par command as shown below.

```
#####
Place & Route Options
#####
Type "par -h" for a detailed list of par command line options
#####
Program par
-w; # Overwrite existing placed and routed ncd
-intstyle xflow; # Message Reporting Style: ise, xflow, or silent
-ol high; # Overall effort level
-t 1; # Placer cost table entry.
-xe c;
<design>_map.ncd; # Input mapped NCD file
<inputdir><design>.ncd; # Output placed and routed NCD
<inputdir><design>.pcf; # Input physical constraints file
END Program par
```

## Post Place-and-Route Timing Report Options

Edit the following default post place-and-route timing report options for the `post_par_trce` command as shown below.

```
#####
Post PAR Timing Report Options
#####
Type trce -h for detailed list of trce command line options
#####
Program post_par_trce
-intstyle xflow;
-e 100;
<inputdir><design>.ncd;
<inputdir><design>.pcf;
-o <design>.twr;
END Program post_par_trce
```

## Bitgen Generation Options

Edit the following default bit generation options for the `bitgen` command as shown below.

```
#####
Bitgen Generation
#####
Type "bitgen -h" for detailed list of bitgen command line options
Uncomment following commands to generate a bitstream
#####
##Program bitgen
##-intstyle xflow;
##<inputdir><design>.ncd;
##-l; # Create logic allocation file
##-w; # Overwrite existing output file
##-m; # Create mask file
##<inputdir><design>.bit;
##<inputdir><design>.pcf;
##END Program bitgen
```

## Specifying Xilinx Global Placement Options

For graph-based physical synthesis in Xilinx designs, you can override the default values for global placement in the options file.

1. Create a file with your preferred global placement options.
2. Use the `SYN_XILINX_GLOBAL_PLACE_OPT` environment variable to point to your options file:

```
SYN_XILINX_GLOBAL_PLACE_OPT = "C:/Temp/test.opt"
```

# Setting Constraints for Physical Synthesis

The following describe how to set up various constraints for physical synthesis. Some procedures are technology-specific or may apply only if you are using Design Planner.

- [Using Design Planner Floorplan Constraints](#), on page 347
- [Translating Pin Location Files](#), on page 348
- [Translating Actel I/O Constraints](#), on page 348
- [Setting Physical Synthesis Constraints for Altera](#), on page 349

For additional information on translating Altera QSF and Xilinx UCF constraints to the synthesis sdc format, see [Translating Altera QSF Constraints, on page 253](#) and [Converting and Using Xilinx UCF Constraints, on page 255](#), respectively.

## Using Design Planner Floorplan Constraints

For physical synthesis flows that use Design Planner (Altera and Xilinx technologies), use the floorplan constraints you generated to drive synthesis. You can use the following procedure before logic synthesis, because you only need a compiled design.

1. Start with a compiled design.

The Design Planner flows are only supported for certain Altera and Xilinx technologies.

2. Floorplan the design with Design Planner, and assign RTL modules, paths or components to regions on the device. See [Chapter 11, Floorplanning with Design Planner](#) for details.
3. Add the floorplan file to the design.
  - After floorplanning, add the design plan file (.sfp), to the project.
  - Enable the file in the Implementation Options ->Design Planning tab.

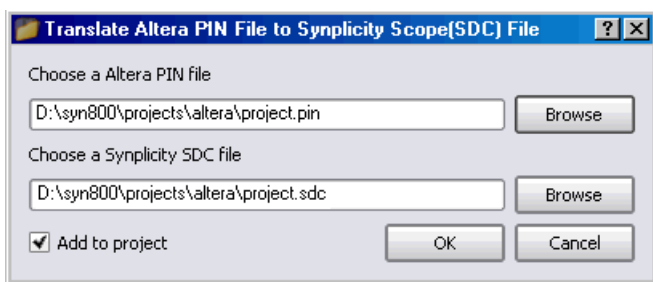
4. Run synthesis.

You can run the preliminary logic synthesis step of the physical synthesis flows, or you can run physical synthesis. When you run physical synthesis, the placement constraints from the sfp file are honored.

## Translating Pin Location Files

You can automatically convert Altera and Xilinx place-and-route pin location constraint files to SCOPE constraint files (.sdc) with the Run->Translate Vendor IO command.

1. Select Run ->Translate Vendor IO.
2. Enter the name of the Altera .pin or Xilinx .pad file you want to translate.
3. Type a name for the .sdc constraint file you want to create.



4. Click on Add to Project, as appropriate, then click OK.

The pin locations from the files are translated into SDC constraints. If you add the constraint file to the project, it can be used for design planning and synthesis. Note that if there are pin assignment conflicts between SDC constraints and pin locations assigned in Design Planner, the SDC constraints take precedence. For other methods to assign pins, see [Importing Pin Assignments from Pin Assignment Files](#), on page 498 and [Assigning Pins Interactively](#), on page 495.

## Translating Actel I/O Constraints

This section provides information on defining timing and physical constraints and attributes for physical synthesis in Actel designs. You can use the pdc2sdc utility to automatically convert the Actel constraints, or you can do it manually.

1. Lock down the I/O placement for physical synthesis.
  - Make sure you have run logic synthesis.



- Convert the I/O loc constraints in the *designname\_ba.pdc* place-and-route file to the *sdc* format.

You do this by assigning the *syn\_loc* attribute to the I/Os either in the Attributes tab of the SCOPE UI, or by manually assigning the *syn\_loc* attribute to the I/Os in the *.pdc* file. For the attribute syntax, see [syn\\_loc Attribute](#), on page 1029 of the *Reference Manual*.

Alternatively, you can use the *pdcs2sdc* command line utility to convert Actel physical constraints to synthesis constraints. Run *pdcs2sdc* from any shell window using the following syntax to translate the pin locations from the fil into *sdc* constraints:

```
pdcs2sdc -ipdc <constraints_file>.pdc
 -osdc <constraints_file>.sdc
 [-opdc <residual_constraints_file>.pdc] [-all]
 [-silent]
```

For more detail about the syntax, see [Actel pdcs2sdc Utility](#), on page 545 in the *Reference Manual*.

## 2. Run a constraint check.

- Make sure you target a technology that supports this feature.
- Generate a constraint file, then select Run->Constraint Check. This command generates a report that checks the syntax and applicability of the timing constraints in the *.sdc* file(s) for your project. The report is written to the *project\_name\_cck.rpt* file.

## Setting Physical Synthesis Constraints for Altera

The following procedure describes the general procedure for setting up physical synthesis constraints for Altera designs.

1. Create an *sdc* timing constraints file, as described in [Creating a Constraint File Using the SCOPE Window](#), on page 212, and add it to your project.

Use timing constraints to specify performance goals for the design and describe the environment.

2. Translate Altera QSF constraints, as described in [Translating Altera QSF Constraints](#), on page 253.

3. Select Run->Constraint Check to check the constraints in the constraints file. See [Checking Constraint Files, on page 104](#).

This command generates a report that checks the syntax and applicability of the timing constraints in the .sdc file(s) for your project. The report is written to the *project\_name\_cck.rpt* file.

4. If you are using Design Planner, create a floorplan file and add it to your project. See [Using Design Planner Floorplan Constraints, on page 347](#) for details.

# Forward-Annotating Physical Synthesis Constraints

See the following for more information:

- [Forward Annotating Altera Physical Constraints](#), on page 351

## Forward Annotating Altera Physical Constraints

You can choose to forward annotate physical constraints from the Synplify Premier physical synthesis tool, or else, let the Quartus II place-and-route tool determine how to handle the physical constraints. Forward-annotation is only available for Altera Stratix IV, Stratix III, Stratix II GX, or Stratix II devices. Use the following procedure.

### New Implementation

Do the following for a new implementation:

1. Enable the Physical Synthesis switch.

If the switch is disabled, physical synthesis optimizations are performed but placement constraints will not be forward annotated.

2. Click on the Add P&R Implementation button from the Project view or right-click and select Add Place & Route Job from the popup menu to create a new P&R implementation.
3. On the Add New Place & Route Job dialog box, enable or disable the Use placement constraints from physical synthesis option. By default, this option is enabled.

The Physical Synthesis switch must be enabled to use this option. When this option is disabled, physical synthesis optimizations are performed but placement constraints will not be forward annotated.

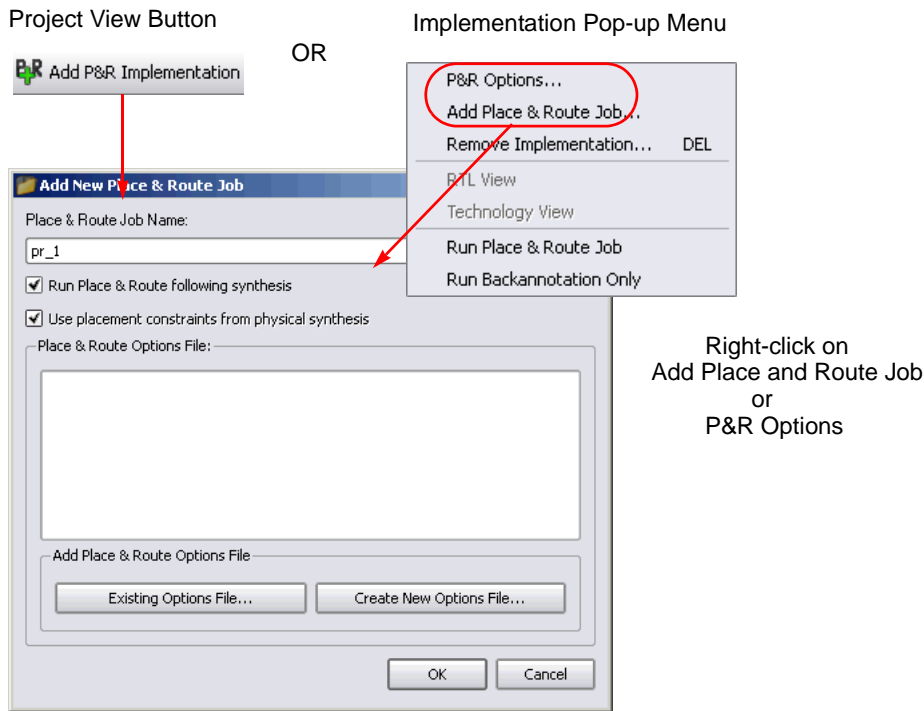
### Existing Implementation

Do the following for an existing place-and-route implementation:

1. Enable the Physical Synthesis switch.

If the switch is disabled, physical synthesis optimizations are performed but placement constraints will not be forward annotated.

2. In the Project view, select the place-and-route implementation, then right-click and select P&R Options from the popup menu.
3. Enable or disable the Use placement constraints from physical synthesis option from the popup dialog box. By default, this option is enabled.



## Backannotating Physical Synthesis Constraints

For more accurate results, you can resynthesize your design using place-and-route data from the back-end tools.

### Backannotating Place-and-Route Data

You can also choose to back annotate place-and-route data which provides accurate timing and placement information during physical synthesis. However, this option is only applicable for certain Altera and Xilinx technologies.

Use the following procedure to backannotate place and route data. Do not backannotate designs that contain IP cores.

#### New Implementation

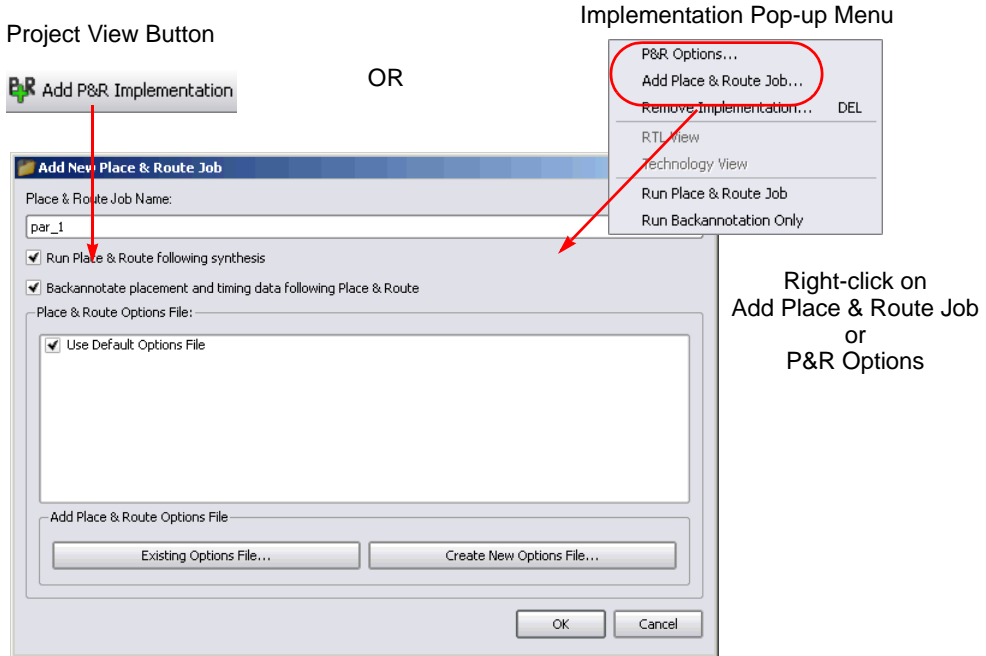
Do the following for a new place-and-route implementation:

1. Click on the Add P&R Implementation button from the Project view.
2. On the Add New Place & Route Job dialog box, enable the Backannotate placement and timing data following Place & Route option.

#### Existing Implementation

Do the following for an existing place-and-route implementation:

1. In the Project view, select the place-and-route implementation, then right-click and select P&R Options from the popup menu.
2. Enable the Back Annotate option from the popup dialog box.



## Generating a Xilinx Coreloc Placement File

You can use the Synplify Premier tool to generate a coreloc constraint file that contains the placement for all “anchor” or “core” instances in the design, like IOs, BlockRAM s, BlockMULTs, FIFOs, DSP48s, DCMs, and BUFs. You can pass this file to the P&R tool for backannotation.

Using core location constraints ensures that block and I/O placement are equivalent when you compare logic and physical synthesis. Include the coreloc.sdc file to both logic and physical synthesis runs. This placement information is also very useful for timing convergence. By stabilizing block and I/O locations, you eliminate any changes to results that stem from placement variations when running small designs..

To generate a coreloc file, do the following:

1. Before you run synthesis, make sure to enable the Backannotate placement and timing Data following Place & Route switch when you create the place-and-route implementation before running synthesis.

The `filename_coreloc.sdc` file is generated during place-and-route backannotation and is written to the results directory. The *filename* is the same base name as the EDIF output netlist (`filename.edf`).

2. After the synthesis run, when the file has been generated, add the `filename_coreloc.sdc` file to your project.

To guarantee consistent and stable comparisons, include the `coreloc.sdc` file to both logic and physical synthesis runs.

3. Re-run synthesis.



**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.



## CHAPTER 8

# 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 358
- [Defining State Machines for Synthesis](#), on page 367
- [Inferring RAMs](#), on page 372
- [Initializing RAMs](#), on page 400
- [Inferring Shift Registers](#), on page 410
- [Working with LPMs](#), on page 416

## 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 358
- [Instantiating Black Boxes and I/Os in VHDL](#), on page 360
- [Adding Black Box Timing Constraints](#), on page 362
- [Adding Other Black Box Attributes](#), on page 366

The Fix Gated Clocks option is only available in the Synplify Pro and Synplify Premier tools. For information about using black boxes with the Fix Gated Clocks option, see [Working with Gated Clocks](#), on page 464.

### 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 360. Additional information about black boxes can be found in [Working with Gated Clocks](#), on page 464, [Instantiating CoreGen Cores](#), on page 804, and [Instantiating Virtex PCI Cores](#), on page 805. The Fix Gated Clocks option is only available in the Synplify Pro and Synplify Premier tools.

The following process shows you how to instantiate both types as black boxes. Refer to the `Synplify_install_dir/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 `Synplify_install_dir/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 BBDLHS(D,E,GIN,GOUT,PAD,Q)
 /* synthesis syn_black_box black_box_pad_pin="PAD" */
endmodule
```

- 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 362](#) and [Adding Other Black Box Attributes, on page 366](#).
  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 358](#).

The following process shows you how to instantiate both types as black boxes. Refer to the `Synplify_install_dir/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 `Synplify_install_dir/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.

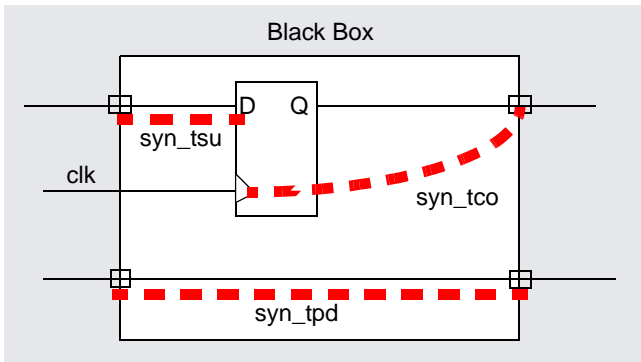
```
begin
data_pad: mybuf port map (
 O => data_core,
 I => data);
```

4. Add timing constraints and attributes. See [Adding Black Box Timing Constraints, on page 362](#), [Using Gated Clocks for Black Boxes, on page 471](#), and [Adding Other Black Box Attributes, on page 366](#). The Fix Gated Clocks option is only available in the Synplify Pro and Synplify Premier tools.

## 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 366](#) and [Using Gated Clocks for Black Boxes, on page 471](#). The Fix Gated Clocks option is only available in the Synplify Pro and Synplify Premier tools.



1. Define the instance as a black box, as described in [Instantiating Black Boxes and I/Os in Verilog, on page 358](#) or [Instantiating Black Boxes and I/Os in VHDL, on page 360](#).
2. Determine the kind of constraint for the information you want to specify:

| To define...                                       | Use...  |
|----------------------------------------------------|---------|
| Propagation delay through the black box            | syn_tpd |
| Setup delay (relative to the clock) for input pins | syn_tsu |
| Clock-to-output delay through the black box        | syn_tco |

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<br>syntax | attribute <i>attribute_name</i> < <i>n</i> > : " <i>att_value</i> " |
|----------------|---------------------------------------------------------------------|

---

|                           |                                                                                                         |
|---------------------------|---------------------------------------------------------------------------------------------------------|
| Verilog-style<br>notation | attribute <i>attribute_name</i> < <i>n</i> > of <i>bbox_name</i> :<br>component is " <i>att_value</i> " |
|---------------------------|---------------------------------------------------------------------------------------------------------|

---

The following table shows the appropriate syntax for *att\_value*. See the *Reference Manual* for complete syntax information.

| Attribute                   | Value Syntax                                  |
|-----------------------------|-----------------------------------------------|
| <i>syn_tsu</i> < <i>n</i> > | <i>bundle</i> -> [!]clock = <i>value</i>      |
| <i>syn_tco</i> < <i>n</i> > | [!]clock -> <i>bundle</i> = <i>value</i>      |
| <i>syn_tpd</i> < <i>n</i> > | <i>bundle</i> -> <i>bundle</i> = <i>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_tsu1 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 *Reference Manual*.

```
module ram32x4 (z, d, addr, we, clk)
 /* synthesis syn_black_box
 syn_tpd1="addr[3:0]->z[3:0]=8.0"
 syn_tsu1="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.

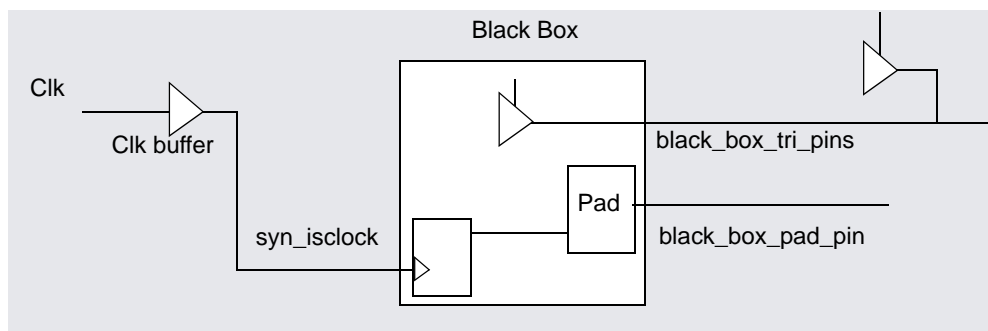
The resulting constraint file contains syntax like this:

```
define_attribute v:{blackbox_module} attribute<n> {att_value}
```

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 *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 471](#). The Fix Gated Clocks option is only available in the Synplify Pro and Synplify Premier tools.



1. To specify that a clock pin on the black box has access to global clock routing resources, use `syn_isclock`.

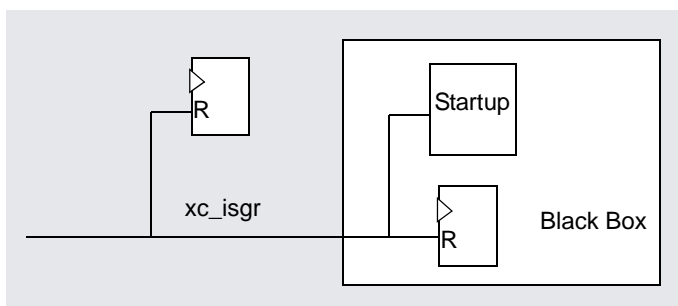
Depending on the technology, different clock resources are inserted. In Xilinx, the software inserts BUFG, for Actel it inserts CLKBUF, and for QuickLogic, it inserts Q\_CKPAD.

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 Xilinx, some Altera families, Actel, QuickLogic, and some Lattice 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`.

4. To ensure consistency between synthesized black box netlist names and the names generated by third party tools or IP cores, use the following attributes (Xilinx only):
  - `syn_edif_bit_format`
  - `syn_edif_scalar_format`
5. To specify that a port on a black box is connected to an internal STARTUP block in Xilinx XC4000 architectures, use the `xc_isgr` directive.



## 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 368
- [Defining State Machines in VHDL](#), on page 369
- [Specifying FSMs with Attributes and Directives](#), on page 369

For information about the attributes used to define state machines, see [Running the FSM Compiler](#), on page 455.

## Defining State Machines in Verilog

The synthesis software recognizes and automatically extracts state machines from the Verilog source code if you follow these coding guidelines. The software attaches the `syn_state_machine` attribute to each extracted FSM.

For alternative ways to define state machines, see [Defining State Machines in VHDL, on page 369](#) and [Specifying FSMs with Attributes and Directives, on page 369](#).

- In Verilog, model the state machine with `case`, `casex`, 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.
- Use explicit state values for states using `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 whereas `parameter` definitions are local. Local definitions make it easier to reuse certain state names in multiple FSM designs. For example, you might want to reuse common state names like `RESET`, `IDLE`, `READY`, `READ`, `WRITE`, `ERROR` and `DONE`. If you use ``define` to assign state names, you cannot reuse a state name because the name has already been taken in

the global name space. To use the names multiple times, you have to ``undef` state names between modules and redefine them with ``define` state names in the new FSM modules. 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 coding guidelines. For alternative ways to define state machines, see [Defining State Machines in Verilog, on page 368](#) and [Specifying FSMs with Attributes and Directives, on page 369](#).

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 (see [Optimizing State Machines, on page 453](#)), or you can manually specify attributes to define the state machines. You attach the attributes to the state registers. For detailed information about the attributes and their syntax, see the *Reference Manual*.

The following steps show you how to use attributes to define FSMs for extraction. For alternative ways to define state machines, see [Defining State Machines in Verilog, on page 368](#) and [Defining State Machines in VHDL, on page 369](#).

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 [Entering Attributes and Directives, on page 304](#).

2. To determine the encoding style used 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 and the FSM Explorer honor this setting. The different values for this attribute are briefly described here:

| Situation: If...                            | <code>syn_encoding</code> Value                                                                                  | Explanation                                                                                                                                                                                                    |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Area is important                           | <code>sequential</code>                                                                                          | One of the smallest encoding styles.                                                                                                                                                                           |
| Speed is important                          | <code>onehot</code>                                                                                              | Usually the fastest style and suited to most FPGA styles.                                                                                                                                                      |
| Recovery from an invalid state is important | <code>safe</code> , with another style. For example:<br><pre>/* synthesis syn_encoding = "safe, onehot" */</pre> | Forces the state machine to reset. For example, where an alpha particle hit in a hostile operating environment causes a spontaneous register change, you can use <code>safe</code> to reset the state machine. |

| <b>Situation: If...</b>                | <b>syn_encoding Value</b> | <b>Explanation</b>                                                                                                                                                                                                                             |
|----------------------------------------|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| There are <5 states                    | sequential                | Default encoding.                                                                                                                                                                                                                              |
| Large output decoder follows the FSM   | sequential or gray        | Could be faster than onehot, even though the value must be decoded to determine the state. For <b>sequential</b> , more than one bit can change at a time; for <b>gray</b> , 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 this method for user-defined FSM encoding. For example:

```
attribute syn_enum_encoding of state_type : type is "001 010 101";
```

## Inferring RAMs

There are two methods of handling RAMs: instantiation and inference. The software can automatically infer RAMs if they are structured correctly in your source code. For details, see the following sections:

- [Inference Versus Instantiation](#), on page 372
- [Basic Guidelines for Coding RAMs](#), on page 373
- [Specifying RAM Implementation Styles](#), on page 378
- [Implementing Altera RAMs Automatically](#), on page 379
- [Implementing Xilinx RAMs Automatically](#), on page 383
- [Implementing Altera FLEX and APEX RAMs](#), on page 385
- [Implementing Altera Stratix Multi-Port RAMs](#), on page 388
- [Inferring Altera Stratix III LUTRAMs](#), on page 389
- [Inferring Xilinx Block RAMs Using Registered Addresses](#), on page 391
- [Inferring Xilinx Block RAMs Using Registered Output](#), on page 393
- [Initializing Xilinx RAM](#), on page 404
- [Mapping Xilinx ROM to Block RAM](#), on page 398

For information about generating RAMs with SYNCore, see [Specifying RAMs with SYNCore](#), on page 119.

## Inference Versus Instantiation

There are two methods to handle RAMs: instantiation and inference. Many FPGA families provide technology-specific RAMs that you can instantiate in your HDL source code. The software supports instantiation, but you can also set up your source code so that it infers the RAMs. The following table sums up the pros and cons of the two approaches.



**Inference in Synthesis****Advantages**

Portable coding style  
 Automatic timing-driven synthesis  
 No additional tool dependencies

**Limitations**

Glue logic to implement the RAM might result in a sub-optimal implementation.  
 Can only infer synchronous RAMs  
 No support for address wrapping  
 No support for RAM enables, except for write enable  
 Pin name limitations means some pins are always active or inactive

**Instantiation****Advantages**

Most efficient use of the RAM primitives of a specific technology  
 Supports all kinds of RAMs

**Limitations**

Source code is not portable because it is technology-dependent.  
 Limited or no access to timing and area data if the RAM is a black box.  
 Inter-tool access issues, if the RAM is a black box created with another tool.

## Basic Guidelines for Coding RAMs

Read through the limitations before you start. See [Inference Versus Instantiation, on page 372](#) for information. The following steps describe general rules for coding RAMs so that the compiler infers them; to ensure that they are mapped to the vendor-specific implementation you want, see [Specifying RAM Implementation Styles, on page 378](#), [Implementing Altera RAMs Automatically, on page 379](#), and [Implementing Xilinx RAMs Automatically, on page 383](#).

1. Make sure that the RAM meets minimum size and address width requirements for your technology. The software implements RAMs that are smaller than the minimum as registers.
2. Structure the assignment to a VHDL signal/Verilog register as follows:
  - To infer a RAM, structure the code as an indexed array or a case structure. Code it as a two-dimensional array (VHDL) or memory (Verilog) with writes to one process.
  - Control the structure with a clock edge and a write enable.

The software extracts RAMs even if write enables are tied to true (VCC), if you have complex write enables coded in nested IF statements, or if you have RAMs with synchronous resets.

3. For a single-port RAM, make the address for indexing the write-to the same as the address for the read-from. The following code and figure illustrate how the software infers a single-port RAM.

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

entity ramtest is
port (q : out std_logic_vector(3 downto 0);

 d : in std_logic_vector(3 downto 0);
 addr : in std_logic_vector(2 downto 0);
 we : in std_logic;
 clk : in std_logic);
end ramtest;

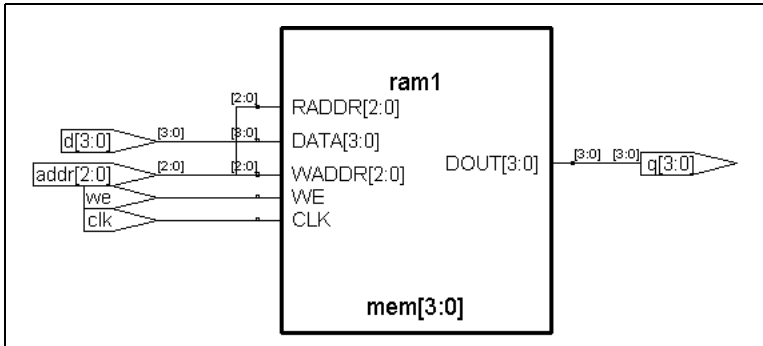
architecture rtl of ramtest is

type mem_type is array (7 downto 0) of std_logic_vector
 (3 downto 0);
signal mem : mem_type;

begin
q <= mem(conv_integer(addr));

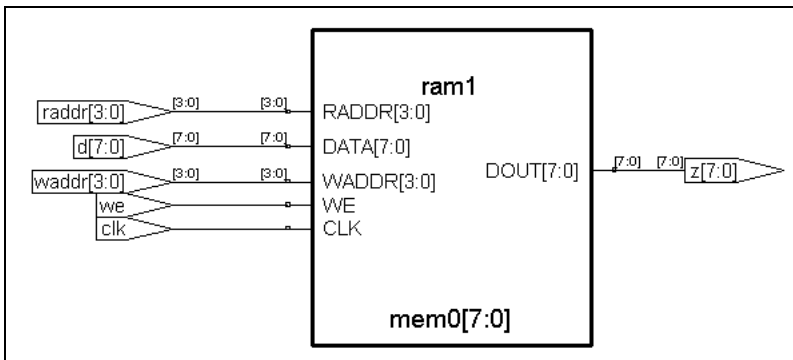
process (clk) begin
if rising_edge(clk) then
 if (we = '1') then
 mem(conv_integer(addr)) <= d;
 end if;
end if;
end process;

end rtl;
```



For technology-specific details, see [Implementing Altera RAMs Automatically, on page 379](#) and [Implementing Xilinx RAMs Automatically, on page 383](#).

4. For a dual-port RAM, make the write-to and read-from addresses different. The following figure and code example illustrate how the software infers a dual-port RAM.



```

module ram16x8(z, raddr, d, waddr, we, clk);
output [7:0] z;
input [7:0] d;
input [3:0] raddr, waddr;
input we;
input clk;
reg [7:0] z;
reg [7:0] mem0, mem1, mem2, mem3, mem4, mem5, mem6, mem7;
reg [7:0] mem8, mem9, mem10, mem11, mem12, mem13, mem14, mem15;
always @(mem0 or mem1 or mem2 or mem3 or mem4 or mem5 or mem6 or
mem7 or mem8 or mem9 or mem10 or mem11 or mem12 or mem13 or

```

```
 mem14 or mem15 or raddr)
begin
 case (raddr[3:0])
 4'b0000: z = mem0;
 4'b0001: z = mem1;
 4'b0010: z = mem2;
 4'b0011: z = mem3;
 4'b0100: z = mem4;
 4'b0101: z = mem5;
 4'b0110: z = mem6;
 4'b0111: z = mem7;
 4'b1000: z = mem8;
 4'b1001: z = mem9;
 4'b1010: z = mem10;
 4'b1011: z = mem11;
 4'b1100: z = mem12;
 4'b1101: z = mem13;
 4'b1110: z = mem14;
 4'b1111: z = mem15;
 endcase
end

always @(posedge clk) begin
 if(we) begin
 case (waddr[3:0])
 4'b0000: mem0 = d;
 4'b0001: mem1 = d;
 4'b0010: mem2 = d;
 4'b0011: mem3 = d;
 4'b0100: mem4 = d;
 4'b0101: mem5 = d;
 4'b0110: mem6 = d;
 4'b0111: mem7 = d;
 4'b1000: mem8 = d;
 4'b1001: mem9 = d;
 4'b1010: mem10 = d;
 4'b1011: mem11 = d;
 4'b1100: mem12 = d;
 4'b1101: mem13 = d;
 4'b1110: mem14 = d;
 4'b1111: mem15 = d;
 endcase
 end
end
endmodule
```

For technology-specific details, see [Implementing Altera RAMs Automatically, on page 379](#) and [Implementing Xilinx RAMs Automatically, on page 383](#).

5. To infer multi-port RAMs or nrams (certain technologies only), do the following:
  - Target a technology that supports multi-port RAMs.
  - Register the read address.
  - Add the `syn_ramstyle` attribute with a value of `no_rw_check`. If you do not do this, the compiler errors out.
  - Make sure that the writes are to one process. If the writes are to multiple processes, use the `syn_ramstyle` attribute to specify a RAM.
6. For RAMs where inference is not the best solution, use either one of these approaches:
  - Implement them as regular logic using the `syn_ramstyle` attribute with a value of `registers`. You might want to do this if you have to conserve RAM resources.
  - Instantiate RAMs using the black box methodology. Use this method in cases where RAM is implemented in two cells instead of one because the RAM address range spans the word limit of the primitive and the software does not currently support address wrapping. If the address range is 8 to 23 and the RAM primitive is 16 words deep, the software implements the RAM as two cells, even though the address range is only 16 words deep. Refer to the list of limitations in [Inference Versus Instantiation, on page 372](#) and the vendor-specific information referred to in the previous step to determine whether you should instantiate RAMs.
7. Synthesize your design.

The compiler infers one of the following RAMs from the source code. You can view them in the RTL view:

|      |                |
|------|----------------|
| RAM1 | RAM            |
| RAM2 | Resettable RAM |
| NRAM | Multi-port RAM |

If the number of words in the RAM primitive is less than the required address range, the compiler generates two RAMs instead of one, leaving any extra addresses unused.

Once the compiler has inferred the RAMs, the mapper implements the inferred RAMs in the technology you specified. For details of how to map the RAM inferred by the compiler to the implementation you want, see [Specifying RAM Implementation Styles, on page 378](#), [Implementing Altera RAMs Automatically, on page 379](#), and [Implementing Xilinx RAMs Automatically, on page 383](#).

## Specifying RAM Implementation Styles

You can manually influence how RAMs are implemented with the `syn_ramstyle` attribute, as described in the following procedure. The valid values vary slightly, depending on the technology you use. Check the *Reference Manual* for the values that apply to the technology you choose.

If you would rather set up your design so that the software automatically maps the RAMs to the components you want, see [Implementing Altera RAMs Automatically, on page 379](#) and [Implementing Xilinx RAMs Automatically, on page 383](#) for some vendor-specific details.

1. If you do not want to use RAM resources, attach the `syn_ramstyle` attribute with a value of `registers` to the RAM instance name or to the signal driven by the RAM.

Use this value for small RAMs. The software implements the RAMs according to the technology. They can be implemented as registers (Altera, Xilinx), LPMs, dedicated RAM resources (QuickLogic) or synchronous dual-port memory cells (some Lattice technologies).

2. To use the dedicated memory resources on the FPGA (Altera technologies), do the following:
  - Set `syn_ramstyle` to `block_ram`.
  - For newer Altera technologies like Stratix, specify mapping to TriMatrix memories by setting `syn_ramstyle` to `M512`, `M4K`, or `M-RAM`.
  - For Flex10K architectures, register the read address, because the technology does not support dual-port RAMs.
  - If you do not want glue logic created, register the RAM output. For Altera Stratix designs, you can set `syn_ramstyle` to `no_rw_check`.

The software implements the RAMS as EABs or ESBs, depending on the technology.

3. To implement RAMs using dedicated Block SelectRAM+ in Xilinx Virtex technologies, do the following. To use distributed memory, see the next step.
  - Set `syn_ramstyle` to `block_ram`.
  - Register the read address, because the technology is fully synchronous.
  - If you do not want to generate glue logic for dual-port RAMs, either register the RAM output or set `syn_ramstyle` to `no_rw_check`. Use this attribute value only if you do not care about a read/write check.
4. To implement RAMs using distributed memory in Xilinx technologies, set `syn_ramstyle` to `select_ram`. Set `syn_ramstyle` explicitly, because by default the software first implements block RAM, and select RAM only if it cannot implement block RAM.

## Implementing Altera RAMs Automatically

The following procedure shows you how to implement various Altera RAMs automatically. You can always override the automatic implementation by specifying the `syn_ramstyle` attribute, as described in [Specifying RAM Implementation Styles, on page 378](#) or instantiate LPMs instead of using RAMs.

1. Follow the guidelines described for RAM inference by the compiler ([Basic Guidelines for Coding RAMs, on page 373](#)).

The Altera mapper does not implement any RAMs that are not first inferred by the compiler.

2. To implement RAM in Flex and Apex families, see the details described in [Implementing Altera FLEX and APEX RAMs, on page 385](#).
3. To implement Stratix block RAM, follow these guidelines:
  - If you are a Verilog user, avoid using blocking statements when you model the RAMs because not all blocking assignments are mapped to block RAM.

- Synchronize the read and write addresses by registering either the read address or output. RAMs with asynchronous read and write are mapped to logic.
- Use `syn_ramstyle` with a value of `no_rw_check` to disable the creation of glue logic in dual-port mode.
- Make sure the Altera Quartus tool is installed, for best results with Stratix II devices. When Quartus is installed, the synthesis software takes advantages of the Stratix II RAMs and MACs and writes out Stratix II primitives that you can view in the Technology view. If you do not install Quartus, the synthesis software infers LPMs.

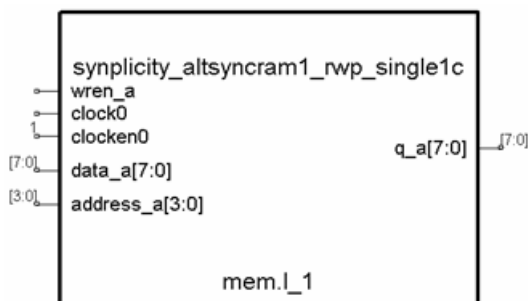
During synthesis, the mapper maps Altera Stratix RAM to ALTSYNCRAM in the following modes:

|               |                                                                                                                        |
|---------------|------------------------------------------------------------------------------------------------------------------------|
| Single-port   | One address bus                                                                                                        |
| Dual-port     | One address bus (where old data cannot be obtained in single-port mode), or<br>Two buses: one each for read and write. |
| Bidirectional | Two buses: one for read/write and one for read only                                                                    |

4. To implement Stratix single-port RAMs, ensure the following:
  - The read and write addresses share a single address.
  - There is only one data input.
  - There is only RAM output.
  - Either the read address or the output is registered.
  - For multiple clocks, both the read address and the output must be registered.

The mapper maps the RAM to the dedicated memory resource, ALTSYNCRAM, which is fully synchronous. It is mapped in `SINGLE_PORT` mode, and all ports are registered. The ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

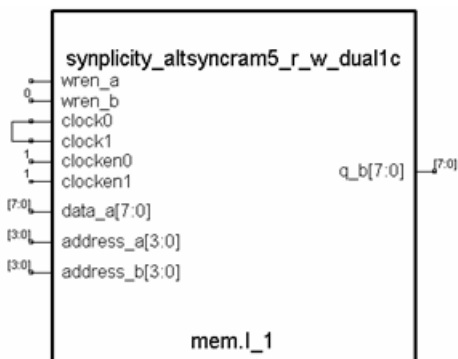




5. To implement Stratix dual-port RAMs, make sure of the following:
- The code is written so that the hardware exactly matches the RTL behavior. For example, if your code allows simultaneous reads and writes to the same address, it can result in a mismatch between the RTL and hardware behaviors. In such a case, the mapper does not map the RAM inferred by the compiler to the dedicated ALTSYNCRAM resources and you get a warning message. See [Dual-Port RAM Code Examples, on page 1349](#) of the *Reference Manual*.
  - The design has different read and write addresses.
  - There is only one data input.
  - There is only RAM output.
  - Either the read address or the output is registered.
  - The read and write addresses can have different clocks. However if you register the read, write, and output, at least two of them must share a clock.
  - For multiple clocks, both the read address and the output must be registered.

The mapper maps the RAM to ALTSYNCRAM in DUAL\_PORT mode, which is fully synchronous. The actual ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

The following figure shows one dual-port RAM implementation:



6. To implement Stratix dual-port RAMs in bidirectional mode, make sure of the following:
  - The code must be written so that there are no mismatches between the hardware and RTL behaviors. See [Dual-Port RAM Code Examples, on page 1349](#) of the *Reference Manual* for an explanation and examples.
  - The design has different read and write addresses. There are two read addresses.
  - There is only one data input.
  - There are two RAM outputs.
  - Either the read address or the output is registered.
  - The read and write addresses can have different clocks. However if you register the read, write, and output, at least two of them must share a clock.
  - For multiple clocks, both the read address and the output must be registered.

The mapper maps the RAM to ALTSYNCRAM in BIDIR\_DUAL\_PORT mode, which is fully synchronous. The actual ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

7. To implement Stratix multi-ports RAMs automatically, see [Implementing Altera Stratix Multi-Port RAMs, on page 388](#).

## Implementing Xilinx RAMs Automatically

The following procedure shows you how to implement various Xilinx RAMs automatically. You can always override the automatic implementation by specifying the `syn_ramstyle` attribute, as described in [Specifying RAM Implementation Styles](#), on page 378.

1. Follow the guidelines described for RAM inference by the compiler ([Basic Guidelines for Coding RAMs](#), on page 373).

The Xilinx mapper does not implement any RAMs that are not first inferred by the compiler.

2. To automatically implement distributed RAM, make sure the write operation is synchronous and the read operation is asynchronous.

The Xilinx mapper implements RAMs inferred by the compiler as asynchronous RAMs, using the CLB resources.

3. To implement block SelectRAM+, do the following:

- Make the write port synchronous. The read port can be asynchronous.
- Register the read address (see [Inferring Xilinx Block RAMs Using Registered Addresses](#), on page 391).
- Make sure the RAM is a minimum of 2K bits.

The Xilinx mapper automatically implements RAMs inferred by the compiler as Block SelectRAM+, using the dedicated memory resources on the FPGA. The enable pin is tied to active and the reset pin is tied to inactive.

4. To implement single-port block RAM automatically, do the following:

- Register the output. (see [Inferring Xilinx Block RAMs Using Registered Output](#), on page 393).
- Make the read and write addresses the same.
- Make sure that the read and write clocks are the same.
- Make sure the read and write enables are the same.

The Xilinx mapper automatically implements RAMs inferred by the compiler as single-port Block SelectRAM+, using the dedicated memory resources on the FPGA. The enable signal has the highest priority.

Where applicable, the tool uses the parity bus to infer data bus widths. The mapper also uses the Write modes in some Xilinx architectures, as described in the next step.

5. To implement dual-port block RAM automatically, do the following:
  - Register the output. (see [Inferring Xilinx Block RAMs Using Registered Output](#), on page 393).
  - Your design can have different read and write addresses, multiple clocks, and different read and write enables.

The Xilinx mapper implements RAMs inferred by the compiler as dual-port block SelectRAM+, using the dedicated memory resources on the FPGA. The dual-port RAM has only one write port. The software automatically inserts glue logic for address collision and recovery, unless you specify otherwise with the `syn_ramstyle` attribute.

The mapper also implements the Write modes available with certain Xilinx architectures to indicate the output value when the write enable is active. The RAM implementations are shown here:

| Write Mode                               | Xilinx Architecture                                                                          | RAM Implementation                                                |
|------------------------------------------|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| Writefirst<br>(data_in goes to data_out) | Virtex-II, Virtex-II Pro,<br>Virtex-4, Spartan-3,<br>Spartan-3 Automotive, and<br>Spartan-3E | Block SelectRAM+<br>(single-port or dual-port)<br>Distributed RAM |
|                                          | Virtex, Virtex-E, Spartan-II,<br>Spartan-IIE, and Spartan-IIE<br>Automotive                  | Block SelectRAM+<br>(single-port or dual-port)<br>Distributed RAM |

| Write Mode                                | Xilinx Architecture                                                                          | RAM Implementation                                   |
|-------------------------------------------|----------------------------------------------------------------------------------------------|------------------------------------------------------|
| Readfirst<br>(memory goes to<br>data_out) | Virtex-II, Virtex-II Pro,<br>Virtex-4, Spartan-3,<br>Spartan-3 Automotive, and<br>Spartan-3E | Block SelectRAM+<br>(single-port)<br>Distributed RAM |
|                                           | Virtex, Virtex-E, Spartan-II,<br>Spartan-IIE, and Spartan-IIE<br>Automotive                  | Distributed RAM                                      |
| Nochange<br>(data_out is unchanged)       | Virtex-II, Virtex-II Pro,<br>Virtex-4, Spartan-3,<br>Spartan-3 Automotive, and<br>Spartan-3E | Block SelectRAM+<br>(single-port)<br>Distributed RAM |
|                                           | Virtex, Virtex-E, Spartan-II,<br>Spartan-IIE, and Spartan-IIE<br>Automotive                  | Distributed RAM                                      |

6. To implement true dual-port block (multi-port) RAM automatically, make sure the design meets the following conditions:
- The compiler has inferred multi-port RAMs (nrams). See [Basic Guidelines for Coding RAMs, on page 373](#) for details.
  - The inferred nram has two writes and one read. The read shares an address with only one of the write ports, or two inferred RAMs share the same write addresses, clocks, and enables, but have different read addresses. In the latter case, the mapper pairs the RAMs together and maps them to true dual-port RAM.

The Xilinx mapper implements RAMs inferred by the compiler as true dual-port block SelectRAM+, using the dedicated memory resources on the FPGA. The dual-port RAM has one read port and multiple write ports. Each write port has its own write clock, write enable, data in, and write address.

## Implementing Altera FLEX and APEX RAMs

The alternative to inferring RAMs in an Altera design is to instantiate LPMs. See [Inference Versus Instantiation, on page 372](#) and [Working with LPMs, on page 416](#).

The software supports single-port RAMs for the FLEX10K family and single-port or dual-port RAMs for the FLEX10KE, APEX20K, and 20KE families. It inserts bypass logic to resolve a read/write behavior difference between the RTL and post-synthesis gate-level simulations. There is a half-cycle difference between the two: the post-RTL simulation shows memory updates occurring on the positive edge of the system clock, and the post-synthesis simulation shows memory updates on the negative edge. The following procedure shows you how to set up your code.

1. Structure your source code as described in [Basic Guidelines for Coding RAMs, on page 373](#).
2. Include an explicit read address register.

The address must be registered to implement a synchronous RAM in an LPM. You do not need an explicit read address for the Flex 10KE, ACEX, APEX, APEX II, Excalibur, and Mercury families, because these architectures support dual-port RAMs with independent read and write registers.

3. To eliminate bypass logic, register the output of the RAM. The following example defines a register, Q, for this purpose:

```
module ram_test(q, a, d, we, clk);
output[7:0] q;
input [7:0] d;
input [6:0] a;
input we, clk;
//Register the RAM output to eliminate glue logic
reg [7:0] q;
reg [6:0] read_add;
reg [7:0] mem [127:0];
always @(posedge clk) begin
q = mem[read_add];
end

always @(posedge clk) begin
if(we)
//Register RAM data and read address
mem[read_add] <= d;
read_add <= a;
end
endmodule
```

When you synthesize this example, the software creates a single-port synchronous RAM, implemented with as few registers as possible. If you

do not care about the insertion of glue logic, do not register the RAM output:

```

module ram_test(q, a, d, we, clk);
output[7:0] q;
input [7:0] d;
input [6:0] a;
input we, clk;
reg [6:0] read_add;
reg [7:0] mem [127:0];
assign q = mem[read_add];

always @(posedge clk) begin
 if(we)
 //Register RAM data and read address
 mem[read_add] <= d;
 read_add <= a;
end
endmodule

```

When you synthesize this example, the software creates a bypass mux to resolve the read/write simulation behavior on the positive and negative edges of the clock.

You can use the `syn_ramstyle` attribute to ensure that the RAM is implemented as an EAB or ESB, or to disable RAM inference as needed. See [Specifying RAM Implementation Styles, on page 378](#) for details.

#### 4. Run synthesis.

The software automatically infers Altera-specific synchronous RAMs and implements them in EABs or ESBs. When source code is written as a single-port RAM, the software implements it as a dual-port RAM with single-port RAM functionality, using the `LPM_RAM_DQ:ALTDPRAM` primitive. The following table lists the family-specific details of implementation:

|                                 |                                                       |          |
|---------------------------------|-------------------------------------------------------|----------|
| FLEX10K                         | Single-port synchronous RAMs                          | LPMRAMDQ |
| FLEX10KE<br>APEX20K<br>APEX20KE | Single-port or dual-port RAMs with asynchronous READs | ALTDPRAM |

## Implementing Altera Stratix Multi-Port RAMs

The software can infer true multi-port RAMs, where both ports are used to read and write simultaneously. Implementing Stratix ALTSYNCRAM components is a two-step process: first the synthesis compiler infers the RAM primitive, and then the mapper maps the primitive to ALTSYNCRAM.

1. Make sure the compiler infers an `nram`, by following the guidelines in [Basic Guidelines for Coding RAMs, on page 373](#).

For multi-port RAMs, the compiler infers an `nram` primitive, where  $n$  is the number of write ports. You can view this in the RTL view.

2. To map the `nram` automatically to ALTSYNCRAM, ensure that it follows these guidelines:
  - The `nram` has two writes and one read. The read shares an address with only one of the write ports.
  - Make sure there are only two clocks, one for each port.
  - You cannot have more than two write ports; `nram` primitives with more than two ports are mapped to logic.
  - The read address is registered.
  - If the output is registered, the mapper retimes and infers block RAM.

The software maps `nram` primitives as follows:

| Primitive Description                                                                                                                                | Mapping                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| 2 write ports, 1 read. The read shares an address with only one of the write ports                                                                   | ALTSYNCRAM in bidir mode                 |
| 2 <code>nrams</code> each with 2 write addresses and 1 read, which share the same write addresses, clocks, and enables, but different read addresses | Paired together and mapped to ALTSYNCRAM |
| > 2 write ports                                                                                                                                      | Logic                                    |
| > 2 clocks                                                                                                                                           | Logic                                    |

After synthesis, the software writes out the following for the place-and-route tool:

```
defparam mem_1_1_Z.lpm_type = "altsyncram";
```



## Inferring Altera Stratix III LUTRAMs

The Altera Stratix III technology has LUTRAM memory components. MLAB (Memory LAB) resources are configured as LUTRAM. MLABs can be configured as single-port RAM or ROM, or simple dual-port RAM. LUTRAM writes occur on the falling edge of the clock and can be configured to have synchronous or asynchronous read.

The following procedure shows you how to set up the synthesis tool to map memory to MLABs and LUTRAMs. Note that you cannot currently map to a LUTRAM ROM, nor can you initialize asynchronous memory.

1. Start with a Stratix III design.
2. Enable the Clearbox flow option.

If this option is not enabled, the memories are mapped to ALTSYNGRAM or ALTDPRAM instead of LUTRAM.

3. Set the `syn_ramstyle` attribute to MLAB.

This automatically maps the RAM to MLAB resources, which can be configured as LUTRAMs. If you do not want to infer LUTRAM, set `syn_ramstyle` to registers.

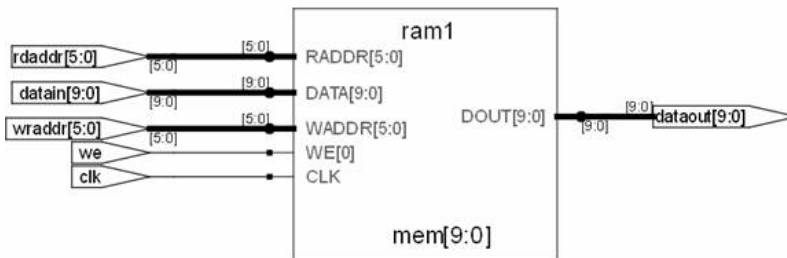
For Verilog code examples that implement LUTRAM, see [Altera Stratix III LUTRAM Examples, on page 695](#) in the *Reference Manual*.

4. Synthesize your design.

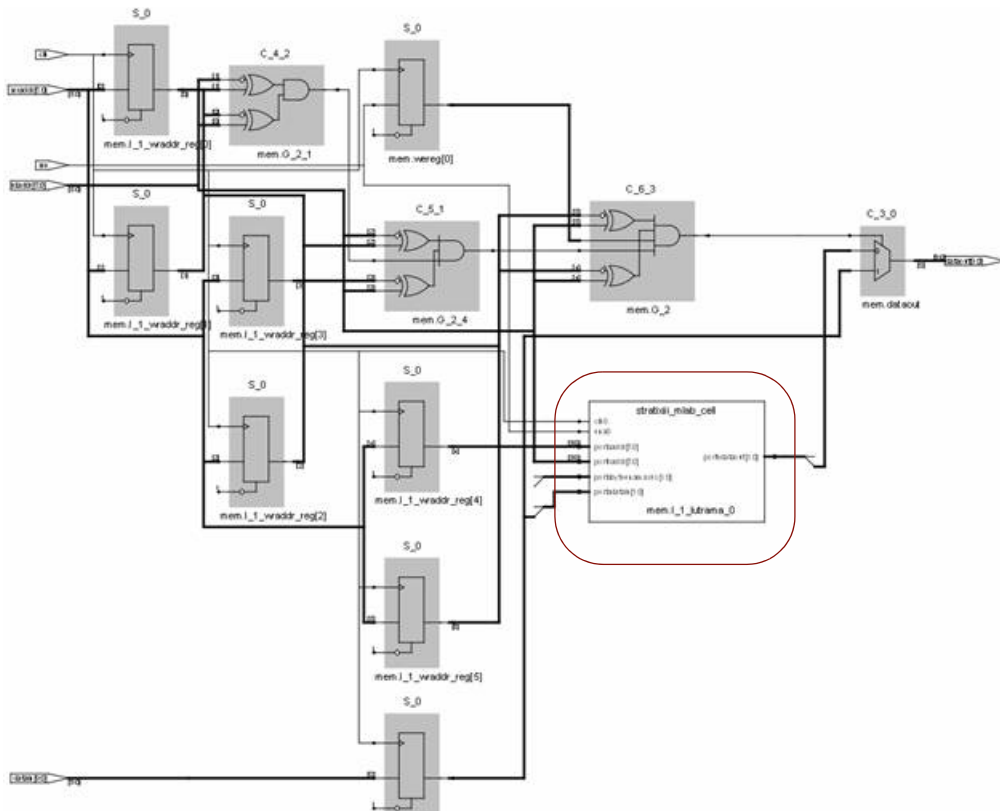
The software maps asynchronous RAMs to LUTRAM, and reports resource utilization in the log file, like this example:

```
Memory ALUTs: 10 (0% of 19000)
```

The following shows how the software maps an SDPRAM with registered output and asynchronous read to a simple dual-port RAM in the RTL view:



The following shows how the same memory is mapped in the Technology view to a `stratixiii_mlab_cell` LUTRAM component:



## Inferring Xilinx Block RAMs Using Registered Addresses

There are two ways to infer block RAMs in Xilinx Virtex designs: using registered addresses and using registered output. For information about the latter, see [Inferring Xilinx Block RAMs Using Registered Output, on page 393](#). The following procedure shows you how to set up your code with an explicit read address register.

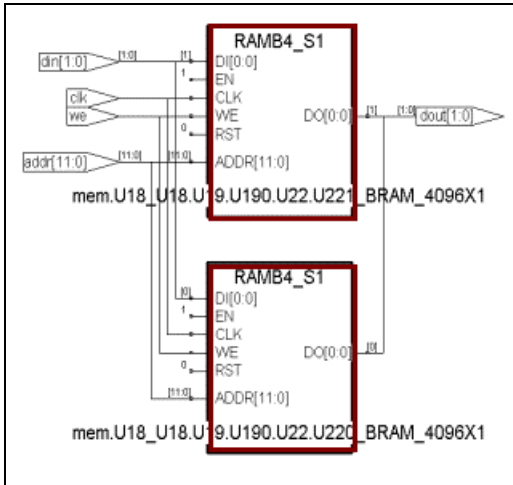
The software does not currently infer block RAMs for Virtex designs automatically; you have to use an attribute. It inserts bypass logic to resolve a read/write behavior difference between the RTL and post-synthesis gate-level simulations. It inserts the glue logic because it does not know the output at the read port when the read address and the write address access the same memory location.

1. Use instantiation instead of inference in the following cases where the software currently does not infer the RAMs:
  - RAMs with enable signals, RAM resets, or initialization settings.
  - Inaccessible pins: read enable pins are always active, and reset pins are always inactive.
  - Dual-port RAMs with read/write on a port.
2. For single-port RAM, do the following:
  - Make sure the read and write clocks are the same.
  - Make sure the read and write addresses are the same.
  - Make sure the enable signals are the same. Use only write enable signals.
  - Register the address, as shown in the following code:

```
always @(posedge clk)
 if(we)
 mem[addr] = din;

always @(posedge clk)
 addr_reg = addr;

assign dout = mem[addr_reg]
```



- To forward-annotate initialization values, use the Xilinx INIT property, as described in [Initializing Xilinx RAM, on page 404](#).

3. For dual-port RAM, do the following:

- Register the address as shown in this code:

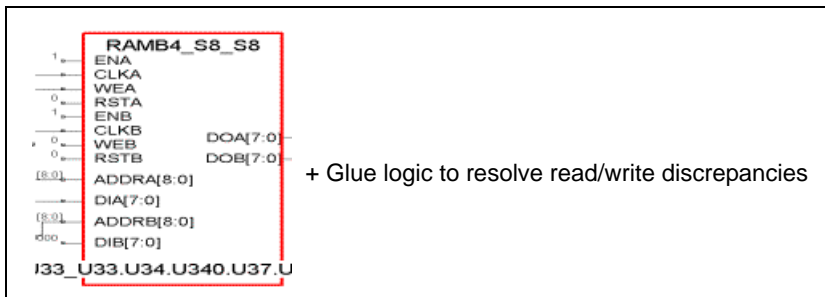
```

always @(posedge clk)
 if (we)
 mem[waddr] = din;

always @(posedge clk)
 raddr_reg = raddr;

assign dout = mem[raddr_reg]

```



- To forward-annotate initialization values, see [Initializing Xilinx RAM, on page 404](#).
- 4. To prevent the insertion of glue logic, add the `syn_ramstyle="no_rw_check"` attribute.

By default, the software inserts glue logic when the read and write addresses access the same memory location, because it does not know the output of the read port. The glue logic prevents a mismatch between the RTL and post-synthesis simulation results. See [Specifying RAM Implementation Styles, on page 378](#) or the *Reference Manual* for more information about this attribute.

- 5. To infer Virtex block RAM, add the `syn_ramstyle="block_ram"` attribute to the register signal in your source code, or to the output signal of the RAM in the SCOPE window. See [Specifying RAM Implementation Styles, on page 378](#) or the *Reference Manual* for more information about this attribute.
- 6. Run synthesis.

The software implements the circuit using Xilinx `RAMB4_S<n>_S<n>` primitives. The Xilinx dual-port block RAM is implemented with one write port.

## Inferring Xilinx Block RAMs Using Registered Output

For Virtex-II and Virtex-II Pro designs, you can code block RAMs with registered output as described here, or with registered addresses (see [Inferring Xilinx Block RAMs Using Registered Addresses, on page 391](#)). For information about forward-annotating initialization values, see [Initializing Xilinx RAM, on page 404](#).

This information is organized into these subtopics:

- [Advantages of Using Registered Output, on page 394](#)
- [Block RAM Mapping for Virtex-II Write Modes, on page 394](#)
- [Xilinx Single-Port Example with Registered Output, on page 396](#)
- [Xilinx Single-Output Dual-Port Example with Registered Output, on page 398](#)

## Advantages of Using Registered Output

The registered output method allows you to use reset and enable lines as well as the different write modes in Virtex architectures. The following table shows the advantages of using registered output instead of registered addresses:

| Registered Read Address                | Registered Output                 |
|----------------------------------------|-----------------------------------|
| Read and write clocks must be the same | Can have different clocks         |
| No enable, reset supported             | Supports enable and reset signals |

## Block RAM Mapping for Virtex-II Write Modes

The following table summarizes how the software implements Block RAM for the write modes in different Virtex families when you register the outputs. See [Xilinx Single-Port Example with Registered Output, on page 396](#) and [Xilinx Single-Output Dual-Port Example with Registered Output, on page 398](#) for examples.

|                                                | Virtex     | Virtex-E   | Virtex-II | Virtex-II Pro |
|------------------------------------------------|------------|------------|-----------|---------------|
| WRITEFIRST Mode                                |            |            |           |               |
| With enable and reset, enable takes precedence | SP         | SP         | SP        | SP            |
| With enable and reset, reset takes precedence  | SP         | SP         | SP        | SP            |
| Without enable                                 | SP         | SP         | SP        | SP            |
| Without reset                                  | SP         | SP         | SP        | SP            |
| Without enable or reset                        | SP         | SP         | SP        | SP            |
| READFIRST Mode                                 |            |            |           |               |
| With enable and reset, enable takes precedence | Select RAM | Select RAM | SP        | SP            |
| SP: Single-port block RAM                      |            |            |           |               |
| DP: Single-output, dual-port block RAM         |            |            |           |               |

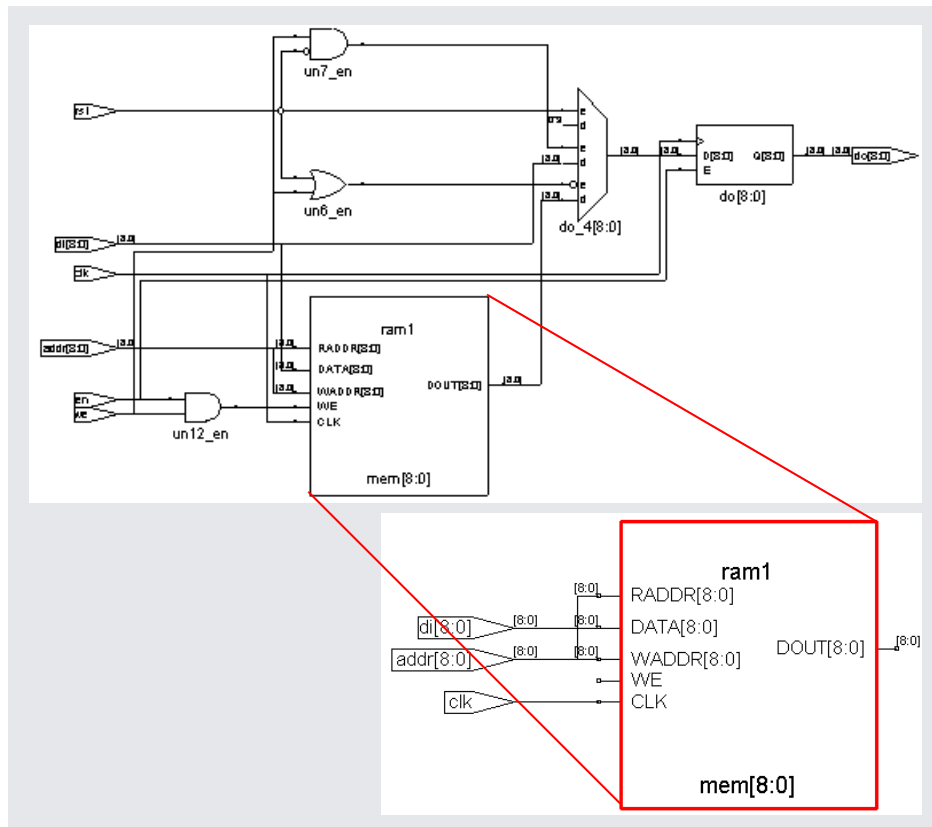
|                                                | <b>Virtex</b> | <b>Virtex-E</b> | <b>Virtex-II</b> | <b>Virtex-II Pro</b> |
|------------------------------------------------|---------------|-----------------|------------------|----------------------|
| With enable and reset, reset takes precedence  | Select RAM    | Select RAM      | SP               | SP                   |
| Without enable                                 | Select RAM    | Select RAM      | SP               | SP                   |
| Without reset                                  | Select RAM    | Select RAM      | SP               | SP                   |
| Without enable or reset                        | Select RAM    | Select RAM      | SP               | SP                   |
| <b>NOCHANGE Mode</b>                           |               |                 |                  |                      |
| With enable and reset, enable takes precedence | DP            | DP              | SP               | SP                   |
| With enable and reset, reset takes precedence  | DP            | DP              | SP               | SP                   |
| Without enable                                 | DP            | DP              | SP               | SP                   |
| Without reset                                  | DP            | DP              | SP               | SP                   |
| Without enable or reset                        | DP            | DP              | SP               | SP                   |

SP: Single-port block RAM

DP: Single-output, dual-port block RAM

## Xilinx Single-Port Example with Registered Output

This example shows the single-port 257x9 RAM with reset and enable extracted from the following code, where the output is registered. To forward-annotate initialization values, use the Xilinx INIT property as described in [Initializing Xilinx RAM, on page 404](#).





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

entity ramtest is port(
do : out std_logic_vector(8 downto 0);
addr : in std_logic_vector(8 downto 0);
di : in std_logic_vector(8 downto 0);
en,clk,we,rst : in std_logic);

end ramtest;

architecture beh of ramtest is
type memtype is array (256 downto 0) of std_logic_vector(8 downto 0);
signal mem : memtype;
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "block_ram";

begin
process(clk)
begin
if clk'event and clk='1' then
if(en='1') then
if (rst='1') then
do <= "000000000";
elsif (we='1') then
do <= di;
else
do <= mem(CONV_INTEGER(addr));
end if;
end if;
end if;
end process;

process(clk)
begin
if clk'event and clk='1' then
if (en='1' and we='1') then
mem(CONV_INTEGER(addr)) <= di;
end if;
end if;
end process;

end beh;
```

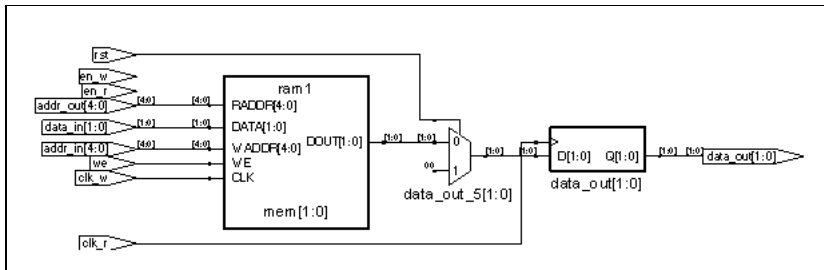
## Xilinx Single-Output Dual-Port Example with Registered Output

For Virtex and Virtex-II Pro designs, the software maps components to single-output dual-port block RAMs when the RAMs are coded with different read and write addresses, different read and write clocks, and different enable signals.

To forward-annotate initialization values, see [Initializing Xilinx RAM, on page 404](#).

In the following example, the read port has no enable, and the component is mapped to single-output dual-port block RAM:

```
always@(posedge clk_r)
 if(rst == 1)
 data_out = 0;
 else
 data_out = mem[addr_out];
always @(posedge clk_w)
 if (we) mem[addr_in] = data_in;
```



## Mapping Xilinx ROM to Block RAM

For Xilinx Virtex architectures, the software can map ROM into block RAM, provided you follow the guidelines in this procedure.

1. Place a dff register in front of the ROM, or place one of the following after the ROM:

| <b>Asynchronous</b> | <b>Synchronous</b>  |
|---------------------|---------------------|
| dff, dffe           |                     |
| dffr, dffre         | sdffr, sdffre       |
| dffs, dffse         | sdffs, sdffse       |
| dffpatr, dffpatre   | sdffpatr, sdffpatre |

where dffe is an enabled flip-flop, dffre is an enabled flip-flop with asynchronous reset, dffse is an enabled flip-flop with asynchronous set, and dffpatre is an enabled, vectored flip-flop with asynchronous reset pattern.

2. Ensure that the registers and ROMs are within the same hierarchy.
3. Ensure that the number of outputs of the candidate ROM is 64 or fewer.
4. Make sure that at least half the addresses possess assigned values. For example, in a ROM with ten address bits (1024 unique addresses), at least 512 of those unique addresses must be assigned values.
5. Specify the `syn_romstyle` attribute with the value set to `block_rom`.
6. Synthesize the design.

The software maps the ROM into block RAM.

# Initializing RAMs

You can specify startup values for RAMs and pass them on to the place-and-route tools. See the following for ways to set the initial values:

- [Initializing RAMs in Verilog](#), on page 400
- [Initializing RAMs in VHDL](#), on page 401
- [Initializing Xilinx RAM](#), on page 404

## 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 the 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). For Xilinx RAMs, you can alternatively initialize RAMs using the `INIT` property, as described in [Specifying the INIT Property for Xilinx RAMs \(Verilog\)](#), on page 405 and [Specifying the INIT Property with Attributes](#), on page 408.

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 653 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 | $readmemh ("fileName", memoryName) ;
```

Use `$readmemb` for a binary file and `$readmemh` for a hex file. For descriptions of the syntax, see [Initial Values in Verilog](#), on page 650 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] me_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 404](#).

See [RAM Initialization Example, on page 652](#) in the *Reference Manual* for an example of a Verilog single-port RAM.

## Initializing RAMs in VHDL

There are two ways to initialize RAMs in the VHDL code: with signal declarations or with variable declarations. For Xilinx RAMs, you can also initialize RAMs using the `INIT` property (see [Specifying the `INIT` Property for Xilinx RAMs \(VHDL\), on page 407](#) and [Specifying the `INIT` Property with Attributes, on page 408](#)).

### 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 403](#) and [Specifying the `INIT` Property for Xilinx RAMs \(VHDL\), on page 407](#).

```
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 := (
 "111111111111111111000000000000"
 , "111110011011101010011110001"
 , "111001111000111100101100111"
 , "110010110011101110011110001"
 , "1010011110001111111100110111"
 , "100000000000001111111111111"
 , "010110000111001111100110111"
 , "001101001100011110011110001"
 , "000110000111001100101100111"
 , "000001100100011010011110001"
 , "000000000000001000000000000"
 , "000001100100010101100001110"
 , "000110000111000011010011000"
 , "001101001100010001100001110"
 , "010110000111000000011001000"
 , "0111111111111111000000000000"
 , "101001111000110000011001000"
 , "110010110011100001100001110"
 , "111001111000110011010011000"
 , "111110011011100101100001110"
 , "111111111111111011111111111"
 , "111110011011101010011110001"
 , "111001111000111100101100111"
 , "110010110011101110011110001"
 , "1010011110001111111100110111"
 , "100000000000001111111111111"
 , 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 401](#) and [Initializing RAMs with \\$readmemb and \\$readmemh, on page 404](#).

```

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
 WRITE1_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 Xilinx RAM

In addition to the methods described in [Initializing RAMs in Verilog, on page 400](#) and [Initializing RAMs in VHDL, on page 401](#), you can forward-annotate RAM initialization values using the Xilinx INIT property. If you are using Verilog, you can also initialize the RAM using the \$readmemb and \$readmemh system tasks. See the following:

- [Initializing RAMs with \\$readmemb and \\$readmemh, on page 404](#)
- [Specifying the INIT Property for Xilinx RAMs \(Verilog\), on page 405](#)
- [Specifying the INIT Property for Xilinx RAMs \(VHDL\), on page 407](#)
- [Specifying the INIT Property with Attributes, on page 408](#)

Note the following differences between the methods:

- You can use the INIT property with any code. The \$readmemb and \$readmemh system tasks are only applicable in Verilog.
- The Verilog initial values only affect the output of the compiler, not the mapper. They ensure that the synthesis results match the simulation results and are not forward annotated.

### 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 653](#) 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 | $readmemh ("fileName", memoryName) ;
```

Use \$readmemb for a binary file and \$readmemh for a hex file. For details about the syntax, see [Initial Values in Verilog, on page 650](#) in the *Reference Manual*.



## Specifying the INIT Property for Xilinx RAMs (Verilog)

You can initialize and forward-annotate the values for Xilinx Verilog RAMs by specifying the INIT property. In Verilog, you can do this by using the `defparams` statement or by specifying the property in a global comment. The following examples illustrate these two methods.

- [Using defparam to Specify Initialization Values for Xilinx RAMs](#), on page 405
- [Using Global Comments to Specify Initialization Values for Xilinx RAMs](#), on page 406

### Using defparam to Specify Initialization Values for Xilinx RAMs

1. Include `defparam` statements in the Verilog file, using one statement for each word. Use the following syntax for the INIT property:

```
defparam name.INIT_XX=value;
```

*name*    Is the name of the RAM.

*xx*       Indicates the part of the RAM you are initializing. It can be any hex number from 00 to FF.

*value*    Sets the initialization value in hex.

The following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64 x 4 and 256 x 16 = 4K).

```
RAMB4_S4 pkt_len_ram_lo (
 .CLK (clock),
 .RST (1'b0),
 .EN (1'b1),
 .WE (we),
 .ADDR (address),
 .DI (data),
 .DO (q)
);

defparam pkt_len_ram_lo.INIT_00=
"00170016001500140013001200110010000f000e000d000c000b000a00090008 ";
defparam pkt_len_ram_lo.INIT_01=
"00270026002500240023002200210020001f001e001d001c001b001a00190018";
```

```
defparam pkt_len_ram_lo.INIT_02=
"00370036003500340033003200310030002f002e002d002c002b002a00290028" ;
...
defparam pkt_len_ram_lo.INIT_0F=
"0107010601050104010301020101010000ff00fe00fd00fc00fb00fa00f900f8" ;
```

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx P&R software in the EDIF netlist.

## Using Global Comments to Specify Initialization Values for Xilinx RAMs

### 1. Add the INIT property.

- Attach the INIT property to the instance as shown:

```
RAM /* synthesis INIT = "value" */

Block RAM /* synthesis INIT_xx = "value" */

```

- Define the INIT\_xx=*value* property as follows:

**xx**      Indicate the part of the RAM you are initializing with a number from 00 to FF.

**value**    Set the initialization value, in hex. You have 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

- Keep the entire statement on one line. Let your editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

### 2. For RAM, specify a hex value for the INIT statement as shown here:

```
RAM16X1S RAM1(...) /* synthesis INIT = "0000" */;
```

### 3. For Virtex block RAM, specify 16 different INIT statements. End the initialization data with a semicolon.

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

The following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64 x 4 and 256 x 16 = 4K).

```
RAMB4_S4 pkt_len_ram_lo (
 .CLK (clock),
 .RST (1'b0),
 .EN (1'b1),
 .WE (we),
 .ADDR (address),
 .DI (data),
 .DO (q)
)

/* synthesis
INIT_00="00170016001500140013001200110010000f000e000d000c000b000a00090008"
INIT_01="00270026002500240023002200210020001f001e001d001c001b001a00190018"
INIT_02="00370036003500340033003200310030002f002e002d002c002b002a00290028"
...
INIT_0F="0107010601050104010301020101010000ff00fe00fd00fc00fb00fa00f900f8"
*/;
```

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software.

## Specifying the INIT Property for Xilinx RAMs (VHDL)

### 1. Add the INIT property.

- Attach the INIT property to the label as shown:

```
RAM attribute INIT of object : label is "value";

Block RAM attribute INIT_xx of object : label is "value";

```

- Keep the entire statement on one line. Let the editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

### 2. For RAM, specify hex values for the INIT statement as shown:

```
attribute INIT of RAM1 : label is "0000";
```

3. For Virtex block RAM, specify 16 different INIT statements.

- Define the `INIT_xx=value` property as follows:

**xx**      Indicate the part of the RAM you are initializing with a number from 00 to FF.

---

**value**    Set the initialization value, in hex. You have 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

---

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

- End the initialization data with a semicolon.

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software.

## Specifying the INIT Property with Attributes

When you set the INIT property in the source code, it is difficult to pass on the values if the RAM instances are mapped to registers. When you specify INIT as an attribute, either in the SCOPE window or the constraint file, you are working with a mapped RAM, and the values are passed to the P&R tool. You can use this method to specify the initialization values for a RAM whether you are using Verilog or VHDL.

1. Compile and map the design.
2. Select the RAM in the SCOPE window.
  - Open SCOPE and go to the Attributes panel.
  - Open the Technology view. Drag and drop the RAM into the window.
3. Define the INIT (RAM) or `INIT_xx = value` (Block RAM) property in SCOPE. Alternatively you can edit the `.sdc` file using `define_attribute` statements.

**xx**      Indicates the part of the RAM you are initializing with a number from 00 to FF.

---

**value**    Sets the initialization value, in hex. You have 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

---

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

When you synthesize the design, the software forward-annotates the initialization values as constraints in the .sdc file. The following example shows a value of ABBADABAABBADABA defined for INIT\_00 and INIT\_01 on mem.mem\_0\_0 in the .sdc file:

```
define_attribute {i:mem.mem_0_0} INIT_00 {ABBADABAABBADABA}
define_attribute {i:mem.mem_0_0} INIT_01 {ABBADABAABBADABA}
```

These initialization values are forward-annotated as constraints to the Xilinx place-and-route software.

## Inferring Shift Registers

The software infers shift registers for Xilinx Virtex and Altera Stratix architectures. For Altera Stratix designs, you can implement the shift register as an `altshift_tap` with tap points. Use the following procedure.

1. Set up the HDL code for the sequential shift components. See [Shift Register Examples, on page 412](#) for examples.

Note the following:

- |        |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Altera | <ul style="list-style-type: none"> <li>• The registers must be dff or dffe registers.</li> <li>• All registers must have the same type and use the same control signals.</li> </ul>                                                                                                                                                                                                                                                            |
| Xilinx | <ul style="list-style-type: none"> <li>• The new component represents a set of three or more registers that can be shifted left (from a low address to a higher address).</li> <li>• The contents of only one register can be seen at a time, based on the read address.</li> <li>• For static components, the software only taps the output of the last register. The read address of the inferred component is set to a constant.</li> </ul> |

2. Specify the implementation style with the `syn_srlstyle` attribute.

| <b>syn_srlstyle Value</b> | <b>Implemented as...</b>                          |
|---------------------------|---------------------------------------------------|
| registers                 | registers                                         |
| select_srl                | Xilinx SRL16 primitives                           |
| no_extractff_srl          | Xilinx SRL16 primitives without output flip-flops |
| altshift_tap              | Altera Altshift_tap components                    |

You can set the value globally or on individual registers. For example, if you do not want the components automatically mapped to shift registers, set it globally to `registers`. You can then override this with specific settings on individual registers as needed.

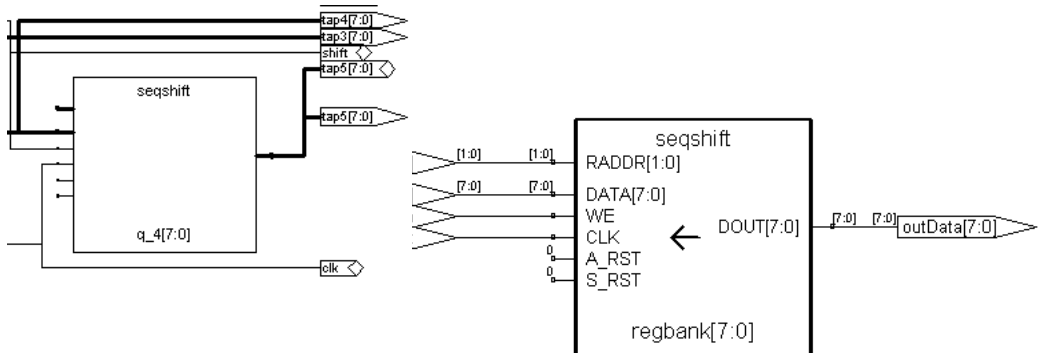
- For Altera shift registers, use attributes to control how the registers are packed:

| To...                                                                 | Attach...                                                                                                                                                 |
|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Prevent a register from being packed into shift registers             | <code>syn_useioff</code> or <code>syn_noprune</code> to the register. You can also use <code>syn_srlstyle</code> with a value of <code>registers</code> . |
| Prevent two registers from being packed into the same shift registers | <code>syn_keep</code> between the two registers. The algorithm slices the chain vertically, and packs the two registers into separate shift registers.    |
| Specify that two registers be packed in different shift registers     | <code>syn_srlstyle</code> with different group names for the registers you want to separate ( <code>syn_srlstyle= altshift_tap, group_name</code> )       |

If you do not specify anything, registers are packed across hierarchy. In all cases, the registers are not packed if doing so violates DRC restrictions.

- Run synthesis

After compilation, the software displays the components as `seqShift` components in the RTL view. The following figure shows the components in the RTL view.



In the technology view, the components are implemented as Xilinx SRL16 or Altera `altshift_tap` primitives or registers, depending on the attribute values you set.

5. Check the results in the log file and the technology file.

The log file reports the shift registers and the number of registers packed in them.. The following is an Altera example, showing the number of registers packed and taps inferred:

```
ShiftTap: 1 (10100 registers)
```

## Shift Register Examples

### Altera Shift Register (VHDL)

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

entity test is
port (
clk : in std_logic;
din : in std_logic_vector(31 downto 0);
dout : out std_logic_vector(31 downto 0);
tap7 : out std_logic_vector(31 downto 0);
tap6 : out std_logic_vector(31 downto 0);
tap5 : out std_logic_vector(31 downto 0);
tap4 : out std_logic_vector(31 downto 0);
tap3 : out std_logic_vector(31 downto 0);
tap2 : out std_logic_vector(31 downto 0);
tap1 : out std_logic_vector(31 downto 0)
);
end test;

architecture rtl of test is
type dataAryType is array(31 downto 0) of std_logic_vector(31
downto 0);
signal q : dataAryType;

begin
process (Clk)
begin
if (Clk'Event And Clk = '1') then
q <= (q(30 DOWNT0 0) & din);
end if;
end process;
```



```
dout <= q(31);
tap7 <= q(27);
tap6 <= q(23);
tap5 <= q(19);
tap4 <= q(15);
tap3 <= q(11);
tap2 <= q(7);
tap1 <= q(3);

end rtl;
```

### Altera Shift Register (Verilog)

```
module
test(dout, tap7, tap6, tap5, tap4, tap3, tap2, tap1, din, shift, clk);

output [7:0] dout;
output [7:0] tap7;
output [7:0] tap6;
output [7:0] tap5;
output [7:0] tap4;
output [7:0] tap3;
output [7:0] tap2;
output [7:0] tap1;
input [7:0] din;
input shift, clk;
reg [7:0] q[63:0];

integer n;

assign dout = q[63];
assign tap7 = q[55];
assign tap6 = q[47];
assign tap5 = q[39];
assign tap4 = q[31];
assign tap3 = q[23];
assign tap2 = q[15];
assign tap1 = q[7];

always @(posedge clk)
 if (shift)
 begin
 q[0] <= din;
 for (n=0; n<63; n=n+1)
 begin
```

```
 q[n+1] <= q[n];
 end
end
endmodule
```

## Xilinx Shift Register (VHDL)

This is a VHDL example of a shift register with no resets. It has four 8-bit wide registers and a 2-bit wide read address. Registers shift when the write enable is 1.

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

entity srltest is
 port (inData: std_logic_vector(7 downto 0);
 clk, en : in std_logic;
 outStage : in integer range 3 downto 0;
 outData: out std_logic_vector(7 downto 0)
);
end srltest;

architecture rtl of srltest is
 type dataAryType is array(3 downto 0) of std_logic_vector(7
downto 0);
 signal regBank : dataAryType;

begin
 outData <= regBank(outStage);
 process(clk, inData)
 begin
 if (clk'event and clk = '1') then
 if (en='1') then
 regBank <= (regBank(2 downto 0) & inData);
 end if;
 end if;
 end process;
end rtl;
```

## Xilinx Shift Register (Verilog)

```
module test_srl(clk, enable, dataIn, result, addr);
input clk, enable;
input [3:0] dataIn;
input [3:0] addr;
output [3:0] result;
reg [3:0] regBank[15:0];
integer i;

always @(posedge clk) begin
 if (enable == 1) begin
 for (i=15; i>0; i=i-1) begin
 regBank[i] <= regBank[i-1];
 end
 regBank[0] <= dataIn;
 end
end

assign result = regBank[addr];

endmodule
```

## Working with LPMs

Some technologies support LPMs (Library of Parameterized Modules), which are technology-independent logic functions that are parameterized for scalability and adaptability. There are two ways to instantiate LPMs in your source code: as black boxes, or by using prepared components.

The following table compares the methods for instantiating LPMs.

|                              | <b>Black Box Method</b> | <b>Verilog Library/VHDL Prepared Component Method</b> |
|------------------------------|-------------------------|-------------------------------------------------------|
| Applies to any LPM           | Yes                     | No                                                    |
| Synthesis LPM timing support | No                      | Yes                                                   |
| Synthesis procedure          | More coding             | Simple                                                |

See the following for more information about instantiating LPMs:

- [Instantiating Altera LPMs as Black Boxes](#), on page 417
- [Instantiating Altera LPMs Using VHDL Prepared Components](#), on page 421
- [Instantiating Altera LPMs Using a Verilog Library](#), on page 423

## Instantiating Altera LPMs as Black Boxes

The method described here uses either Verilog or VHDL LPMs in the Altera-prescribed megafunction format. Alternatively, you can use the methods described in [Instantiating Altera LPMs Using a Verilog Library, on page 423](#) or [Instantiating Altera LPMs Using VHDL Prepared Components, on page 421](#). For information about using Clearbox in Synplify Pro Stratix designs, see [Implementing Megafunctions with Clearbox Models, on page 165](#).

1. Generate the LPM using the Altera MegaWizard Plug-in Manager. If you generate the file using another method, make sure to use the same MegaWizard format, where ALTSYNCRAM is instantiated.

For examples of coding style, see [LPM Megafunction Example \(Verilog\), on page 417](#) and [LPM Megafunction Example \(VHDL\), on page 419](#).

2. Manually edit the LPM file and add the `syn_black_box` attribute to make the LPM a black box for synthesis.

See the examples in [LPM Megafunction Example \(Verilog\), on page 417](#) and [LPM Megafunction Example \(VHDL\), on page 419](#).

3. Instantiate the LPM in your design so that the LPM is not the top level. Synthesize the design.

The synthesis software treats the LPM as a black box. After synthesis, the software writes out a `.vqm` file where the module is a black box.

4. Add the original LPM file to the results directory and use it along with the `.vqm` file to place and route your design.

The place-and-route software uses the synthesized design information from the `.vqm` file and adds in the ALTSYNCRAM parameter information from the original megafunction file to place and route the LPM RAM correctly.

### LPM Megafunction Example (Verilog)

The following file shows the coding style the Altera MegaWizard uses to generate a Verilog LPM file, with the `syn_black_box` attribute added for synthesis.

```

module mylpm (
 data,
 wren,
 wraddress,
 rdaddress,
 clock,
 q)/* synthesis syn_black_box */;

 input [7:0] data;
 input wren;
 input [4:0] wraddress;
 input [4:0] rdaddress;
 input clock;
 output [7:0] q;

 wire [7:0] sub_wire0;
 wire [7:0] q = sub_wire0[7:0];

 altsyncram altsyncram_component (
 .wren_a (wren),
 .clock0 (clock),
 .address_a (wraddress),
 .address_b (rdaddress),
 .data_a (data),
 .q_b (sub_wire0));

 defparam
 altsyncram_component.operation_mode = "DUAL_PORT",
 altsyncram_component.width_a = 8,
 altsyncram_component.widthhad_a = 5,
 altsyncram_component.numwords_a = 32,
 altsyncram_component.width_b = 8,
 altsyncram_component.widthhad_b = 5,
 altsyncram_component.numwords_b = 32,
 altsyncram_component.lpm_type = "altsyncram",
 altsyncram_component.width_byteena_a = 1,
 altsyncram_component.outdata_reg_b = "UNREGISTERED",
 altsyncram_component.indata_aclr_a = "NONE",
 altsyncram_component.wrcontrol_aclr_a = "NONE",
 altsyncram_component.address_aclr_a = "NONE",
 altsyncram_component.address_reg_b = "CLOCK0",
 altsyncram_component.address_aclr_b = "NONE",
 altsyncram_component.outdata_aclr_b = "NONE",
 altsyncram_component.read_during_write_mode_mixed_ports
 = "DONT_CARE",
 altsyncram_component.ram_block_type = "AUTO",
 altsyncram_component.intended_device_family = "Stratix";

endmodule

```

## LPM Megafunction Example (VHDL)

Instantiate a file like this one at the top level, and include it in the project file, as shown in the preceding figure.

```

ENTITY myram IS
 PORT(
 data : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
 wren : IN STD_LOGIC := '1';
 wraddress : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
 rdaddress : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
 clock : IN STD_LOGIC ;
 q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END myram;

ARCHITECTURE SYN OF mylpram IS
 SIGNAL sub_wire0 : STD_LOGIC_VECTOR (7 DOWNTO 0);

 COMPONENT altsyncram
 GENERIC (
 operation_mode : STRING;
 width_a : NATURAL;
 widthad_a : NATURAL;
 numwords_a : NATURAL;
 width_b : NATURAL;
 widthad_b : NATURAL;
 numwords_b : NATURAL;
 lpm_type : STRING;
 width_byteena_a : NATURAL;
 outdata_reg_b : STRING;
 indata_aclr_a : STRING;
 wrcontrol_aclr_a : STRING;
 address_aclr_a : STRING;
 address_reg_b : STRING;
 address_aclr_b : STRING;
 outdata_aclr_b : STRING;
 read_during_write_mode_mixed_ports : STRING;
 ram_block_type : STRING;
 intended_device_family : STRING
);

```

```

PORT (
 wren_a : IN STD_LOGIC ;
 clock0 : IN STD_LOGIC ;
 address_a : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
 address_b : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
 q_b : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
 data_a : IN STD_LOGIC_VECTOR (7 DOWNTO 0)
);

END COMPONENT;

BEGIN
 <= sub_wire0(7 DOWNTO 0);

 altsyncram_component : altsyncram
 GENERIC MAP (
 operation_mode => "DUAL_PORT",
 width_a => 8,
 widthad_a => 5,
 numwords_a => 32,
 width_b => 8,
 widthad_b => 5,
 numwords_b => 32,
 lpm_type => "altsyncram",
 width_byteena_a => 1,
 outdata_reg_b => "CLOCK0",
 indata_aclr_a => "NONE",
 wrcontrol_aclr_a => "NONE",
 address_aclr_a => "NONE",
 address_reg_b => "CLOCK0",
 address_aclr_b => "NONE",
 outdata_aclr_b => "NONE",
 read_during_write_mode_mixed_ports => "DONT_CARE",
 ram_block_type => "AUTO",
 intended_device_family => "Stratix"
)

 PORT MAP (
 wren_a => wren,
 clock0 => clock,
 address_a => wraddress,
 address_b => rdaddress,
 data_a => data,
 q_b => sub_wire0
);

END SYN;

```



## Instantiating Altera LPMs Using VHDL Prepared Components

Prepared LPM Components, available as component declarations in LPM\_COMPONENTS, use generics instead of attributes to specify different design parameters. After you specify library and use clauses, instantiate the components and assign (map) the ports and the values for the generics. Refer to the Max+Plus II or Quartus II documentation for ports and generics that require mapping. See [Prepared LPM Components Provided by Synopsys \(VHDL\)](#), on page 1345 in the *Reference Manual* for a list of available LPM components.

The prepared components method is the simplest to use, but it does not cover all available LPMs. For other methods, see [Instantiating Altera LPMs as Black Boxes](#), on page 417, or [Instantiating Altera LPMs Using a Verilog Library](#), on page 423 (Altera only). The prepared components method uses generics instead of attributes to specify design parameters. You specify the library, instantiate the components, and assign (map) the ports and the values for the generics.

1. In the higher-level entity, specify the appropriate library and use clauses.

```
library lpm;
use lpm.components.all;
```

The prepared components are in the <install\_dir>\lib\vhd directory. The Altera LPM prepared components are in lpm.

2. Instantiate the prepared component.
3. Assign the ports and values for the generics. These assignments override the generic values in the library. Refer to the vendor documentation for details about ports and values for generics.

This is an example of an LPM instantiated at a higher level:

```
library ieee, lpm;
use ieee.std_logic_1164.all;
use lpm.components.all;

entity test is
 port(data : in std_logic_vector (5 downto 0);
 distance : in std_logic_vector (7 downto 0);
 result : out std_logic_vector (5 downto 0);
end test;
```

```

architecture arch1 of test is
begin
u1 : lpm_clshift
 generic map (LPM_WIDTH=>6, LPM_WIDTHDIST =>8)
 port map (data=>data, distance=>distance, result=>result);
end arch1;

```

## Prepared Components LPM Example (Altera)

This example shows the instantiation of the prepared component `lpm_ram_dq`:

```

library lpm;
use lpm.lpm_components.all;
library ieee;
use ieee.std_logic_1164.all;

entity lpm_inst is
 port (clock, we: in std_logic;
 data : in std_logic_vector(3 downto 0);
 address : in std_logic_vector(3 downto 0);
 q : out std_logic_vector (3 downto 0));
end lpm_inst;

architecture arch1 of lpm_inst is
begin
 I0 : lpm_ram_dq
 generic map (LPM_WIDTH => 4,
 LPM_WIDTHHAD => 4,
 LPM_TYPE => "LPM_RAM_DQ")
 port map (data => data,
 address => address,
 we => we,
 inclock => clock,
 outclock => clock,
 q => q);
end arch1;

```

## Instantiating Altera LPMs Using a Verilog Library

For Altera LPMs, you can also instantiate LPMs from a Verilog library. For other methods of instantiating LPMs, see [Instantiating Altera LPMs as Black Boxes, on page 417](#) and [Instantiating Altera LPMs Using VHDL Prepared Components, on page 421](#).

1. Add the Verilog library file `<install_dir>/lib/altera/altera_lpm.v` to your project. The following shows the code for LPM\_RAM\_DP.

```
module lpm_ram_dp (q, data, wraddress, rdaddress, rdclock,
 wrclock,
 rdclken, wrclken, rden, wren) /*synthesis syn_black_box*/;

parameter lpm_type = "lpm_ram_dp";
parameter lpm_width = 1;
parameter lpm_widthhad = 1;
parameter numwords = 1<<lpm_widthhad;
parameter lpm_indata = "REGISTERED";
parameter lpm_outdata = "REGISTERED";
parameter lpm_rdaddress_control = "REGISTERED";
parameter lpm_wraddress_control = "REGISTERED";
parameter lpm_file = "UNUSED";
parameter lpm_hint = "UNUSED";

input [lpm_width-1:0] data;
input [lpm_widthhad-1:0] rdaddress, wraddress;
input rdclock, wrclock, rdclken, wrclken, wren, rden;
output [lpm_width-1:0] q;
endmodule //lpm_ram_dp
```

2. Instantiate the LPM in the higher-level module. For example:

```
module top(d, q1, wclk, rclk, wraddr, raddr, wren, rden,
 wrclken, rdclken) ;
parameter AWIDTH = 4;
parameter DWIDTH = 8;
parameter WDEPTH = 1<<AWIDTH;

input [AWIDTH-1:0] wraddr, raddr;
input [DWIDTH-1:0] d;
input wclk, rclk, wren, rden;
input wrclken, rdclken;
output [DWIDTH-1:0] q1;
```

```
lpm_ram_dp u1(.data(d), .wrclock(wclk), .rdclock(rclk), .q(q1),
 .waddress(wraddr), .rdaddress(rdaddr), .wren(wren),
 .rden(rden), .wrclken(wrclken), .rdclken(rdclken));
defparam u1.lpm_width = DWIDTH;
defparam u1.lpm_widthad = AWIDTH;
defparam u1.lpm_indata = "REGISTERED";
defparam u1.lpm_outdata = "REGISTERED";
defparam u1.lpm_wraddress_control = "REGISTERED";
defparam u1.lpm_rdaddress_control = "REGISTERED";
endmodule
```

For information about using the LPMs in Altera simulation flows, see [Using LPMs in Simulation Flows, on page 777](#).

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 9

# Specifying Design-Level Optimizations

---

This chapter covers techniques for optimizing your design using built-in tools or attributes. For vendor-specific optimizations, see [Chapter 18, \*Optimizing for Specific Targets\*](#). It describes the following:

- [Tips for Optimization](#), on page 426
- [Pipelining](#), on page 429
- [Retiming](#), on page 433
- [Preserving Objects from Optimization](#), on page 440
- [Optimizing Fanout](#), on page 446
- [Sharing Resources](#), on page 450
- [Inserting I/Os](#), on page 452
- [Optimizing State Machines](#), on page 453
- [Inserting Probes](#), on page 461
- [Working with Gated Clocks](#), on page 464
- [Optimizing Generated Clocks](#), on page 477

## Tips for Optimization

The software automatically makes efficient tradeoffs 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 426
- [Optimizing for Area](#), on page 427
- [Optimizing for Timing](#), on page 428

### 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 427](#). For timing optimization, see [Optimizing for Timing, on page 428](#).

- 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 428](#). General optimization tips are in [General Optimization Tips, on page 426](#).

- 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 446](#) for more information.
- Check 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. See [Sharing Resources, on page 450](#) 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 369](#).
- 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 369](#).
- 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 427](#). General optimization tips are in [General Optimization Tips, on page 426](#).

- 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. Use clock, false path, and multi-cycle path constraints to make the constraints realistic.
- 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 446](#) for information about setting limits.
- 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.
- If you are working in the Synplify Premier tool and performance and quality of results (QoR) are not essential to the application (as with early prototyping and “what if” scenarios), use the Fast Synthesis option. For details, refer to [Chapter 10, Fast Synthesis](#).



The Fast Synthesis option reduces the amount and number of mapper optimizations performed so that you get faster synthesis runtimes. Once the design has been evaluated with fast synthesis, the mapper optimization effort can be returned to its normal, default level for optimum performance/QoR evaluations. This option is only available with the Synplify Premier tool for devices from the Xilinx Virtex and Spartan families or the Altera Stratix families.

## Pipelining

The pipelining feature is only available in the Synplify Pro and Synplify Premier tools. 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 433](#) 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 429
- [Pipelining the Design](#), on page 430

### Prerequisites for Pipelining

The pipelining feature is only available in the Synplify Pro and Synplify Premier tools.

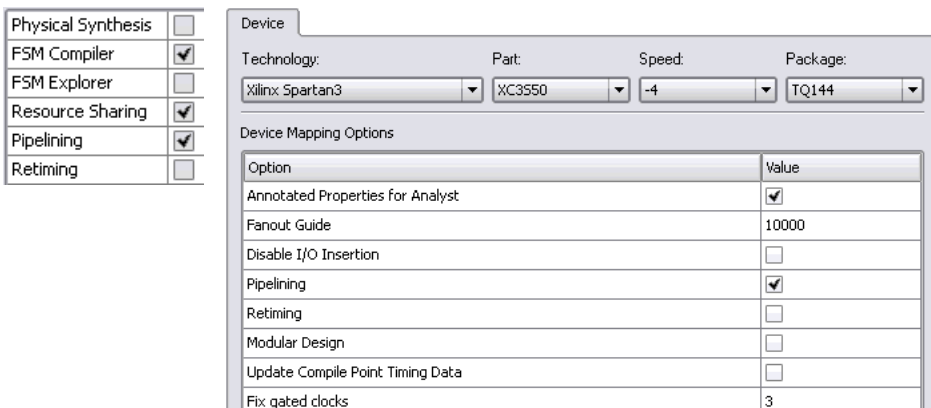
- Currently, pipelining is only supported for certain Actel, Altera, Lattice, and Xilinx technologies.
- In Xilinx Virtex designs, you can pipeline ROMs and multipliers. In Altera designs, you can pipeline multipliers, but not ROMs.
- For Xilinx Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-4 devices, you can only pipeline multipliers if the adjacent register has a synchronous reset.

- ROMs to be pipelined must be at least 512 words. Anything below this limit is too small.
- For Xilinx Virtex designs, you can push any kind of flip-flop into the module, as long as all the flip-flops in the pipeline have the same clock, the same set/reset signal or lack of it, and the same enable control or lack of it. For Altera designs, you must have asynchronous set/resets if you want to do pipelining.

## Pipelining the Design

The following procedure shows you techniques for pipelining.

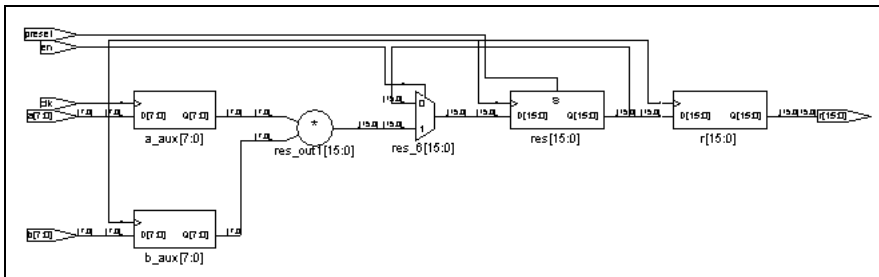
1. Make sure the design meets the criteria described in [Prerequisites for Pipelining](#), on page 429.
2. To enable pipelining for the whole design, check the Pipelining check box. from the button panel in the Project window, or with the Project->Implementation Options command (Device tab). The option is only available in the appropriate technologies.



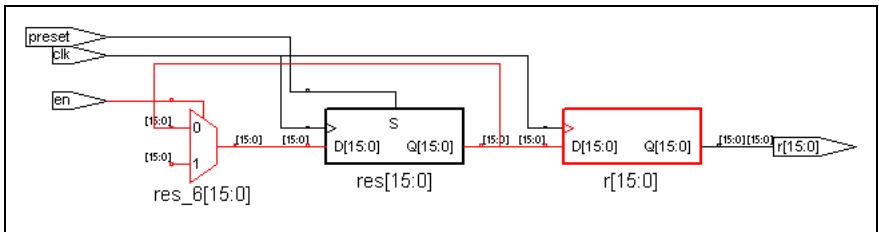
Use this approach as a first pass to get a feel for which modules you can pipeline. If you know exactly which registers you want to pipeline, add the attribute to the registers in the source code or interactively using the SCOPE interface.

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.



4. 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

The retiming feature is available in the Synplify Pro and Synplify Premier tools. Retiming is a technique for improving the timing performance of sequential circuits without having to modify source code. Retiming automatically moves registers (register balancing) across combinatorial gates or LUTs to improve 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 retiming algorithm retimes only edge-triggered registers. It does not retime level-sensitive latches. Currently you can use retiming for certain Actel, Altera, and Xilinx families. The option is not available if it does not apply to the family you are using.

These sections contain details about using retiming.

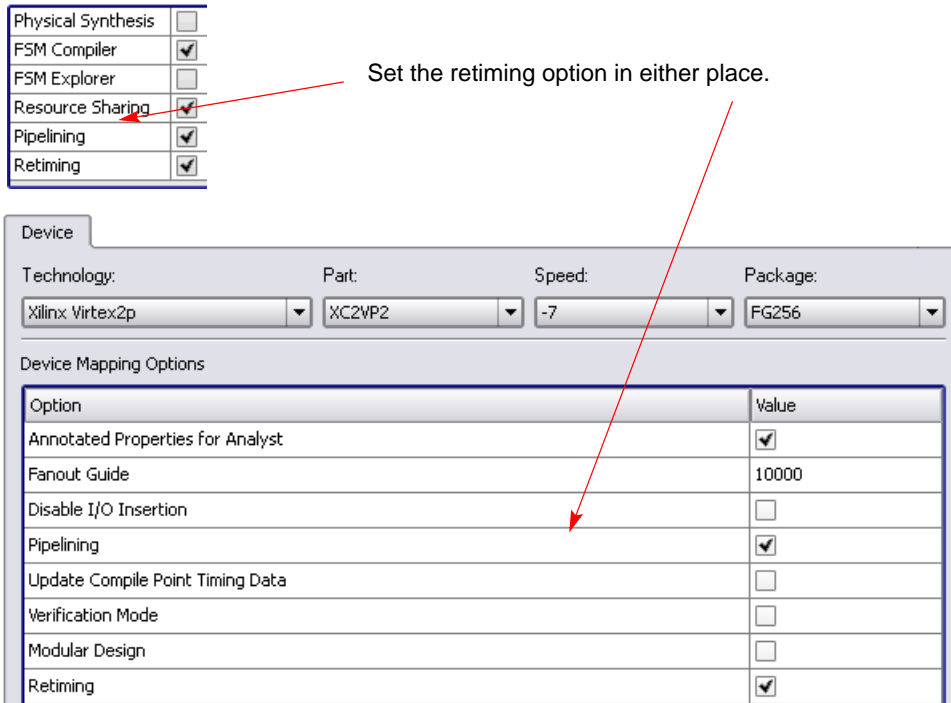
- [Controlling Retiming](#), on page 433
- [Retiming Example](#), on page 435
- [Retiming Report](#), on page 436
- [How Retiming Works](#), on page 437

## 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 (Device tab). The option is only available in certain technologies.



For Altera and Xilinx designs, retiming is a superset of pipelining, so when you select Retiming, you automatically select Pipelining. See [Pipelining, on page 429](#) for more information. For Actel designs, retiming does not include pipelining.

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_retiming` 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 437](#) for a list of the components the retiming algorithm will move.
  - Do not check the Retiming checkbox. Attach the `syn_allow_retiming` 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_retiming=1` on any registers implies `syn_pipeline=1`.

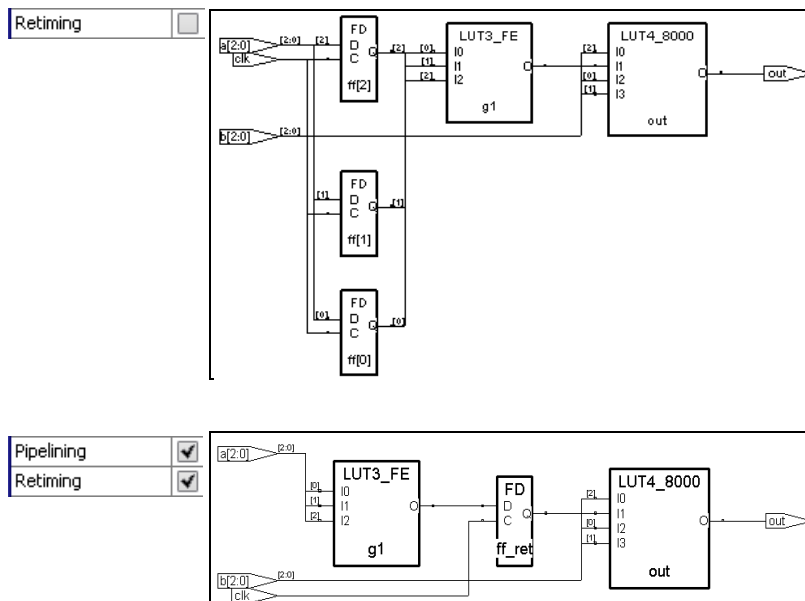
3. You can also fine-tune retiming using attributes:
  - To preserve the power-on state of flip-flops without sets or resets (FD or FDE) during retiming, set `syn_preserve=1` or `syn_allow_retiming=0` on these flip-flops.
  - To force flip-flops to be packed in I/O pads, set `syn_useioff=1` as a global 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 437](#) 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 435](#) for an example. The software honors other attributes you set, like `syn_preserve`, `syn_useioff`, and `syn_ramstyle`. See [How Retiming Works, on page 437](#) for details.

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 436](#) for more information about the report.

## Retiming Example

The following example shows a design with retiming disabled and enabled.



The top figure shows two levels of logic between the registers and the output, and no levels of logic between the inputs and the registers.

The bottom figure shows the results of retiming the three registers at the input of the OR gate. The levels of logic from the register to the output are reduced from two to one. The retimed circuit has better performance than the original circuit. Timing is improved by transferring one level of logic from the critical part of the path (register to output) to the non-critical part (input to register).

## 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). It lists some of the implications and results of retiming:

- Flip-flops with no control signals (resets, presets, and clock enables) are the most common type of component 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 combinatorial 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 combinatorial logic has `syn_ramstyle` set to registers, the registers can be retimed into the combinatorial logic.
- Retimed flip-flops are only moved through combinatorial 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. For Altera designs, registers that are in counter modes are not retimed to preserve the performance benefit of the counter mode.
- 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 effects of, or 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>define_reg_input_delay</code> and <code>define_reg_output_delay</code> constraints. Retimed registers affect timing constraints. |

| Attribute/Constraint      | Effect                                                                                                                                                                                                                                                                                                        |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.) |
| syn_hier=macro            | Does not retime registers in a macro with this attribute.                                                                                                                                                                                                                                                     |
| syn_keep                  | Does not retime across keepbufs generated because of this attribute.                                                                                                                                                                                                                                          |
| syn_hier=macro            | Does not retime registers in a macro with this attribute.                                                                                                                                                                                                                                                     |
| syn_pipeline              | Automatically enabled if retiming is enabled.                                                                                                                                                                                                                                                                 |
| syn_preserve              | Does not retime flip-flops with this attribute set.                                                                                                                                                                                                                                                           |
| syn_probe                 | Does not retime net drivers with this attribute. If the net driver is a LUT or gate, no flip-flops are retimed across it.                                                                                                                                                                                     |
| syn_reference_clock       | 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.                                                                                                                                  |
| syn_useioff               | 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.                                                     |
| syn_allow_retiming        | 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.

## How Retiming Works With Synplify Premier Regions

The following conditions can occur after a register has been retimed:

- If the retimed register and its driver and load remain in a Synplify Premier-specific region, then the register will remain in the region.
- If the retimed register is moved outside of a Synplify Premier-specific region but its load remains in the region, then the register will remain in the region.
- If the retimed register and its driver and load are moved outside a Synplify Premier-specific region, then the register will be moved outside the region.
- If the retimed register is moved to the boundary of a Synplify Premier-specific region, then tunneling can occur.
- Retiming may move a register across a Synplify Premier-specific region but not across combinatorial logic.

## Preserving Objects from Optimization

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 *Reference Manual*.

| To Preserve...          | Attach...                                                                                                                                                                    | Result                                                                                                                                                                                                                               |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nets                    | <code>syn_keep</code> on wire or reg (Verilog), or signal (VHDL).<br>For Actel designs (except 500K and PA), use <code>alspreserve</code> as well as <code>syn_keep</code> . | 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. This attribute is only applicable to the Synplify Pro and Synplify Premier software.                                                                                                             |
| Shared registers        | <code>syn_keep</code> on input wire or signal of shared registers                                                                                                            | Preserves duplicate driver cells, prevents sharing. See <a href="#">Using <code>syn_keep</code> for Preservation or Replication</a> , on page 441 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 441
- [Controlling Hierarchy Flattening](#), on page 444
- [Preserving Hierarchy](#), on page 444

## Using `syn_keep` for Preservation or Replication

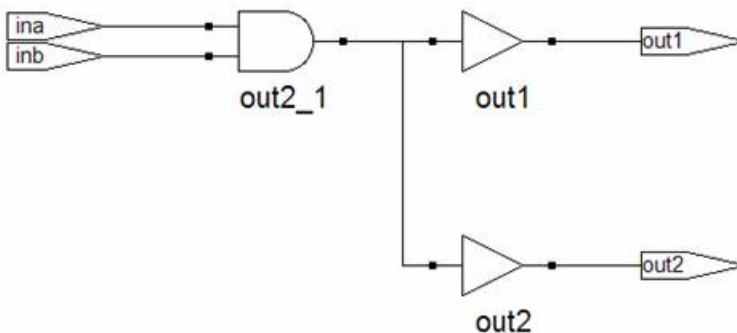
By default the tool considers replicated logic redundant, and optimizes it away. If you want to maintain the redundant logic, use `syn_keep` to preserve the logic that would otherwise be optimized away.

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.



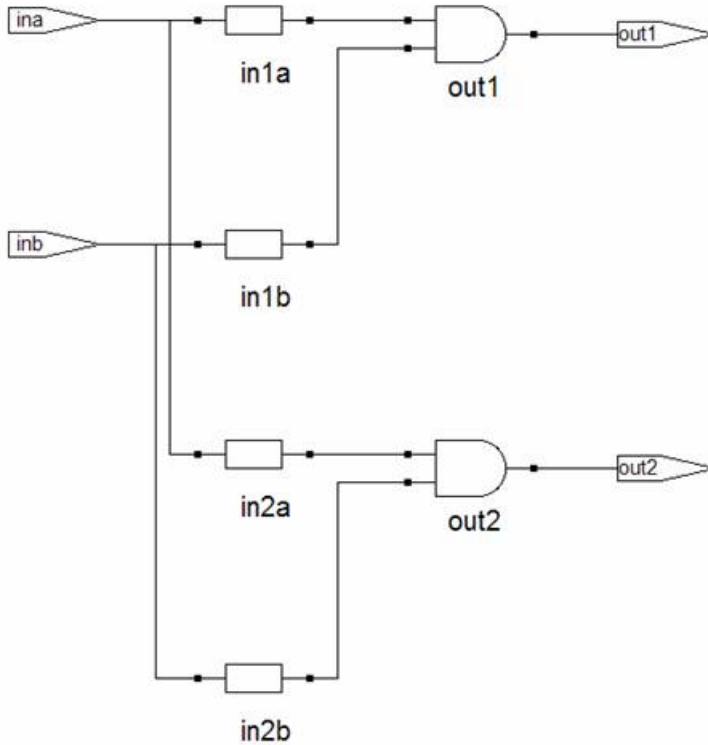
To replicate the AND gate in the previous example, apply `syn_keep` to the input wires, as shown below:

```
module redundant1d(ina,inb,out1,out2);
 input ina,inb;
 output out1,out2;
 wire out1;
 wire out2;

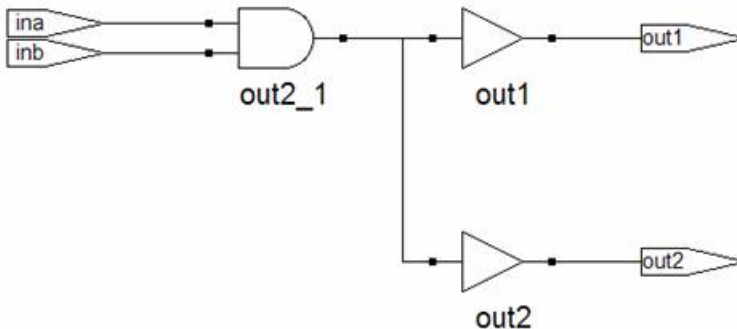
 wire in1a /*synthesis syn_keep = 1*/;
 wire in1b /*synthesis syn_keep = 1*/;
 wire in2a /*synthesis syn_keep = 1*/;
 wire in2b /*synthesis syn_keep = 1 */;

 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 444](#).

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...                     |
|--------------------------------------------------------------------------|------------------------------|
| 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>            |

The software flattens the design as directed. If there is a lower-level `syn_hier` attribute, it takes precedence over a higher-level one.

## 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 444](#).

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:

| <b>To...</b>                                                                                                                                  | <b>Value...</b> |
|-----------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Preserve the interface but allow cell packing across the boundary                                                                             | firm            |
| Preserve the interface with no exceptions (Actel, Altera, and Xilinx only)                                                                    | hard            |
| Preserve the interface and contents with no exceptions (Actel (except PA, 500K, and ProASIC3 families), Altera, Lattice, and QuickLogic only) | 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 use specify objects or hierarchy that you want to preserve during synthesis. For a complete list of all the directives and attributes, see the *Reference Manual*. This section describes the following:

- [Setting Fanout Limits](#), on page 446
- [Controlling Buffering and Replication](#), on page 448

### 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 in 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 448 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. For certain Actel technologies, you can set a global hard fanout limit by doing the following:
  - Select Project-> Implementation Options->Device and type a value for the Fanout Guide option, as described in the previous step.
  - On the same tab, check the Hard Fanout Limit option.

This makes the specified value a global hard fanout limit for the design.

3. 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 (including a global hard limit in Actel technologies). However, these limits still function as soft limits, and are replicated or buffered, as described in [Controlling Buffering and Replication, on page 448](#).

| 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.                                              |

4. 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 448](#).

5. 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`.

6. 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 446](#)) 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 combinatorial 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 in Altera Apex, Actel ProASIC (500K), ProASIC PLUS (PA) and ProASIC3 families, and QuickLogic pASIC3 designs                                                                                                           |
| <code>syn_replicate</code> is 1                     | <code>syn_replicate</code> is 0.<br>Note that the <code>syn_replicate</code> attribute must be used in conjunction with the <code>syn_maxfan</code> attribute for Actel families. The <code>syn_replicate</code> attribute is used only 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.
- To specify that high-fanout clock ports should not be buffered, set `syn_noclockbuf` globally, or on individual input ports. Use this if you want to save clock buffer resources for nets with lower fanouts but tighter constraints.
- In Xilinx designs, you can handle extremely large clock fanout nets by inserting a global buffer (BUFG) in your design. A global buffer reduces delay for a large fanout net and can free up routing resources for other signals.
- Turn off buffering and replication entirely, by setting `syn_maxfan` to a very high number, like 1000.

## Sharing Resources

One of the ways you can optimize area is to use resource sharing. 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.

1. Specify resource sharing globally for the whole design with one of the methods below. Enable the option to improve area; disable it to improve timing.
  - Select Project->Implementation Options->Options, and enable or disable Resource Sharing. Alternatively, enable Resource Sharing in the Project view.
  - Apply the `syn_sharing` directive to the top-level module or architecture in the source code. See [syn\\_sharing Directive](#), on page 1095 of the *Reference Manual* for syntax examples.

---

```
Verilog module top(out, in, clk_in) /* synthesis syn_sharing = "on" */;
```

---

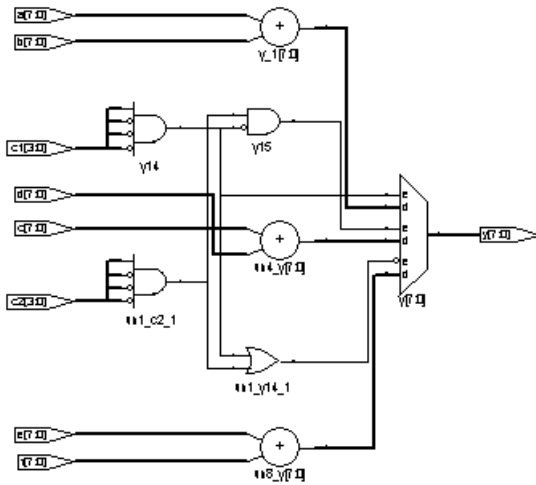
```
VHDL architecture rtl of top is
 attribute syn_sharing : string;
 attribute syn_sharing of rtl : architecture is "off";
```

---

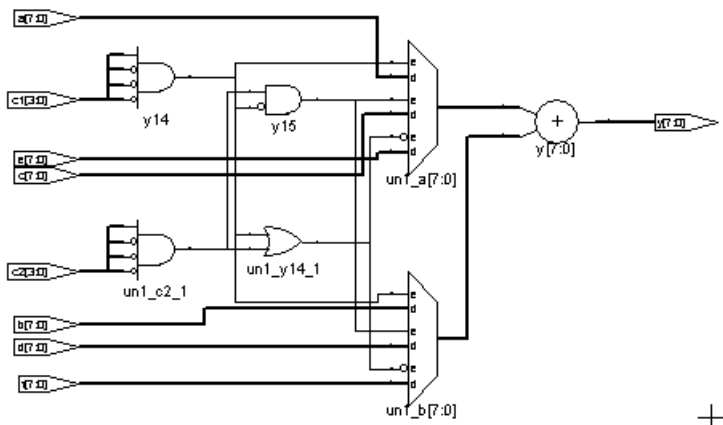
You cannot specify `syn_sharing` from the SCOPE interface, because it is a compiler directive.

2. To specify resource sharing on an individual basis, or to override the global setting, specify the `syn_sharing` attribute for the lower-level module/architecture, using the syntax described in the previous step.

Multiple adders with `syn_sharing` off.



Shared adder resource with `syn_sharing` on.



## Inserting I/Os

You can control I/O insertion globally, or on a port-by-port basis.

1. 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.
2. For Lattice designs, you can force I/O pads to be inserted for input ports that do not drive logic with the `syn_force_pads` attribute:
    - To force I/O pad insertion at the module level, set the `syn_force_pads` attribute on the module. Set the attribute value to 1. To disable I/O pad insertion at the module level, set the `syn_force_pads` attribute for the module to 0.
    - To force I/O pad insertion on an individual port, set the `syn_force_pads` attribute on the port with a value to 1. To disable I/O insertion for a port, set the attribute on the port with a value of 0.

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 and the FSM Explorer tools.

- **The Symbolic FSM Compiler**  
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.
- **The FSM Explorer**  
A specialized state machine optimizer that explores different encoding styles before selecting the best style. It uses the FSM Compiler to extract state machines, and runs the FSM Compiler automatically if it has not been run. The FSM Explorer functionality is only available in the Synplify Pro and Synplify Premier tools.

For more information, see the following:

- [Deciding when to Optimize State Machines](#), on page 453
- [Running the FSM Compiler](#), on page 455
- [Running the FSM Explorer](#), on page 458

## Deciding when to Optimize State Machines

The FSM Explorer and the FSM Compiler are automatic tools 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 369.

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. The FSM Viewer is only available in the Synplify Pro and Synplify Premier tools. For information about the FSM Viewer, see [Using the FSM Viewer, on page 670](#).

- To run the FSM Explorer

The FSM Explorer is a tool that examines all the encoding styles before selecting the best option, based on the state machine extraction done by the FSM Compiler. If the FSM Compiler has not been run previously, the Explorer automatically runs it. For more information about using the FSM Explorer, see [Running the FSM Explorer, on page 458](#).

If you are trying to decide whether to use the FSM Compiler or the FSM Explorer to optimize your state machines, remember these points:

- The FSM Explorer runs the FSM Compiler if it has not already been run, because it picks encoding styles based on the state machines that the FSM Compiler extracts.
- Like the FSM Compiler, you use the FSM Explorer to generate better results for your state machines. Unlike the FSM Compiler, which picks an encoding style based on the number of states, the FSM Explorer tries out different encoding styles and picks the best style for the state machine based on overall design constraints.
- The trade-off is that the FSM Explorer takes longer to run than the FSM Compiler.

## 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 455
- [Running the FSM Compiler on Individual FSMs](#), on page 456

### 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 369.

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, on page 670](#).
  - 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 369](#). 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 369](#). 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 in the log file, the RTL and technology views, and the FSM viewer, which is not available to Synplify users. For information about the FSM viewer, see [Using the FSM Viewer, on page 670](#).

## Running the FSM Explorer

1. If you need to customize the extraction process, set attributes.
  - Use `syn_state_machine=0` to specify state machines you do not want to extract and optimize.

---

```
Verilog reg [3:0] curstate /* synthesis state_machine */ ;
```

---

```
VHDL signal curstate : state_type;
 attribute syn_state_machine : boolean;
 attribute syn_state_machine of curstate : signal is true;
```

---

- Use `syn_encoding` if you want to set a specific encoding style.

---

```
Verilog reg [3:0] curstate /* synthesis syn_encoding="gray"*/ ;
```

---

```
VHDL signal curstate : state_type;
 attribute syn_encoding : string;
 attribute syn_encoding of curstate : signal is "gray";
```

---

The FSM Compiler honors the `syn_state_machine` attribute when it extracts state machines, and the FSM Explorer honors the `syn_encoding` attribute when it sets encoding styles. See [Specifying FSMs with Attributes and Directives, on page 369](#) for details.

2. Enable the FSM Explorer by checking the FSM Explorer 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 or Implementation Options buttons.

If you have not checked the FSM Compiler option, checking the FSM Explorer option automatically selects the FSM Compiler option.

3. Click Run to run synthesis.

The FSM Explorer uses the state machines extracted by the FSM Compiler. If you have not run the FSM Compiler, the FSM Explorer invokes the compiler automatically to extract the state machines, instantiate state machine primitives, and optimize them. Then, the FSM Explorer runs through each encoding style for each state machine that does not have a `syn_encoding` attribute and picks the best style. If you have defined an encoding style with `syn_encoding`, it uses that style.

The FSM Compiler writes a description of each state machine extracted and the set of reachable states for each state machine in the log file. The FSM Explorer adds the selected encoding styles. The FSM Explorer also generates a `<design>_fsm.sdc` file that contains the encodings and which is used for mapping.

4. Select View->View Log File and check the log file for the descriptions. The following extract shows the state machine and the reachable states as well as the encoding style, gray, set by FSM Explorer.

```
Extracted state machine for register cur_state
State machine has 7 reachable states with original encodings of:
 0000001
 0000010
 0000100
 0001000
 0010000
 0100000
 1000000
.....
Adding property syn_encoding, value "gray", to instance
cur_state[6:0]
List of partitions to map:
 view:work.Control(verilog)

Encoding state machine work.Control(verilog)-
cur_state_h.cur_state[6:0]
original code -> new code
 0000001 -> 000
 0000010 -> 001
 0000100 -> 011
 0001000 -> 010
 0010000 -> 110
 0100000 -> 111
 1000000 -> 101
```

5. 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, on page 670](#).



# Inserting Probes

The probe insertion feature is only available with the Synplify Pro and Synplify Premier tools. 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 462.

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 *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;
```

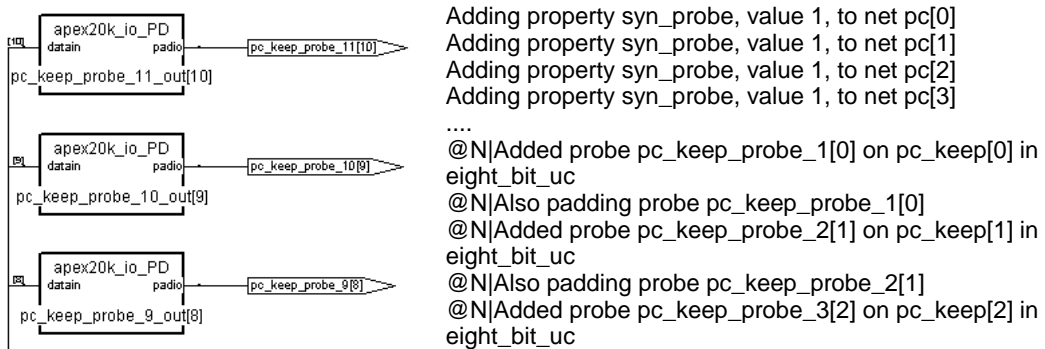
For detailed information about VHDL attributes and sample files, see the *Reference Manual*.

#### 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 461.

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.

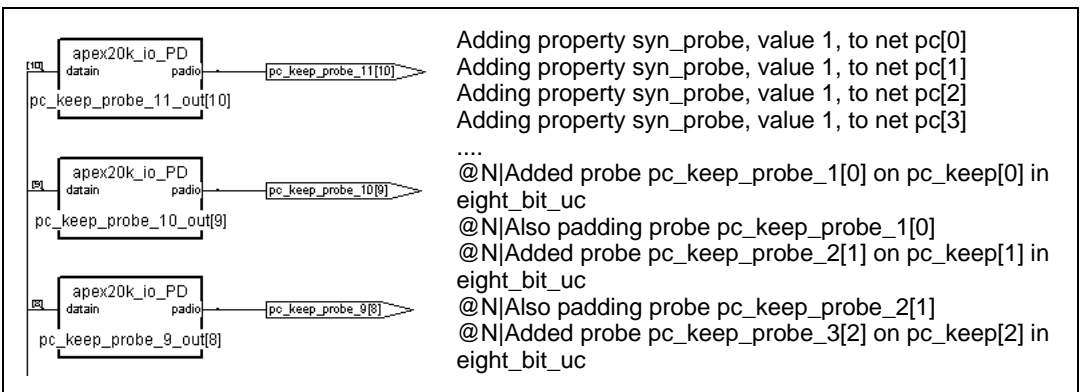
3. Do the following to add the attribute:
  - Drag the net into a SCOPE cell.

- 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.



## Working with Gated Clocks

The gated clock feature is only available in the Synplify Pro and Synplify Premier tools. This section first describes the gated clock solution, which is available for certain Altera, Lattice, and Xilinx technology families. The information is organized into these sections:

- [Gated Clocks in Synopsys FPGA Designs](#), on page 464
- [Prerequisites for Gated Clock Conversion](#), on page 467
- [Synthesizing a Gated Clock Design](#), on page 469
- [Using Gated Clocks for Black Boxes](#), on page 471
- [Analyzing Gated Clock Conversion Reports](#), on page 472
- [Restrictions on Using Gated Clocks](#), on page 475

### Gated Clocks in Synopsys FPGA Designs

ASIC designs typically gate clocks to conserve power, with custom clock trees defined for each individual tree. FPGA design has dedicated resources for low-skew clock nets. If an FPGA design implements a large number of customized clock trees on some other routing resource, it can result in clock skew and timing problems.

If you use the dedicated FPGA global clock trees instead, you free up routing resources and expedite 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.

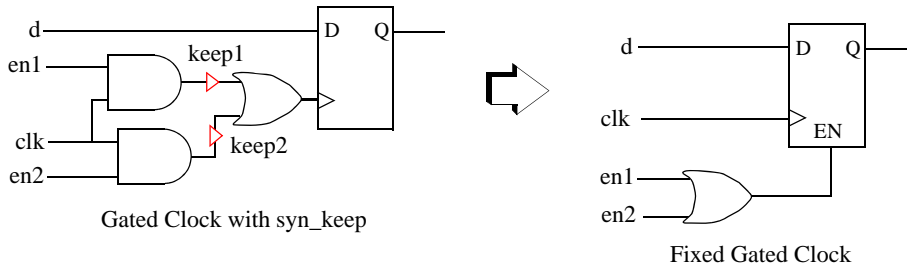
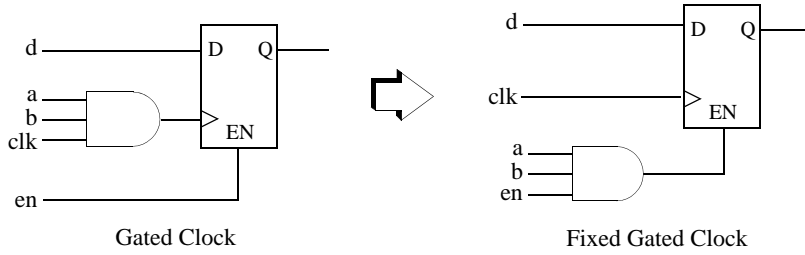
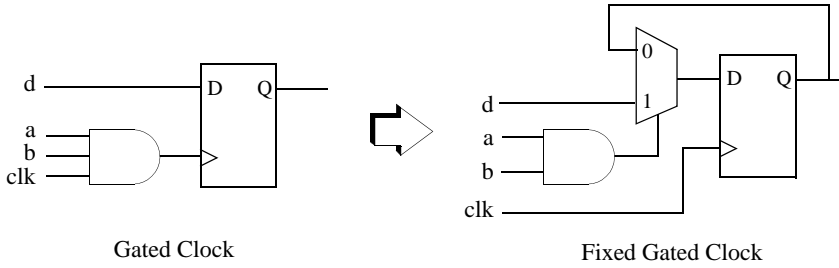
The solution is to separate the gating from the clock inputs, and combine individual clocks trees on the dedicated FPGA global clock trees. The software logically separates the gating from the clock and routes the gating to the clock enables on the sequential devices, using the programmable routing resources 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.

See the following figure for examples of eliminating gated clocks.



## 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 (.sdc) file. This example defines the clk input as the base clock.<br><br><pre>define_clock -name {clk} -freq 10.000 -clockgroup default_clkgroup</pre> |
| 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 black box. See <a href="#">Using Gated Clocks for Black Boxes, on page 471</a> .                |
| Correct logic format     | See <a href="#">Correct Logic Format, on page 467</a> for an example of the correct logic format.                                                                                                                                                                                                                                                                         |

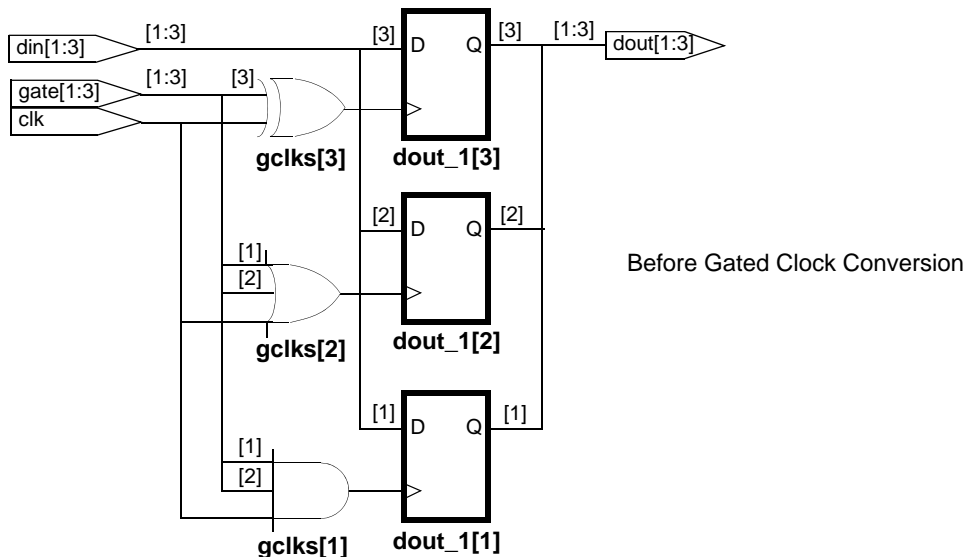
### Correct Logic Format

Specifically, the combinational logic for the gated clock must satisfy the following two conditions to have the correct format:

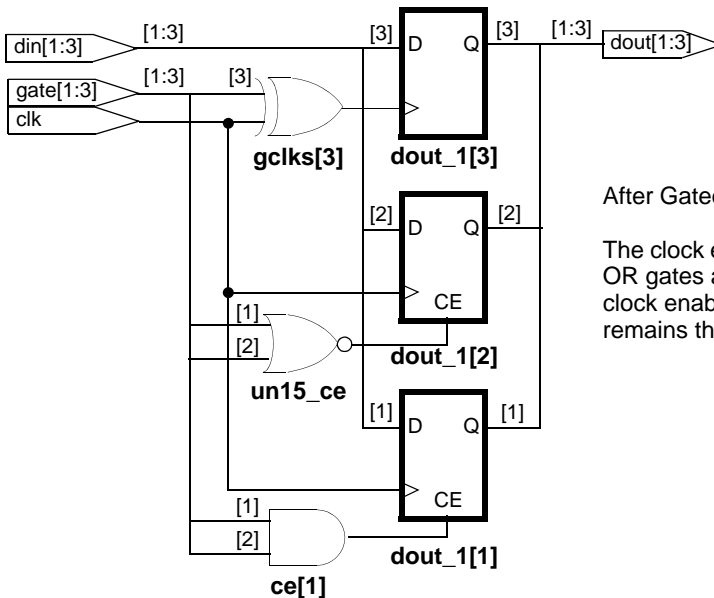
- For at least one set of gating input values, the value output for the gated clock must be constant and not change as the base clock changes.
- For at least one value of the base clock, changes in the gating input must not change the value output for the gated clock.

The correct logic format requirements are illustrated with the simple gates shown in the following figures. When the software synthesizes a design with the Fix Gated Clock 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<br>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<br>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<br>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 the same.

## Synthesizing a Gated Clock Design

The Fix Gated Clocks option described here is only available in the Synplify Pro and Synplify Premier tools for some Altera, Lattice, and Xilinx technology families.

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 467](#) for details.
2. If the gated clock drives a black box, specify the clock and the associated clock enable signal with the following directives: `syn_force_seq_prim`, `syn_isclock`, and `syn_gatedclk_en`. See [Using Gated Clocks for Black Boxes, on page 471](#) for details.
3. Make sure the clock net has a constraint specified in a `.sdc` file for the current implementation.

If you do not specify an explicit constraint on the clock net or set a global frequency constraint, enabling Fix Gated Clocks as described in the next step will not have any effect.

4. Enable the Fixed Gated Clocks option.
  - Select Project->Implementation Options.

Device

Technology:  Part:  Speed:  Package:

Device Mapping Options

| Option                           | Value                               |
|----------------------------------|-------------------------------------|
| Annotated Properties for Analyst | <input checked="" type="checkbox"/> |
| Fanout Guide                     | 10000                               |
| Disable I/O Insertion            | <input type="checkbox"/>            |
| Pipelining                       | <input checked="" type="checkbox"/> |
| Update Compile Point Timing Data | <input type="checkbox"/>            |
| Verification Mode                | <input type="checkbox"/>            |
| Modular Design                   | <input type="checkbox"/>            |
| Retiming                         | <input type="checkbox"/>            |
| Disable Sequential Optimizations | <input type="checkbox"/>            |
| Fix gated clocks                 | 3                                   |
| Fix generated clocks             | 3                                   |

- On the Device tab, set the value of Fixed Gated Clocks according to the kind of report you want to generate in the log file (see the following table).

### Value Effect

|   |                                                                        |
|---|------------------------------------------------------------------------|
| 1 | Does not report any gated clock conversions.                           |
| 2 | Only reports sequential elements that could not be converted.          |
| 3 | The default. Reports the conversion status of all sequential elements. |
| 0 | Disables the option.                                                   |

5. Synthesize the design.

The Fix Gated Clocks option works on 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 475 for additional information.

6. Check the results in the Gated Clock Report section of the log file. See [Analyzing Gated Clock Conversion Reports](#), on page 472 for an example of this report.

## Using Gated Clocks for Black Boxes

To fix gated clocks that drive black boxes, you must identify the clock and clock enable signal inputs to the black box. Use the `syn_force_seq_prim`, `syn_isclock`, and `syn_gatedclk_clock_en` directives to do this. Refer to the *Reference Manual* for information about these directives. You assume responsibility for their functionality.

The following is an example 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;

```

## Analyzing Gated Clock Conversion Reports

The value of the Fix Gated Clocks option determines how the conversions in the log file are reported:

| Value | Effect                                                                                     |
|-------|--------------------------------------------------------------------------------------------|
| 1     | Does not report any gated clock conversions.                                               |
| 2     | Only reports sequential elements that could not be converted.                              |
| 3     | The default. Reports the conversion status of all sequential elements. See example, below. |
| 0     | Disables the option.                                                                       |

For elements that could not be converted, the conversion also lists why the conversion did not occur.

## Example

When Fix Gated Clocks is set to 3 (all sequential elements reported), the report for the logic shown in [Correct Logic Format, on page 467](#) would look like this:

```

===== Gated clock report =====

The following instances have been converted
Seq Inst Clock

dout_1[2] clk_c
dout_1[1] clk_c
=====

The following instances have NOT been converted
Seq Inst Clock Reason for not converting

dout_1[3] G_8 Gating structure not compatible
=====

```

## Working with Gated Clock Error Messages

The following table describes the gated clock conversion error messages that are reported in the Gated Clock Report section of the log file. The following terms are used in the descriptions:

- *user clock* – the clock defined in the SDC file by the user
- *clock driver* – the driver to the clock pin of the sequential element

| Error Message                                 | Explanation                                                                                                                                 |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Added MUX in data path                        | The software added a MUX to the gated clock path because the sequential element did not have an equivalent gate with enable.                |
| Cannot convert primitive instance of the type | The software encountered a primitive in the gating logic that cannot be handled by gated clock conversion.                                  |
| Cannot find gated clock property              | The software cannot find a <code>syn_gatedclk_data_in</code> and/or <code>syn_gatedclk_data_out</code> property on the sequential instance. |
| Enable pin not found                          | There is no enable pin on an equivalent sequential element with enable.                                                                     |

| Error Message                                                                     | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Found combination loop involving the gating logic                                 | There is a combinational loop in the gating logic, which prevented gated clock conversion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Found unsupported combinational gate in gating logic                              | There is an instance in the gating logic that could not be handled currently by gated clock conversion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Gated clock does not have declared clock, add/enable clock constraint in SDC file | The user-defined clock signal is not defined in the SDC file, and this causes the gated clock conversion to fail.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Gated clock either has NO DRIVER or has MULTIPLE DRIVERS                          | <p>The gated clock conversion code cannot determine which clock to use because of one of the following:</p> <ul style="list-style-type: none"> <li>• There is no user clock driving the sequential element through the gating logic.</li> <li>• There are multiple user-defined clocks driving the gating logic.</li> </ul>                                                                                                                                                                                                                                                                                                                                                         |
| Gating structure creates Improper gating logic                                    | <p>The gating logic that corresponds to the sequential element could not be reduced to a form where it satisfies the following three rules needed for gated clock conversion:</p> <ul style="list-style-type: none"> <li>• For certain combinations of the gating signals, the gated clock signal must be capable of being disabled</li> <li>• For the remaining combinations of the gating signals, the gated clock signal equals either the clock signal or its inverted value</li> <li>• Finally, all gated clock signal transitions can only result from the clock signal transitions, and no enable signal transition can result in a gated clock signal transition</li> </ul> |
| Instance has no clock pin                                                         | The sequential gate does not have a clock pin.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Library cell is not marked as sequential                                          | The library cell has been marked as non sequential, with the property <code>syn_force_seq_prim</code> set to zero.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Multiple declared clocks found                                                    | There are multiple user-defined clocks in the gating logic.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| No gating logic found                                                             | There is no gating logic (this message is no longer displayed in the gated clock report).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Not in chip                                                                       | The clock driver is in another FPGA, not in the FPGA in which the sequential element is present.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

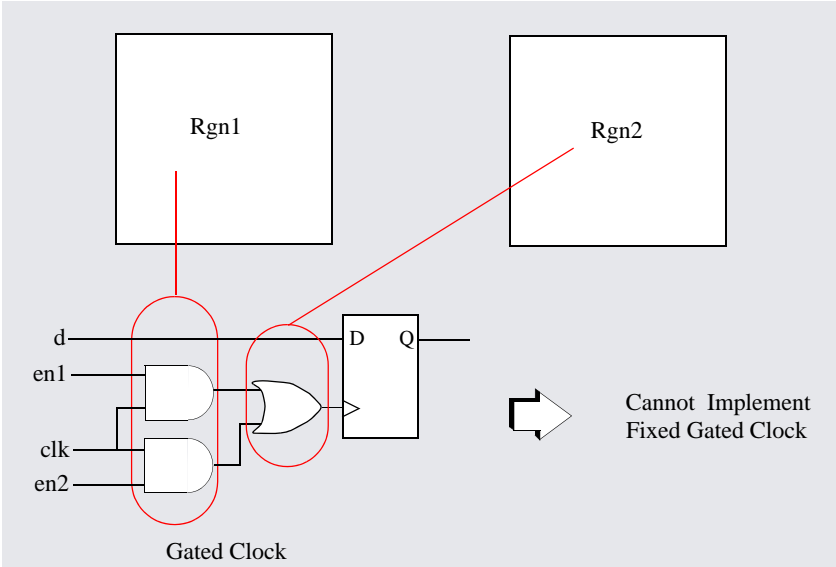
| Error Message                                                  | Explanation                                                                                                                                                                                                                                    |
|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Property <i>dontfixgatedclock</i> found                        | There is a <code>syn_dontfixgatedclock</code> on a sequential instance, which prevented gated clock conversion.                                                                                                                                |
| The width of the input not equal to the width of the output    | There is an input/output data width mismatch on the sequential element. This prevents the software from using a MUX-based feedback loop to enable gated clock conversion. The sequential element does not have an equivalent gate with enable. |
| Unknown reason                                                 | The software is unable to determine the reason why gated clock conversion is failing. Contact Synopsys Support.                                                                                                                                |
| User asserted <code>syn_keep</code> found on gated clock logic | There is a user-asserted <code>syn_keep</code> on one of the gates in the gating logic or one of the nets found in the gating logic. This prevented gated clock conversion.                                                                    |

## Restrictions on Using Gated Clocks

Currently, the Fix Gated Clocks option has the following restrictions:

- If the `syn_keep` attribute is assigned to a net, the Fix Gated Clocks option does not preserve this net during optimization. Refer to the third example in [Gated Clocks in Synopsys FPGA Designs, on page 464](#).
- The Fix Gated Clocks option cannot be implemented for inferred counters in Altera technologies.
- Gated clock conversion is not performed across hard and locked compile-point boundaries.
- The Fix Gated Clocks option cannot be implemented by the Synplify Premier tool if the gates associated with the gated clock are assigned to different Synplify Premier design plan regions. See the following figure.

The Fix Gated Clocks option *can* be applied if all gates associated with the gated clock are assigned to the same Synplify Premier design plan region. Also, the flip-flops can be assigned to any Synplify Premier region.





## Optimizing Generated Clocks

The tool has an option for generated clocks that is available for specific Altera, Xilinx, and Lattice families. When this 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.

### Generated-Clock Optimization Example

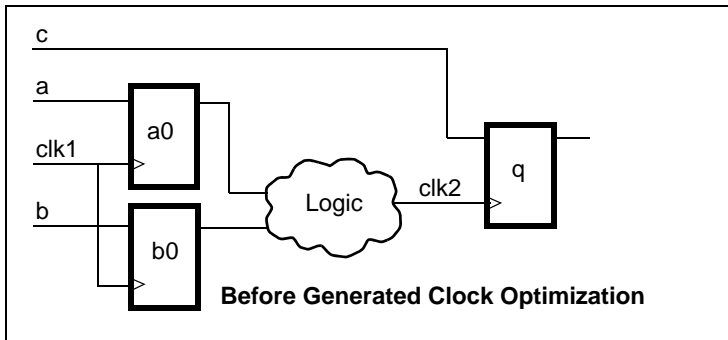
The following code segment is used to illustrate generated-clock optimization:

```
module gen_clk(clk1,a,b,c,q);
input clk1, a, b, c;
output q;
reg ao,bo,q;
wire en;

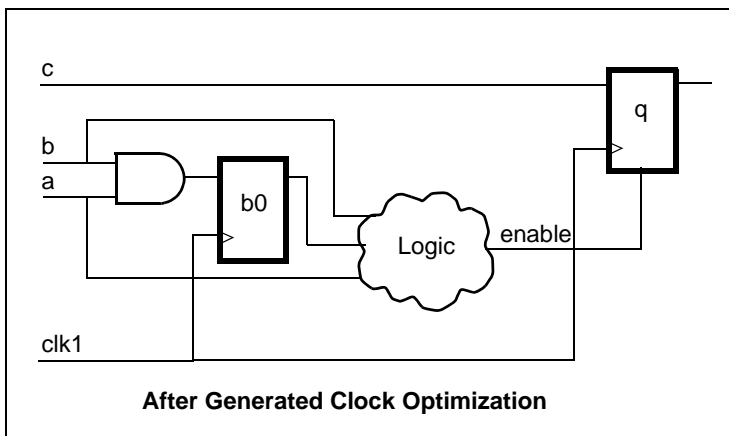
always @(posedge clk1)
begin
 ao <= a;
 bo <= b;
end
assign en = ao & bo;

always @(posedge en)
begin
 q <= c;
end
endmodule
```

With generated-clock optimization disabled (Fix Generated Clocks set to 0), 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 (Fix Generated Clocks set to 1, 2, or 3), 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.



## Enabling Generated-Clock Optimization

Generated-clock optimization is enabled by entering a non-zero value in the Fix Generated Clocks field in the Device tab of the Implementation Options dialog box. The following table describes the options.

| <b>Fix generated clock value</b> | <b>Description</b>                                                                  |
|----------------------------------|-------------------------------------------------------------------------------------|
| 0                                | Disable generated-clock optimization.                                               |
| 1                                | Perform optimization with no messages.                                              |
| 2                                | Perform optimization and report unoptimized sequential elements.                    |
| 3                                | The default. Perform optimization and report the status of all sequential elements. |

When a value of 2 or 3 is entered, the log file includes a generated clock optimization report.

## 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, such as a0 and b0 in the previous figure, cannot have an active set or reset.  
  
For example, if a0 has an active-low reset, then the reset must be disabled (tied 'high') for generated-clock optimization. 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.
4. With generated-clock optimization, you do not have to specify a primary clock.

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 10

# Fast Synthesis

---

The following describe how to use the Fast Synthesis feature in Synplify Premier:

- [About Fast Synthesis](#), on page 482
- [Using Fast Synthesis](#), on page 483

## About Fast Synthesis

Fast synthesis is a feature available in the Synplify Premier software. It is a logic synthesis design flow that is specific to Synplify Premier, like enhanced optimization or design-plan based logic synthesis.

What fast synthesis does is to significantly reduce synthesis runtimes by a factor of 2 or 3. It accomplishes this by reducing the number of optimizations performed, so there is a trade-off in performance.

You can use fast synthesis with designs targeting Altera Stratix and Xilinx Virtex and Spartan devices.

### When to Use Fast Synthesis

Fast synthesis is best used in situations where quality of results (QoR) is not crucial, or quick turnaround times are more important. Do not use this flow if performance is critical. The following list some situations where fast synthesis is effective:

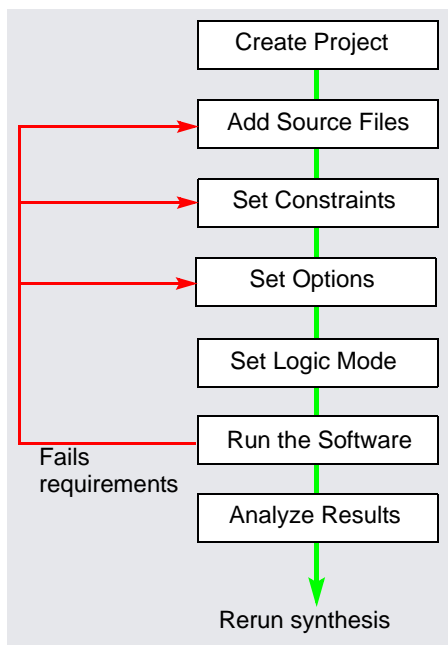
- In the initial design development phase, when you need to quickly evaluate a design or get a baseline result, and performance is secondary.
- When exploring "what if" scenarios when you have different implementations in mind for your design. In such a case, fast synthesis could save you time working through different runs.
- When you need a quick preliminary synthesis result to help get post-synthesis feedback.
- When prototyping a design (Altera Stratix and Xilinx Virtex and Spartan families only). This can speed up the process for ASIC prototype designers who are developing initial board-level implementations to verify the design.
- When you need to have quick RTL-to-board turnaround times for debug iterations.

# Using Fast Synthesis

This section describes how to run Synplify Premier fast synthesis.

## The Fast Synthesis Design Flow

The following figure summarizes the steps in the fast synthesis design flow. The steps are described in [Running Logic Synthesis with Fast Synthesis](#), on page 483.



## Running Logic Synthesis with Fast Synthesis

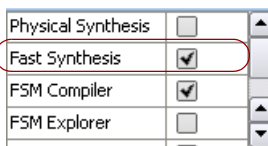
Use the following procedure for quick evaluations or where a faster runtime is more important than the quality of results.

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

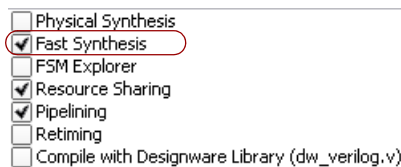
In general, timing attributes are not honored in Fast Synthesis mode.

4. Set options.
  - Set any options that you want in the Implementation Options dialog box.
  - Make sure that the Auto Constrain option is disabled.
5. Specify logic synthesis with fast synthesis.
  - Disable the Physical Synthesis option, either in the Project window or in the Implementation Options dialog box. This ensures that you are only running logic synthesis. You cannot run fast synthesis if the Physical Synthesis option is enabled.
  - In the Device panel of the Implementation Options dialog box, set Target to one of the supported Altera or Xilinx families.
  - Enable Fast Synthesis either in the Project view or the Options panel of the Implementation Options dialog box. This option is off by default, and you must explicitly enable it.

Project View



Implementation Options-&gt;Options



The tool reduces the amount and number of logic synthesis optimizations performed, and this results in faster runtimes. If you have both Fast Synthesis and Enhanced Optimization enabled (see [Logic Synthesis with Enhanced Optimization, on page 36](#)), the software ignores the Enhanced Optimization setting and runs fast synthesis.

- Click OK.
6. Click Run to run logic synthesis.
  7. Analyze the results, using the log file, the HDL Analyst schematic views, the Message window and the Log Watch window.

After you have analyzed the results of this preliminary run, you can do another fast synthesis run, or disable the Fast Synthesis option and repeat synthesis with a full-scale logic or physical synthesis run.

If Fast Synthesis is intended for quick synthesis results and not for a fast board implementation, it is recommended that you do not run P&R on the resulting netlist, as you might get sub-optimal QOR and longer P&R runtimes.



# Fast Synthesis and Other Synthesis Options

When you run fast synthesis, other optimizations can be affected.

| <b>Option</b>         | <b>Usage with Fast Synthesis</b>                                                                                                                                                                                                                        |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Auto Constrain        | Do not use this option with fast synthesis.                                                                                                                                                                                                             |
| Enhanced Optimization | Do not set this option with fast synthesis, as it will be ignored.                                                                                                                                                                                      |
| FSM Explorer          | You can use this option with fast synthesis.                                                                                                                                                                                                            |
| Physical synthesis    | You cannot run fast synthesis with physical synthesis enabled. Fast synthesis only operates in logic synthesis mode in the Synplify Premier tool. Fast synthesis does not generate any placement information.                                           |
| Pipelining            | You can use this option with fast synthesis.                                                                                                                                                                                                            |
| Resource Sharing      | You can use this option with fast synthesis.                                                                                                                                                                                                            |
| Retiming              | If you enable <b>Fast Synthesis</b> , the tool does not do any retiming optimizations.                                                                                                                                                                  |
| Timing constraints    | You can set any design constraints as you would normally do. However, if your goal is to shorten runtimes for a fast board implementation for example, it is recommended that you either use loose timing constraints or set the global clock to 1 Mhz. |

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

---

## CHAPTER 11

# Floorplanning with Design Planner

---

The Synplify Premier Design Planner tool lets you create a design plan to physically constrain portions of a design to specific regions on a device. It is important to place physical constraints carefully, and this tool helps you do this. For an overview of the design flow and steps, see [Design Plan-based Physical Synthesis, on page 48](#); the topics below describe how to use the tool.

Netlist restructure files usually contain primitives that have been bit sliced or modules that have been zippered. The design plan (.sfp) and netlist restructure files are used during optimization to improve the overall design performance.

The following describe the Design Planner tool, bit slicing, and zippering in more detail:

- [Using Design Planner](#), on page 488
- [Assigning Pins and Clocks](#), on page 495
- [Working with Regions](#), on page 505
- [Working with Altera Regions](#), on page 519
- [Working with Xilinx Regions](#), on page 523
- [Assigning Objects to Xilinx Regions](#), on page 527
- [Using Process-Level Hierarchy](#), on page 542
- [Bit Slicing](#), on page 543
- [Zippering](#), on page 550

# Using Design Planner

The Design Planner functionality is only available for certain Altera and Xilinx technologies. You can use the Design Planner tool either in combination with graph-based physical synthesis (*Graph-Based Physical Synthesis with Design Planner*, on page 46) or in a design plan-based physical synthesis flow (*Design Plan-based Physical Synthesis*, on page 48). The following describes the basics of using Design Planner.


- [Starting Design Planner](#), on page 488
- [Copying Objects in the Design Planner Tool](#), on page 490
- [Controlling Pin Display in the Design Plan Editor](#), on page 491
- [Creating and Using a Design Plan File for Physical Synthesis](#), on page 494

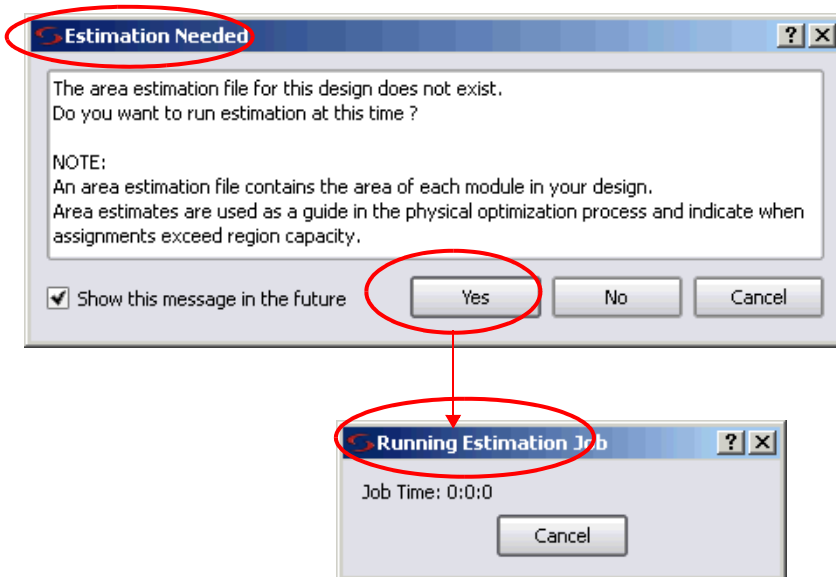
## Starting Design Planner

After the design is compiled, you create a design plan by doing the following:

1. Start with a compiled design.

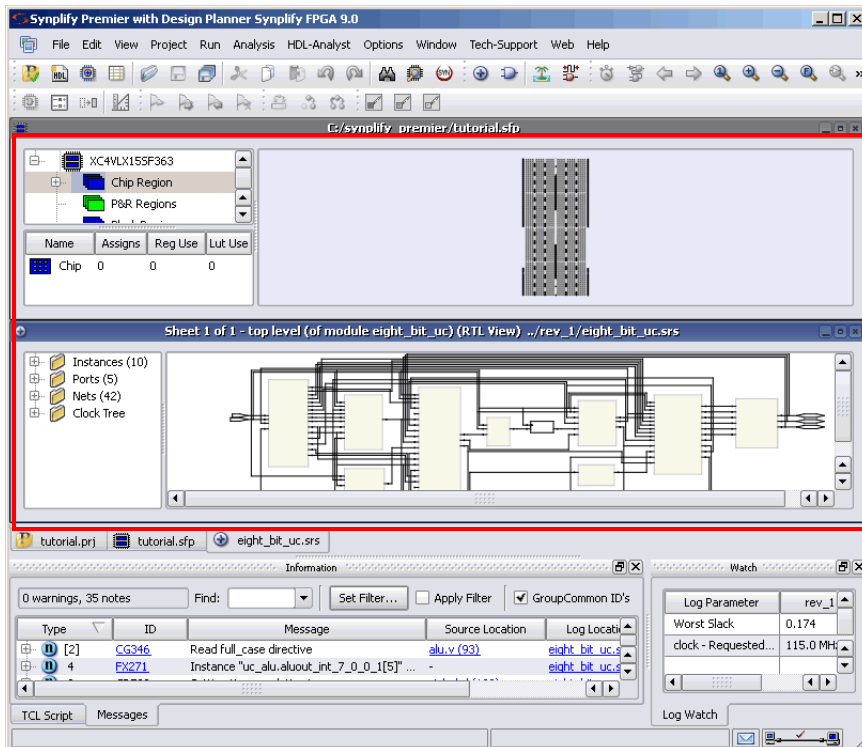
It is best if you run logic synthesis first to ensure that there are no errors before you start physical synthesis, but you can run Design Planner on a design that has just been compiled but not synthesized.

2. Click the New Design Plan icon () in the Project view. Alternatively, you can also create a new design plan file using File->New from the Project menu.
3. If you have not run area estimation, or the area estimation file is out-of-date, the Estimation Needed dialog box appears asking if you want to run estimation.
  - If you do not see this box, the No area estimate warning check box on the Assignments tab of Tools->Design Planner Preferences is disabled.
  - If you click No, the Design Planner is displayed.
  - If you click Yes, the tool first runs area estimation, and the Running Estimation dialog opens and displays the runtime of the job. Once estimation is complete, the Design Planner opens.



The following figure shows the Design Planner and RTL views.

### Design Plan Views



## Copying Objects in the Design Planner Tool

You can use the cut, copy, and paste functions in the Design Plan Editor and Design Plan Hierarchy Browser views instead of drag and drop. Note the following caveats:

- You can only use cut, copy, and paste on assignments (modules, primitives, and nets).
- You cannot cut or copy regions using the Design Planner tool and, you cannot paste to multiple regions.

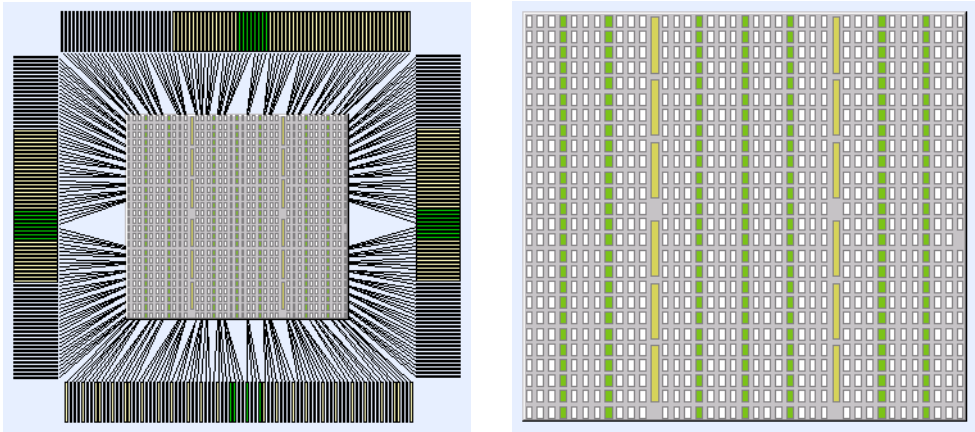
The following table summarizes the cut and paste operations.

| To...                                                                  | Do this...                                                                                                                                                                                                                               |
|------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Assign a module or primitive from a HDL Analyst view                   | <ul style="list-style-type: none"> <li>• Select the module/primitive in HDL Analyst and press <b>Ctrl-c</b> to copy it.</li> <li>• Select the destination region in Design Planner and paste with <b>Ctrl-v</b>.</li> </ul>              |
| Assign a net to an I/O block from HDL Analyst                          | <ul style="list-style-type: none"> <li>• Select the net in HDL Analyst and copy.</li> <li>• Select the I/O block region and paste it.</li> </ul>                                                                                         |
| Assign a module or primitive from the Hierarchy Browser Unassigned Bin | <ul style="list-style-type: none"> <li>• Select the module or primitive in the hierarchy browser and copy it.</li> <li>• Select the destination region and paste it.</li> </ul>                                                          |
| Replicate a module or primitive using the Hierarchy Browser            | <ul style="list-style-type: none"> <li>• Select the module or primitive within the region using the hierarchy browser and copy.</li> <li>• Select the destination region and paste. This displays the Replication dialog box.</li> </ul> |
| Move an assignment using the Hierarchy Browser view                    | <ul style="list-style-type: none"> <li>• Select the module or primitive in the region using the Hierarchy Browser, and cut it using <b>Ctrl-x</b>.</li> <li>• Select the destination region and paste.</li> </ul>                        |

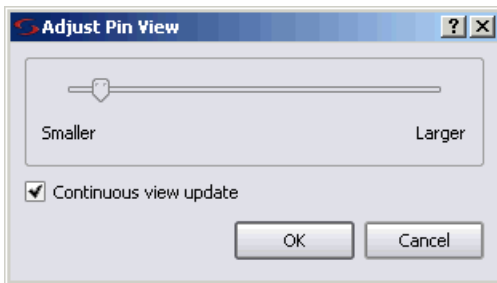
## Controlling Pin Display in the Design Plan Editor

The Design Plan Editor contains the device floorplan and region view in the Design Planner, and is available for Altera and Xilinx devices. It lets you view and assign external ports or internal nets to I/O pins on the device.

1. To expand the pin view, do the following:
  - Open the Design Planner and toggle on **View->Expanded Pin View**, or use **Ctrl-e**. This enables the expanded pin view in the Design Plan Editor. The following figure shows the enabled and disabled views for a design.



2. To adjust the size of the pins in the view, do the following:
  - Select View->Adjust Pin View... the Adjust Pin View dialog box appears.



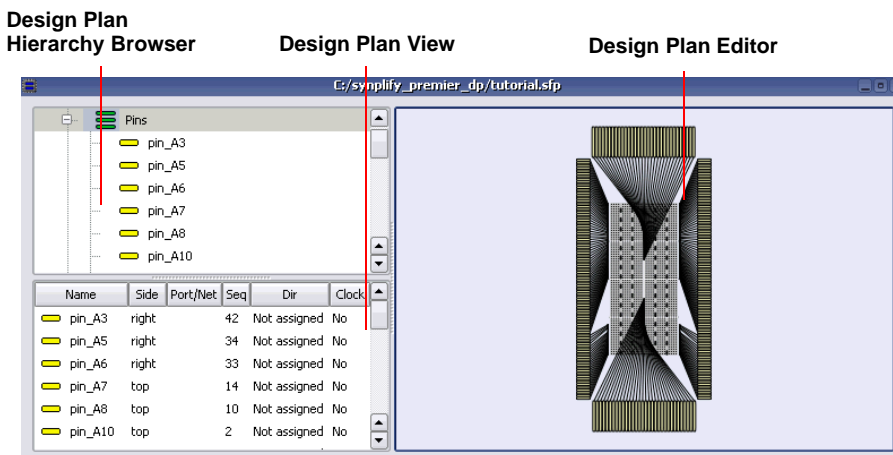
- Adjust the view by moving the slider to either a smaller or larger view of the pins.
- Click OK to save your new pin view setting or Cancel to restore your original pin view setting.



3. To display the device I/O pin names, in different views, see the table below:

| To ...                                                  | Do this...                                                                                                                                                                  |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| List the pin names in the Design Plan Hierarchy Browser | Select the expand icon next to Pins.                                                                                                                                        |
| List the pin names in the Design Plan view              | Click the design name in the Design Plan Hierarchy Browser. This lists the design objects in the Design Plan view.<br>Double-click the Pins folder in the Design Plan view. |
| View information about the pins in the Design Plan view | Right-click and select Show/Hide columns, then select the columns you need in the dialog box: Clock, Name, Side, Seq, Dir, or Port/Net.                                     |
| Display the pin number                                  | Place your cursor over the pin in the Design Plan Editor.                                                                                                                   |

When you select a pin in the Design Plan Hierarchy Browser, the corresponding pin location is highlighted in the other views. The following figure shows an example of I/O pins displayed in all three views of the Design Planner.



## Creating and Using a Design Plan File for Physical Synthesis

To create a design plan file, you must have the Design Planner option. Even if you have this option, you do not need to use a design plan file for graph-based synthesis. However, for older Altera and Xilinx technologies, the design plan file is required to run physical synthesis. The following procedure shows you how to generate a design plan file

1. Use the Design Plan editor to interactively assign RTL modules, paths or components to regions on the device.

For information about working with regions and assignment of logic, see [Assigning Pins and Clocks, on page 495](#) and [Working with Regions, on page 505](#).

For additional, technology-specific information on assigning logic to regions, see [Working with Altera Regions, on page 519](#), [Working with Xilinx Regions, on page 523](#), and [Assigning Objects to Xilinx Regions, on page 527](#).

When you have finished assigning the logic, the tool generates an .sfp physical constraint file.

2. When you create an RTL region, select Block Region Tool and then configure the region. See [Creating Regions, on page 505](#) for details.
3. Use the design plan file for physical synthesis.
  - Add the file to the project.
  - Go to Implementation Options ->Design Planning and enable the file.
  - Run physical synthesis.

The physical synthesis tool uses the placement information in the design plan file as physical constraints for synthesis.

# Assigning Pins and Clocks

This section discusses the following general guidelines for displaying and assigning pin assignments for design planning:

- [Assigning Pins Interactively](#), on page 495
- [Importing Pin Assignments from Pin Assignment Files](#), on page 498
- [Assigning Clock Pins](#), on page 498
- [Modifying Pin Assignments](#), on page 500
- [Using Temporary Pin Assignments](#), on page 501
- [Viewing Assigned Pins in Different Views](#), on page 502
- [Viewing Pin Assignment Information](#), on page 503

## Assigning Pins Interactively

In addition to the methods described in [Importing Pin Assignments from Pin Assignment Files, on page 498](#) for importing pin assignments, you can manually assign pins using the methods described here. You can either assign the pins in the SCOPE window, or use Design Planner to assign pins.

1. To assign pins directly in the SCOPE window, do the following:
  - Open the SCOPE Attributes tab.
  - Select a port. Assign it to a pin location using a pin location constraint appropriate to your technology or the `syn_loc` constraint.
  - Alternatively, manually add constraints to the `.sdc` file.

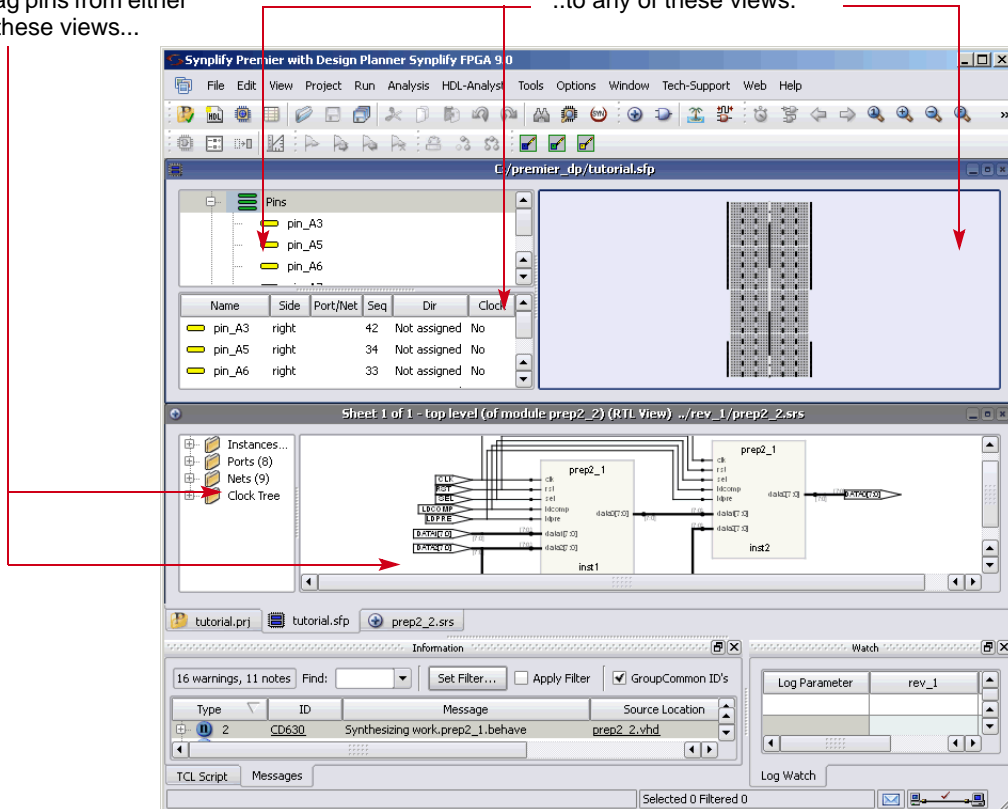
The pin assignments are stored as constraints in the `.sdc` file. For Xilinx designs, if you have the pin locations defined as UCF constraints, you can automatically translate them into SDC constraints. See [Importing Pin Assignments from Pin Assignment Files, on page 498](#) for details.

2. To assign a pin in Design Planner, do the following:
  - Open the Design Plan window and make sure you can see the pins clearly. See [Viewing Pin Assignment Information, on page 503](#) for information on displaying the pins.

- Select a pin in the Design Plan RTL view or the Hierarchy Browser for that RTL view.
- Drag the pin to the location you want in the Design Plan. You can drag it to the appropriate pin in the graphic Design Plan editor view, or to the appropriate pin name in the Design Plan Hierarchy Browser or Design Plan view.

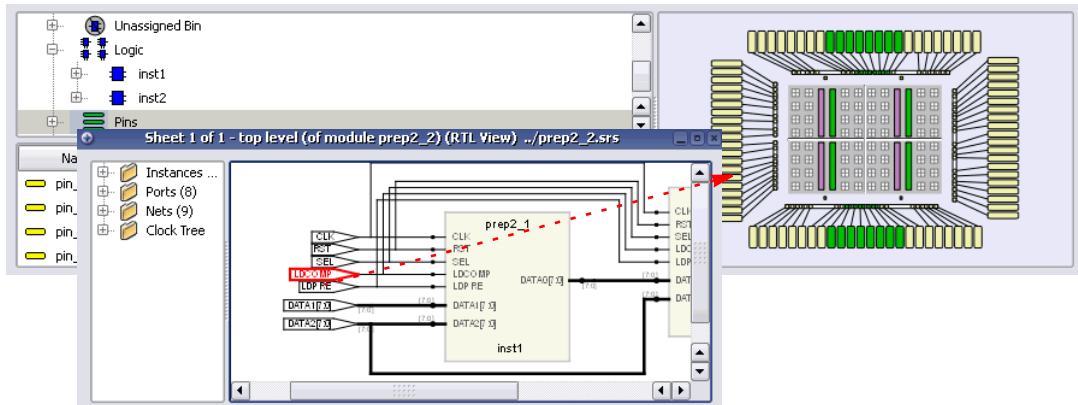
Drag pins from either of these views...

...to any of these views.



The design plan views reflect the new status of the pin. For details, see [Viewing Assigned Pins in Different Views, on page 502](#).

This example shows a pin assignment:

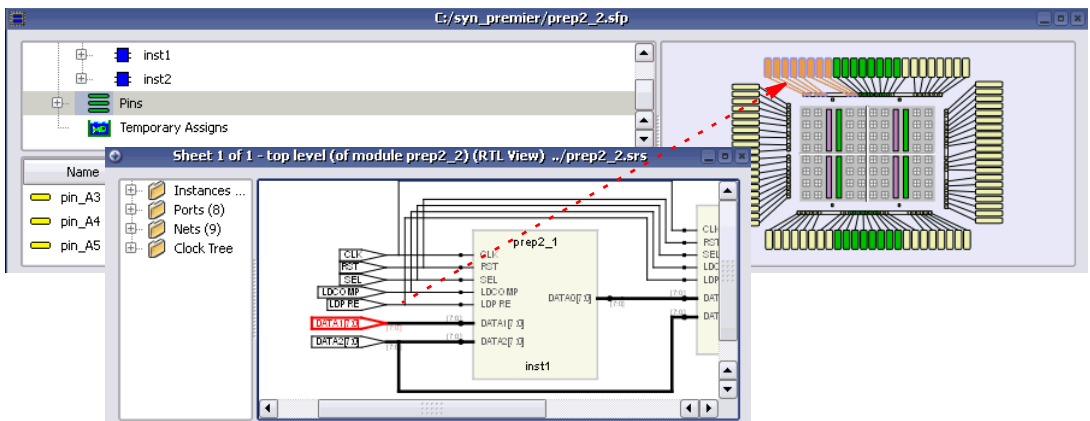


3. To assign a bus port (group of signals), drag a bus port from the RTL view and drop it to one device pin in the Design Plan Editor view.

The software allocates the remaining pin(s) depending upon its location on the device. Pins located on the left and right sides of the device are allocated from bottom to top. Pins located on the top and bottom of the device are allocated from left to right. Pins that are occupied are skipped. All devices allocate pins using this convention.

For information about viewing pin assignments and related information, see [Viewing Assigned Pins in Different Views, on page 502](#).

The following figure shows bus port assignment:



## Importing Pin Assignments from Pin Assignment Files

The following procedure describes methods for reading pin assignments from files into the design plan. To assign pins manually, see [Assigning Pins Interactively, on page 495](#).

The methods described here are used at different phases of the design process. When you import pin assignments from a file, you are usually back annotating the design with pin assignments generated after placement and routing (Altera .pin or Xilinx .pad files). For Xilinx designs, you can also import user-defined constraints in the UCF format at the physical synthesis stage.

1. For Xilinx designs, convert user-defined pin locations to SDC constraints by doing the following:
  - Define the pin locations in the UCF syntax, and save the file.
  - Translate the location constraints to SDC, using the procedure described in [Converting and Using Xilinx UCF Constraints, on page 255](#).
  - Add the constraint file to the project.

Use the UCF file to add pin location constraints to the design before you run initial placement.

2. To convert pin location information from Altera .pin or Xilinx .pad files to SCOPE SDC constraints, see [Translating Pin Location Files, on page 348](#).

Be careful if you use both SDC and .sfp file constraints, because you can create potential mismatches between the two files. The SCOPE constraint file (.sdc) typically takes precedence over the Design Plan file (.sfp) for pin assignment conflicts. For descriptions of possible conflicts, see [Assigning Pins Interactively, on page 495](#).

For information about viewing pin assignments and related information, see [Viewing Assigned Pins in Different Views, on page 502](#).

## Assigning Clock Pins

Clock pins are available for Altera and Xilinx devices, and are displayed in green in the Design Plan Editor to distinguish them from the signal I/O pins.

There are several methods of assigning I/O pins, so you might encounter pin assignment conflicts like the following:

- Imported information from a `.pin` or `.pad` file with different device packages and parts.
- The `.sdc` file might contain I/O pin locations that conflict with the pin locations specified in the `.sfp` file. The SCOPE constraint file (`.sdc`) typically takes precedence over the Design Plan file (`.sfp`) when conflicts exist after pin assignments.
- If pin assignments from the back end place-and-route tool are added into the `.sfp` file, potential pin lock conflicts may occur. In case of a conflict, the tool generates an appropriate warning message.
- Design rule checks are implemented if there are multiple assignments to the same I/O pins or ports.

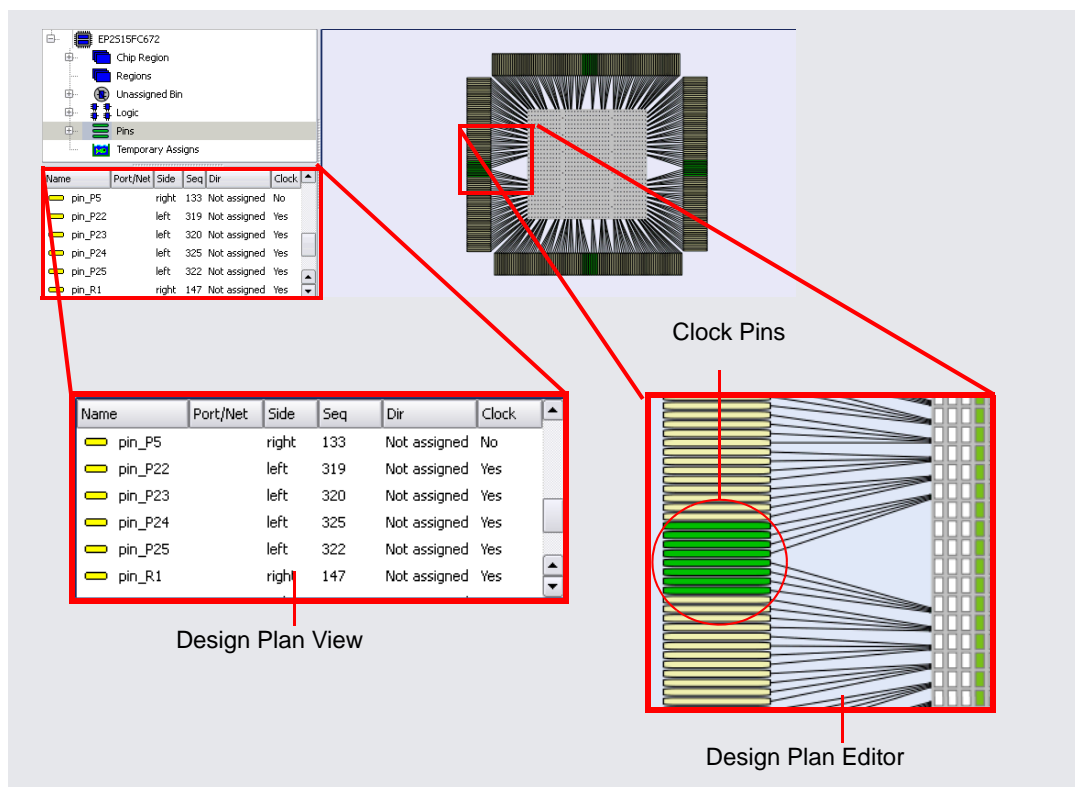
The following procedure shows you how to assign clock pins.

1. To assign a clock pin, drag and drop a signal to a clock pin, as described in [Assigning Pins Interactively, on page 495](#).

A message asks you to confirm the assignment to ensure that the correct signal gets assigned to the clock pin. After assignment, the pin changes to pink.

You cannot drag and drop a bus (group of signals) to a clock pin. If you drag and drop a bus to an I/O pin near a clock pin, the tool skips the clock pin when it assigns the bus to the I/O pins.

2. To view the information in the Design Plan view, enable the Clock column on the Select Columns dialog box. This displays whether or not a pin is a clock (Yes or No).



## Modifying Pin Assignments

The following shows you how to change pin assignments once they have been made.


1. To undo an assignment, click on the pin in any of the three views, right-click, and select Delete Pin Assignment.
2. To reassign a port, drag and drop it at a new location in the Design Plan editor.
3. To change the current order of pin assignments from clockwise to counter-clockwise or vice versa, do the following:
  - Select a set of pins in any view of the Synplify Premier Design Planner.



- Right-click and select Reverse Pin Assignments from the pop-up menu. The reversed pin assignments are displayed in the Design Planner views.
4. To rearrange or reorder pin assignments for nets or ports, do the following:
    - Move the pins to temporary assignments. See [Using Temporary Pin Assignments, on page 501](#) for details.
    - Assign them to the desired locations.

## Using Temporary Pin Assignments

Use temporary assignments to rearrange or reorder pin assignments for nets or ports

1. To create a temporary assignment, drag and drop an assigned pin from the Design Plan editor to the Temporary Assigns icon (  ) in the Hierarchy Browser.

The Temporary Assigns container lists the pins with temporary assignments. Note that you *cannot* drag and drop assignments from the HDL Analyst RTL view to Temporary Assigns.

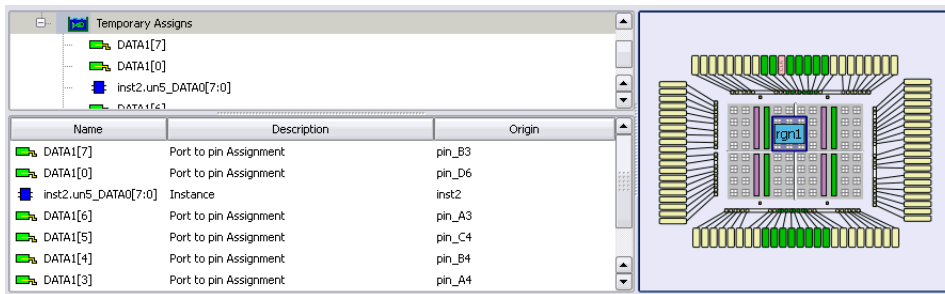
2. To re-assign a pin with a temporary assignment, do either of the following:
  - For assignment to a new location, drag and drop the pin from the Temporary Assigns container to the new pin or region location in the Design Plan Editor. You can also reassign the pin using the methods described in [Assigning Pins Interactively, on page 495](#).
  - To return the pin to its original placement location, select the assignment in the Temporary Assigns. Then, right-click and select Reassign from the pop-up menu.

As pins in the Temporary Assigns container are reassigned, they are automatically removed from Temporary Assigns.

3. To remove assignments from all the pins, select the Temporary Assigns icon, right-click, and select Empty.
4. To sort pin assignments by description, name, or origin in the Design Plan View, do the following:

- Display the appropriate column by right-clicking and selecting Show Columns->Description/Origin from the popup menu.
  - Click on the column heading in the Design Plan View to sort.
5. To undo or redo operations in the Temporary Assigns container, use the Edit->Undo or Edit->Redo commands.

The following figure shows a temporary assignment:



## Viewing Assigned Pins in Different Views

The following table summarizes how pin assignments are displayed in the Design Plan views:

Design Plan  
Hierarchy Browser

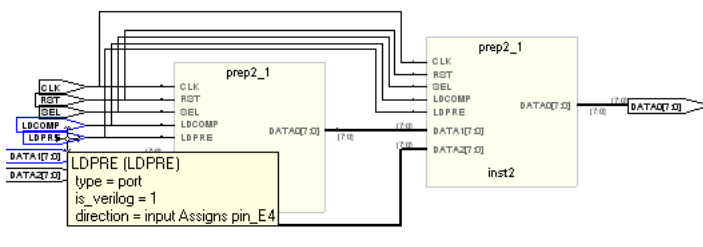
Assigned pins include assignment information. Selected assigned pins are red.

Design Plan view

To view information for the pins, select Show/Hide columns from the popup menu and choose the kinds of information you want to display for the pins, like pin direction and port or net information.

| Name    | Port/Net | Side  | Seq | Dir          | Clock |
|---------|----------|-------|-----|--------------|-------|
| pin_A3  |          | right | 42  | Not assigned | No    |
| pin_A5  |          | right | 34  | Not assigned | No    |
| pin_A6  |          | right | 33  | Not assigned | No    |
| pin_A7  |          | top   | 14  | Not assigned | No    |
| pin_A8  |          | top   | 10  | Not assigned | No    |
| pin_A10 |          | top   | 2   | Not assigned | No    |
| pin_A11 |          | top   | 1   | Not assigned | No    |
| pin_A13 |          | top   | 9   | Not assigned | No    |

|                    |                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Design Plan editor | <p>Orange: Selected assigned pins</p> <p>Red: Unselected assigned pins</p> <p>Blue: Selected unassigned pins</p> <p>Green: Unassigned clock pins</p> <p>Pink: Unselected assigned clock pins</p> <p>To view the pin number and assignment for a pin, place the cursor over the pin.</p> |
| RTL view           | <p>Assigned ports are displayed in blue. Place your cursor over a pin to display information about it.</p>                                                                                                                                                                              |



## Viewing Pin Assignment Information

This section lists different methods you can use to obtain information about your pin assignments. These methods are in addition to the display information described in [Viewing Pin Assignment Information, on page 503](#).

- To view information for a particular pin, use the following:
  - Tooltips  
To display a tooltip, move your cursor over a pin or port in the Design Plan editor or in the RTL view.
  - The visual clues described in [Viewing Pin Assignment Information, on page 503](#).
- To display connectivity between the I/O pads and the assigned logic for regions on the device, do the following:
  - To view connectivity for all regions, right-click in the Design Plan Editor and select Rats Nest->Show from the pop-up menu.
  - To view connectivity for one region, select it and right-click in the Design Plan Editor. Select Rats Nest->Show Selected from the menu.

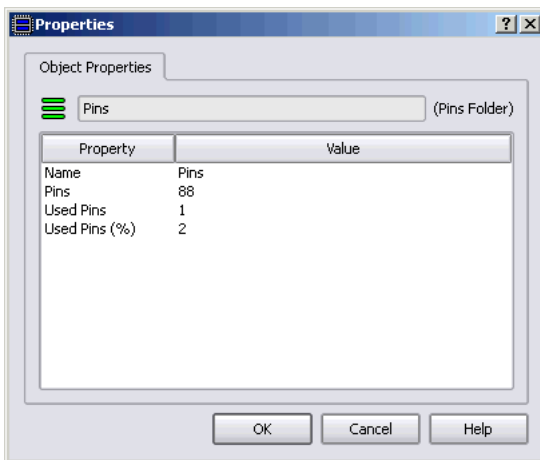
- To disable the connectivity display, right-click in the Design Plan Editor and select Rats Nest->Hide.

Alternatively, you can also select View->Rats Nest from the Project menu, then choose the Show, Hide, or Show Selected command.

The display shows lines (rat's nesting) to indicate the connectivity.

3. To view pin assignment statistics for the design, right-click on the Pins folder in the Design Plan Hierarchy Browser, and select Properties from the pop-up menu.

The Properties dialog box shows the total number of pins, the number of assigned pins, and the percentage of pins assigned.



4. Use crossprobing.

When you select assigned ports in any of the Design Planner views or the HDL Analyst RTL view, the corresponding pins are highlighted in the other views view. Similarly, if you select a net that has an assigned pin in the RTL view, the corresponding pin is highlighted in the Design Planner views. If you select the assigned pin in a Design Planner view, the corresponding internal net is highlighted in the RTL view.

# Working with Regions

This section discusses the following general guidelines for placing and editing regions in the Design Planner before running physical synthesis:

- [Creating Regions](#), on page 505
- [Using Region Tunneling](#), on page 507
- [Viewing Intellectual Property \(IP\) Core Areas](#), on page 510
- [Assigning Logic to Top-level Chip Regions](#), on page 510
- [Assigning Logic to Regions](#), on page 514
- [Replicating Logic Manually](#), on page 515
- [Assigning Critical Paths from Island Timing to a Region](#), on page 516
- [Checking Utilization](#), on page 517

## Creating Regions

Region placement depends on the data flow and pin locations in your design. The following procedure shows you how to create a region.

1. If needed, select View->Expanded Pin View and adjust the view to display the device with or without I/O pins.
2. To create a region, right-click in the Design Plan Editor and select Block Region Tool to begin the region drawing process.

For more information about creating technology-specific regions, refer to the following table depending on the technology you have selected.

| <b>For...</b>  | <b>See</b>                                                                      |
|----------------|---------------------------------------------------------------------------------|
| Altera designs | <a href="#">Creating Design Planner Regions for Altera Designs, on page 520</a> |
| Xilinx designs | <a href="#">Creating Regions for Xilinx Designs, on page 525</a>                |

3. Position the cursor where you want to create the region and then drag the cursor diagonally to create a rectangular area for the region.

- Do not create regions that overlap or are contained within an area reserved for IP (see [Viewing Intellectual Property \(IP\) Core Areas, on page 510](#)).

See the following vendor-specific guidelines to determine where to place the regions.

---

### Altera

|                         |                                                                                                                                                                                                                                                                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Critical path placement | Run the target place-and-route tool with no constraints to obtain the placement of the critical path. Use the Design Plan Editor to create a region in this area. This is a good starting point to determine what row to begin with when placing the critical path on the logic device using the Synplify Premier Design Planner tool. |
| Overlapping regions     | You can overlap regions to optimize placement. However, be aware that that the Synplify Premier Design Planner software treats overlapping regions no differently than regions that do not overlap.                                                                                                                                    |

---

### Xilinx

|                         |                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Region size             | The size and location of Xilinx regions can be easily modified, so a rough estimate is usually sufficient.                                                                                                                                                                                                                              |
| Critical path placement | You can get a good starting point for region placement from the Xilinx floorplanner. Run placement and routing without constraints, then use the floorplanner to determine where the critical path logic is placed. Use this information to create a region in the same general area on the logic device using the Design Planner tool. |
| Overlapping regions     | The Synplify Premier Design Planner software supports overlapping regions, but the Xilinx place-and-route tool cannot always place these designs. Overlapped regions can potentially create an error.                                                                                                                                   |

4. You can configure the region to apply selected tunneling modes. See [Using Region Tunneling, on page 507](#) for ways to configure regions.
5. Then, assign logic to the region. For more information, see [Assigning Logic to Regions, on page 514](#).

## Using Region Tunneling

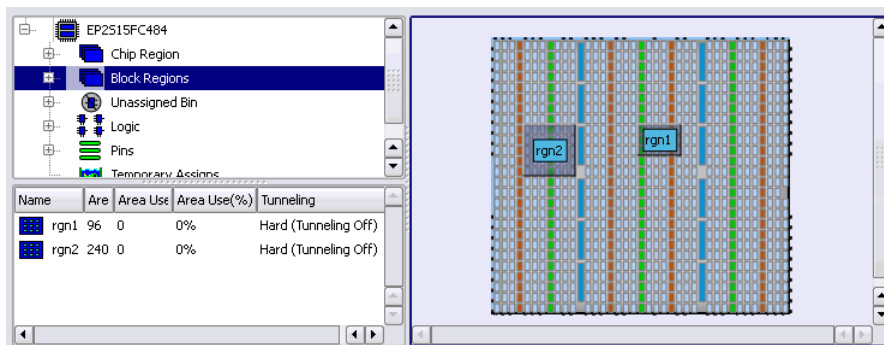
The Synplify Premier software can apply tunneling optimizations to region assignments. To do this:

1. Highlight a region in the Design Planner, then right-click and select Region Type.
2. You can configure the region by selecting one of the following modes shown in the table below. Some options are vendor-specific.

| Set the option to... | To...                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Soft (Tunneling On)  | Allows components to be moved across the region boundaries in both directions. This is the default.                                                     |
| Hard (Tunneling Off) | Ensures that components are not moved out of the region, but allow other objects to be moved into the region.                                           |
| Keep-out (Xilinx)    | Ensures that no placement occurs in the region. Use this option to create decongestion areas for optimizing your design.                                |
| IP Block (Altera)    | Ensures that the region only contains the IP block. Use this for encrypted IP, to ensure that nothing except for the IP logic is placed in this region. |

Note, it is possible to highlight multiple regions and then select a tunneling option for those regions simultaneously.

3. To view tunneling status for the region, do the following:
  - To view tunneling status for a region, highlight the region, then right-click and select the Properties option. From this dialog box, the tunneling status for the region is displayed.
  - To display the tunneling status for all the regions, go to the Design Plan view, right-click and select Show/Hide columns. From the Select Columns dialog box, check the box next to Tunneling. The view displays a column with the tunneling status for all the regions.



The Design Planner can display how tunneling is implemented. Also, the status of the region is saved and written out to the Synplify Premier physical constraint file (.sfp).



## Moving and Sizing Regions

You can move and resize regions using the cursor arrow keys or the mouse button. The following procedure provides details.

1. To move a regions use the arrow keys or the mouse button.
  - Select the region.
  - To use the arrow keys:, use the left, right, up, or down keys to reposition the region.
  - To use the mouse button, press the left mouse button while dragging the region to the desired position on the device.

The tool displays WYSIWYG region boundaries that show you exactly what you are doing when you move or resize the boundaries.

- To preserve the logic and memory resources of a region when it is moved, hold down the Shift key when you move it.

The tool preserves the logic and memory resources when you move a region. For example, Xilinx devices can preserve the number of CLBs and BRAMs and Altera devices can preserve the number of LABs and ESBs in a region.

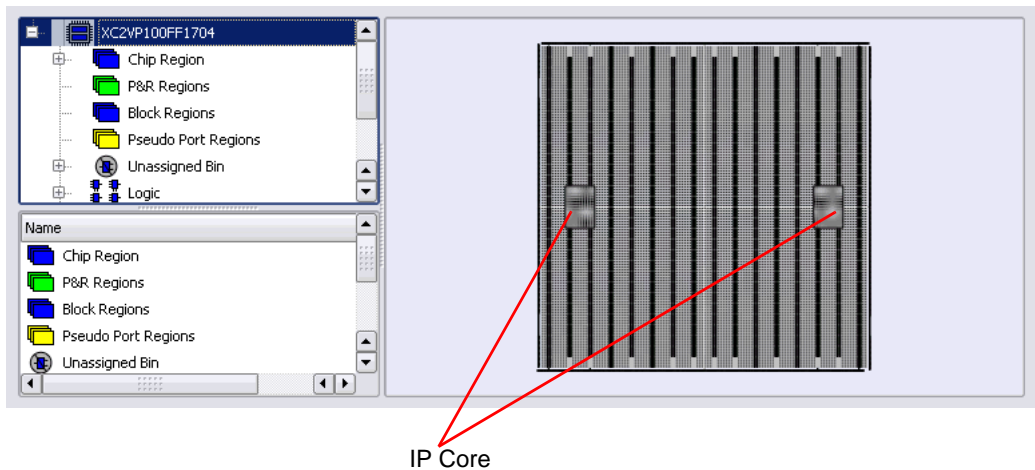
2. To resize a region, use the arrow keys or the mouse button.
  - Select the region.
  - To use the arrow keys:, press and hold the Ctrl and Shift keys simultaneously. An initial resizing arrow appears along the edge of the region. Continue to hold down Ctrl and Shift while pressing the appropriate arrow keys (left, right, up, or down) to resize the region in the direction you want. Release the Shift key. You can no longer resize the region.
  - To resize a region with the mouse button, press the left mouse button on any of the handles of the rectangle while dragging the region in the direction you want the region resized.

The tool does not preserve logic and memory resources when you resize a region.

## Viewing Intellectual Property (IP) Core Areas

Dedicated areas on the device reserved for IP cores appear as gray boxes in the Design Plan Editor. A device can contain up to four IP core areas depending on the part specified for the device.

- To view information about the IP core, move your cursor over the gray box to display a tooltip with information. Do *not* create regions that overlap or are contained within an IP core area.



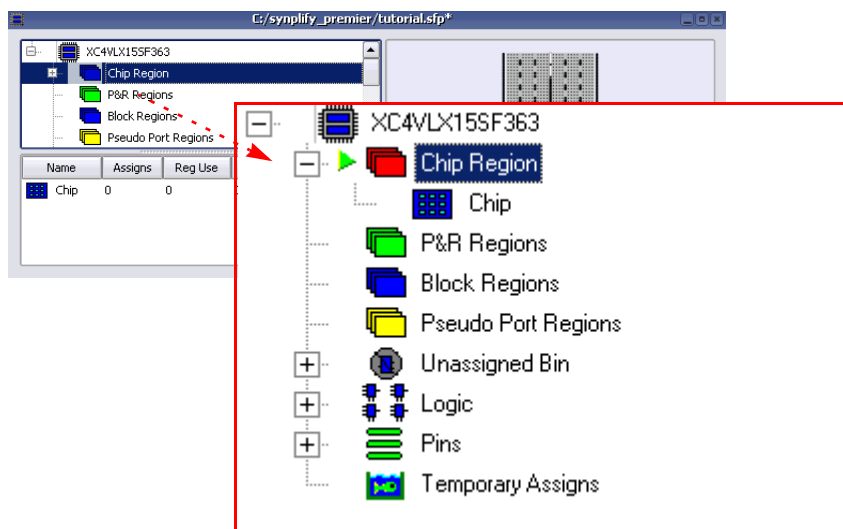
## Assigning Logic to Top-level Chip Regions

You can specify the top-level device as a region and then assign logic to this chip region. To do this:

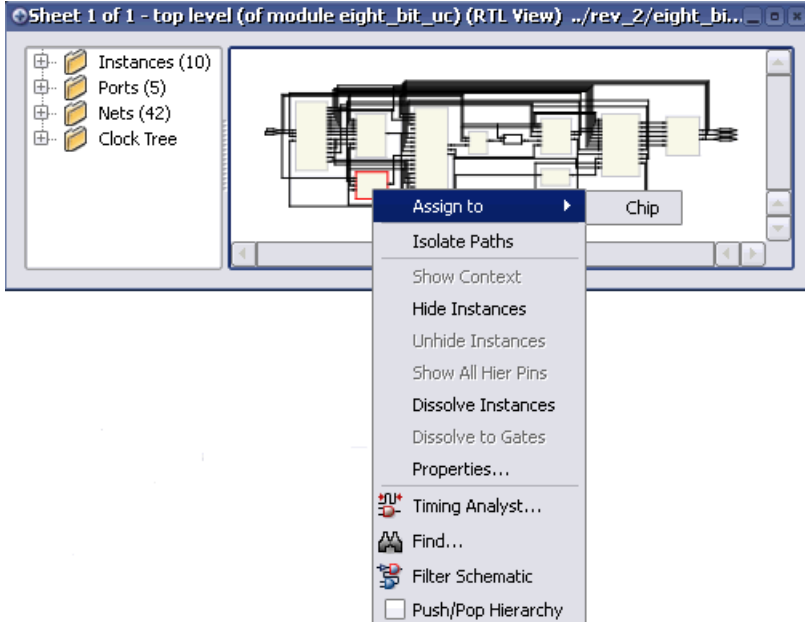
1. Open the Design Plan view.

Notice the Chip Region hierarchy under the device part and package designation in the Design Plan Hierarchy view.

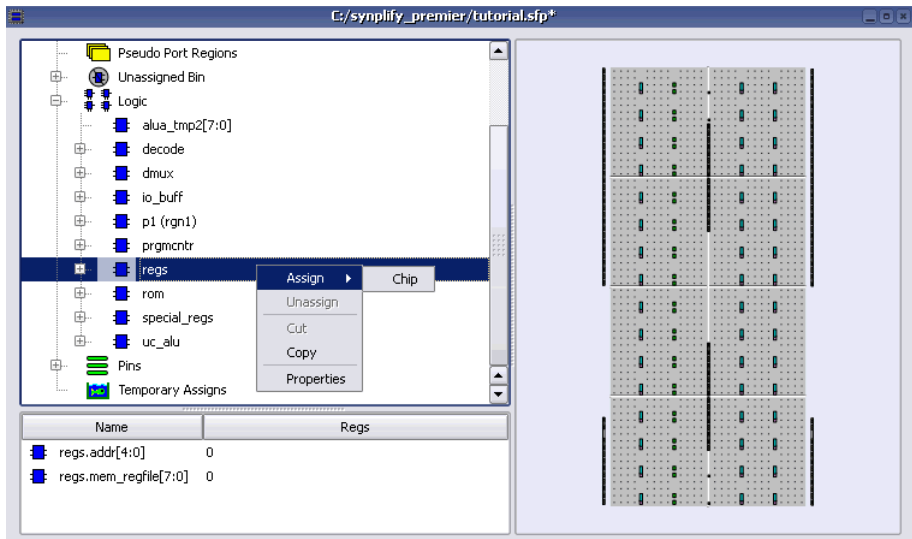
## Design Plan Hierarchy View



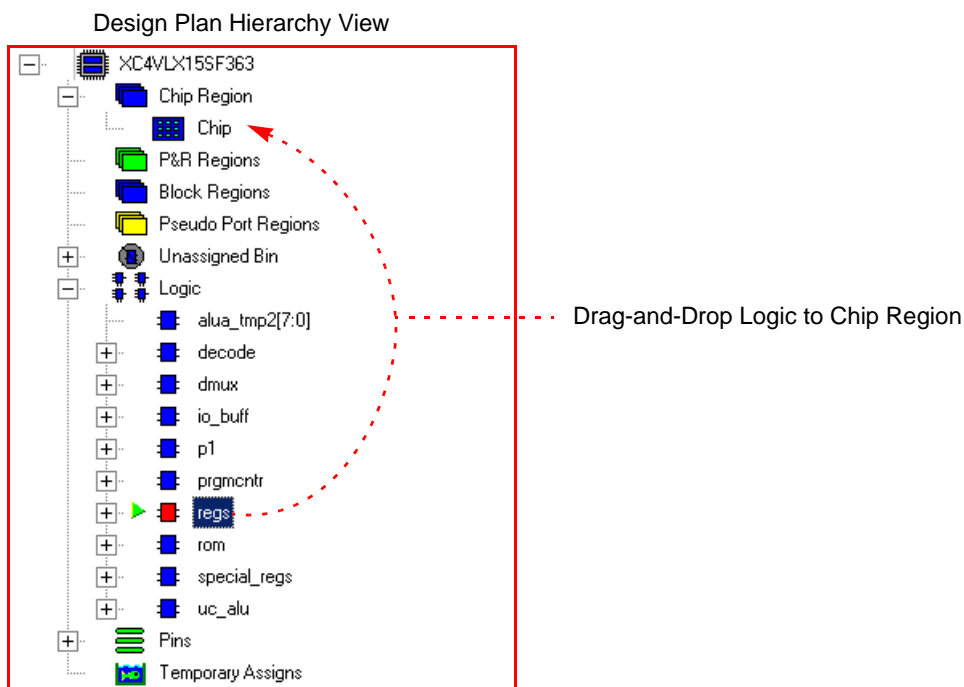
2. Assign logic to the chip region. To do this, you can:
  - Highlight logic in the RTL view, then right-click and select Assign to->Chip from the popup menu.



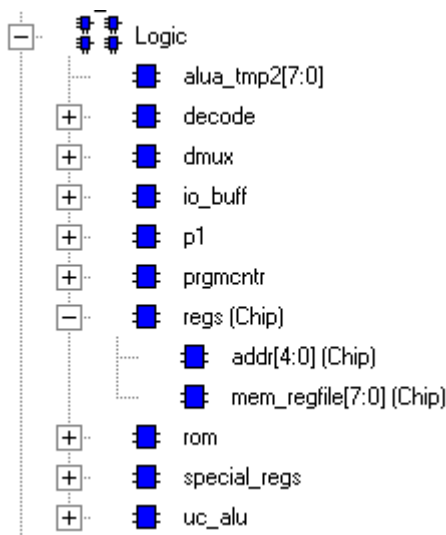
- Highlight logic to assign from Logic in the Design Plan Hierarchy view, then right-click and select Assign to->Chip from the popup menu.



- Otherwise, simply drag-and-drop highlighted logic from the Logic to Chip hierarchy tree in the Design Plan Hierarchy view.



The chip assignments are reflected in the Design Plan Hierarchy view as shown in the following figure.



## Assigning Logic to Regions

1. To assign individual instances to regions, make sure that Edit Regions is enabled and do either of the following:
  - Drag and drop the logic into the region.
  - Select the instance in an HDL Analyst, Design Plan Hierarchy Browser, or Design Plan view. Right-click and select Assign to-> <region\_name>. The regions are listed in order of recent use.

For technology-specific tips about assigning logic to regions, see the following:

- [Assigning Logic to Altera Design Planner Regions, on page 521](#)

2. To assign a critical path to a region, do the following:
  - Select the critical path, filtering it if necessary. You can do this from the log file.
  - Drag the selected critical path from the RTL view or the Island Timing view into a region in Design Planner.

By assigning the critical path instances to the same region, you can optimize the timing.

3. For critical paths from pin-locked I/Os, assign the critical path to a region that is close to the pin positions.

Physically constraining logic close to locked pins minimizes routing delays.

For information about device utilization, see [Checking Utilization, on page 517](#).

## Replicating Logic Manually

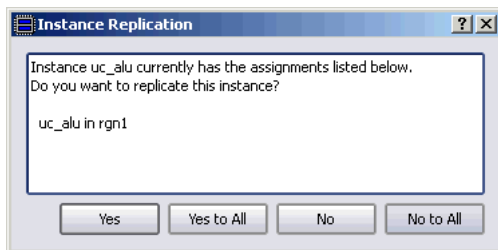
When the fanout from an instance fans goes to instances in several other instances, you might want to replicate the instance to avoid the routing delay between the regions. You can use the methods described here.

1. Replicate logic by copying and pasting.
  - Copy (Ctrl-c) the logic to be replicated from the original region.
  - Paste (Ctrl-v) the replicated logic in the region where it is required.

Each region now contains a local copy of the instance. If you replicate an instance in a region where the instance does not drive any logic, the tool does not create a copy of the instance in that region. Therefore, when you look at the RTL netlist of the region, the replica of the instance does not appear.


2. Assign the same instance logic from the HDL Analyst RTL view to different regions.

The Instance Replication dialog box opens. Confirm whether or not you want to replicate the selected logic instance in the specified region.



## Assigning Critical Paths from Island Timing to a Region

Critical paths in the Synplify Premier island timing report are determined based on a pre-defined range from the worst case slack of the island. The following procedure explains how to assign island critical paths to a region:

1. Synthesize (compile and map) the design.
2. Set up the views.
  - Create a new implementation for the project, which includes a design plan.
  - Click on the New Design Plan icon button () to open the Design Planner view.
  - Open the flattened RTL view.
3. Select the paths.
  - Open the hierarchical-based island timing report file (.tah) or use the Island Timing Analyst. Make sure that the start and end points in this report match start and end points in the place-and-route timing report.
  - Press the Alt key and select the RTL start and end points from the island timing report file (.tah) or use the Island Timing Analyst. Then, do either of the following:

When all the start and end points are selected, right-click and press Filter Analyst from the popup menu in the .tah file.

Click on the Cross Probe button in the Island Timing Analyst and filter the selected gates in the flattened RTL view. Currently, for crossprobing to work properly in the Island Timing Analyst, open the flattened Technology view also.
4. Assign the paths to a region.
  - Right-click and select Expand Paths from the popup menu in the flattened RTL view.
  - Either right-click and select Assign to->region\_name or drag-and-drop the selected expanded paths to the region in the Design Plan Editor of the Design Planner view.
5. Run estimation for any design plans created.



6. Save these assignments to the Synplify Premier design plan file (.sfp).

Run synthesis for this implementation with a design plan.

## Checking Utilization

Use the following tips and guidelines for device and region utilization when assigning logic to regions.

1. To view device utilization, select Run->Estimate Area.

Utilization is reported in the log file.

2. To estimate region utilization, do the following:

- To estimate utilization for all the regions, right-click in the Design Plan Editor view, right-click and select Estimate All Regions.
- To estimate region utilization for an individual region, right-click with a selected region and select Estimate Regions.

Estimates are in terms of instances. As the job runs, the region is greyed-out and a label in the upper-left corner of the region displays the elapsed time of the estimation job. The label remains until estimation is complete. Est Pending appears in the upper-left corner of all regions waiting for region estimation.

Regions displayed in red have a utilization higher than 80%. See step 4 for more information about utilization.

3. To view region utilization information, use one of these methods:

- For utilization information for the whole design, view the log file.
- For utilization information about the current estimation run, view the status in the Tcl Script window. or select Run->Job Status immediately after an estimation run.
- To view utilization information for a selected region, right-click and select Properties. You can also view the tooltip information.
- To display utilization information for regions in the Design Plan view, click Regions in the Design Plan Hierarchy Browser. This updates the Design Plan view with statistics for the regions.
- To determine which statistics to display in the Design Plan view, right-click in this view and select Show/Hide Columns. Select the options

for utilization you want to display. The available options can vary with the technology.

4. Follow these guidelines for device and region utilization.
  - Keep device utilization below 90%. Higher utilization rates can lead to problems with timing closure.
  - If device utilization is over 90%, and if the design contains several finite state machines, try using the sequential encoding style, (instead of one-hot) to free up more space on the device.
  - Keep region utilization below 80%. This allows for Synplify Premier Design Planner area estimations and for additional area required for routing and replicating. The place-and-route tools consider the design plan to be a hard constraint, so if there is not enough area in the region for routing, the place-and-route tool will error out.
5. If utilization exceeds the guidelines, resize the region to ensure that it is not over utilized.

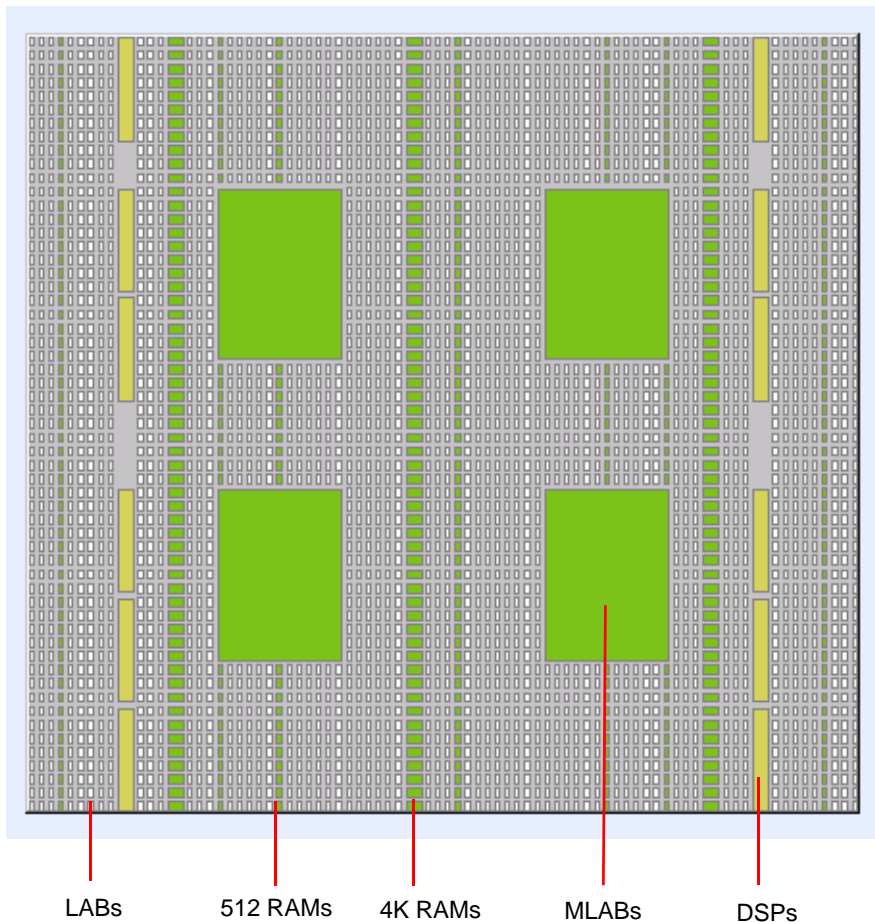
## Working with Altera Regions

The guidelines in this section provide tips and strategies for using the Synplify Premier Design Planner for design planning with Altera Stratix and Cyclone devices. It supports the following technologies:

- Stratix families  
(Stratix IV, Stratix III, Stratix II GX, Stratix II, Stratix GX, and Stratix)
- Cyclone families  
(Cyclone II and Cyclone)

You can use the Design Planner to view the Altera devices, then create regions and assign critical path logic to them. The Stratix and Cyclone family of devices use a row and column coordinate system, with the origin (1,1) located at the lower-left corner of the device. All components align with row and column boundaries. The device features can include LABs (logic blocks), 512 RAMs, 4K RAMs, M9K RAMs, M144K RAMs, and MRAMs (512K RAMs). Depending on the device and part and package used, the number of these blocks on the device may vary.

The following shows an Altera Stratix device in the Design Plan Editor.



This section describes the following:

- [Creating Design Planner Regions for Altera Designs](#), on page 520
- [Assigning Logic to Altera Design Planner Regions](#), on page 521

## Creating Design Planner Regions for Altera Designs

This section contains Stratix- and Cyclone-specific information about creating regions. For information about how to create a region, see [Creating Regions](#), on page 505.

- You can create a region around any number of LAB, RAM, and DSP structures on the device. You can create regions that contain only LABs, only RAMs, only DSPs, or regions that include any combination of LABs, RAMs, and DSPs, as required. However, you cannot create a region that is completely contained within the boundaries of one of these blocks.
- Regions can be moved or resized.
- When you create, move, or resize regions, they snap to the row/column grid on the device.

## Assigning Logic to Altera Design Planner Regions

This section contains tips and guidelines for mapping MACs, RAMs, and ROMs to regions.

### Mapping MACs

1. Enable the Create MAC Hierarchy optimization on the Netlist Restructure tab of the Implementation Options dialog box.

For example, this option is enabled by default for Stratix devices. When enabled, it maps MAC configurations together into one MAC block so that this block can be easily assigned to DSP regions for physical synthesis.

2. Follow these guidelines when assigning MACs:
  - Place MAC blocks in a region containing DSP resources. If you do not do this, the MAC block is mapped to logic and a warning message is generated in the log file (.srr). You can display DSP resources after you estimate utilization. See [Checking Utilization, on page 517](#) for a procedure.
  - Do not place signed and unsigned multipliers in the same DSP block.
3. If you are using the `syn_multstyle` attribute, note the following:
  - Do not set the attribute value to `logic` for a MAC. If the attribute is set to `logic`, the tool maps the MAC to logic and generates a warning message in the log file (.srr).
  - Do not place a multiplier with a `syn_multstyle=lpm_mult` attribute in a region without DSP resources. If you do, the tool maps the multiplier to the MAC block and generates a warning message in the log file.

## Mapping RAMs and ROMs

1. Place RAM/ROM logic in a region containing RAM/ROM resources.

If you do not, the tool maps the RAM/ROM to logic and generates a warning message in the log file (.srr). You can display RAM and ROM resources after you estimate utilization. See [Checking Utilization, on page 517](#) for a procedure.

2. Ensure that the register driving the address or the output register is assigned to the same region.

If not, the RAM will not be inferred.

3. If you are using the syn\_ramstyle attribute, note the following:

- Do not set syn\_multstyle=logic, and then assign the RAM/ROM logic to a region with RAM/ROM resources. The tool maps the ROM/ROM instance to logic, and generates a warning message in the log file (.srr).
- Do not assign RAM/ROM logic with an attached syn\_ramstyle=blockram attribute to a region without RAM/ROM resources. If you do so, the tool maps the RAM/ROM to altsyncram and generates a warning message in the log file.

## Working with Xilinx Regions

The guidelines in this section provide tips and strategies for using the Synplify Premier Design Planner for design planning with Xilinx devices. The Design Planner supports the following Xilinx technologies:

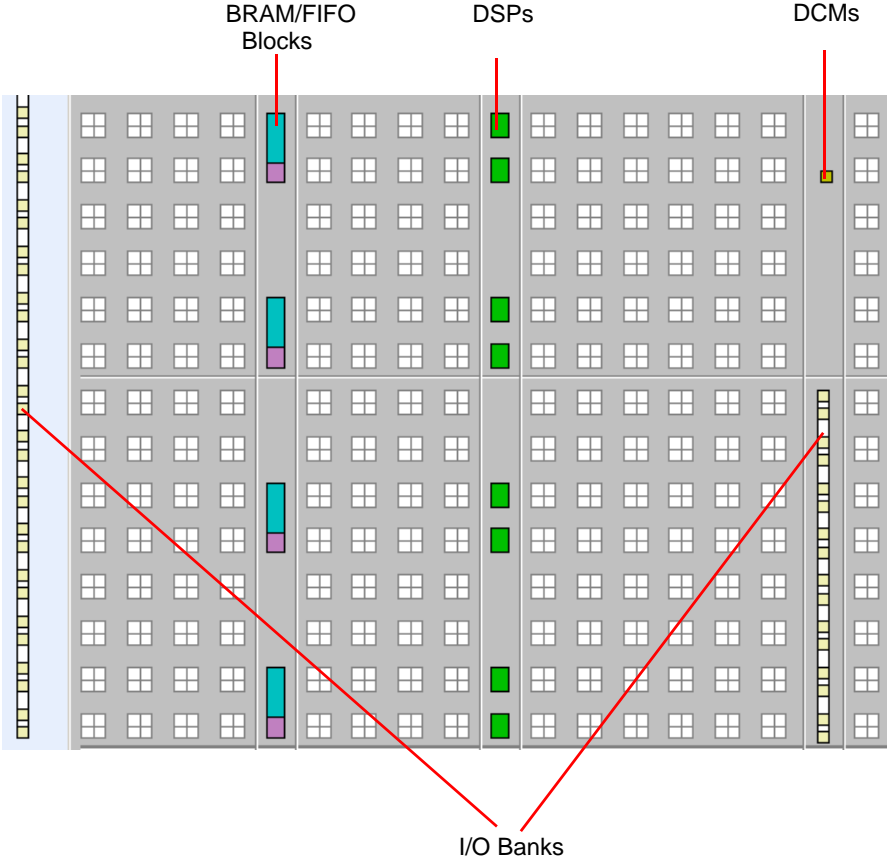
- Virtex families  
(Virtex-5, Virtex-4, Virtex-II Pro, Virtex-II, Virtex-E, and Virtex)
- Spartan-3

For more information see the following:

- [Xilinx Device Resources](#), on page 523
- [Creating Regions for Xilinx Designs](#), on page 525

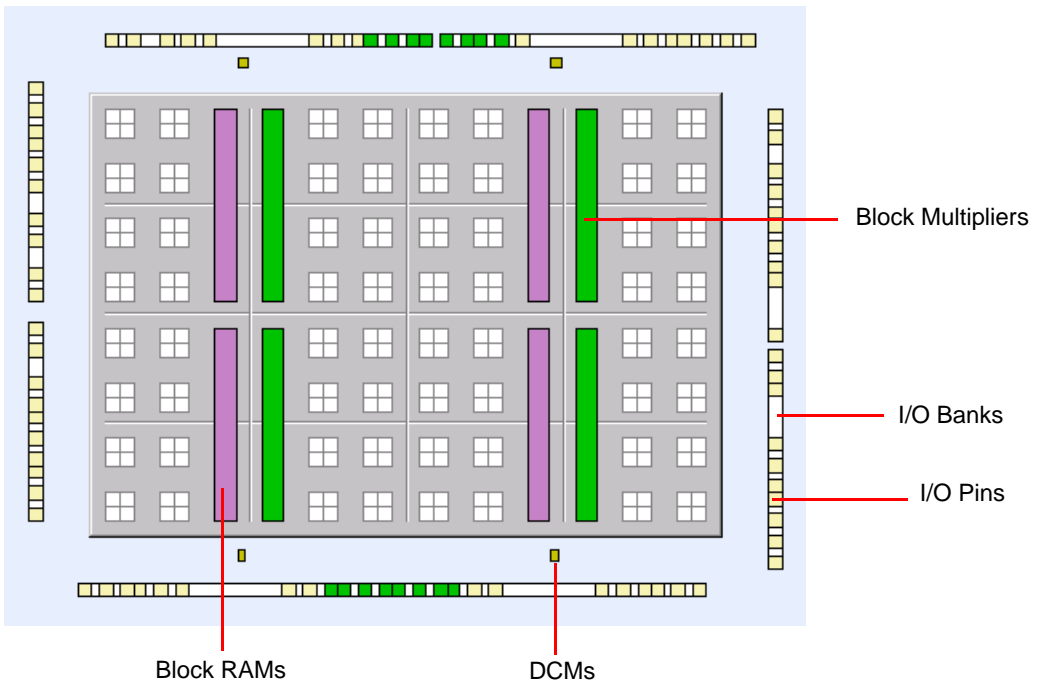
### **Xilinx Device Resources**

The Xilinx devices can include resources like Block FIFOs, DSP elements , Block RAMs, Block multiplexers, DCMs, and I/O banks . The resources are located within the device or around the perimeter, depending on the technology family you select. The number of resources vary with the technology family. The following figure shows the lower left corner of a Virtex-4 device.





The following example displays the same resources on a Virtex-II device.



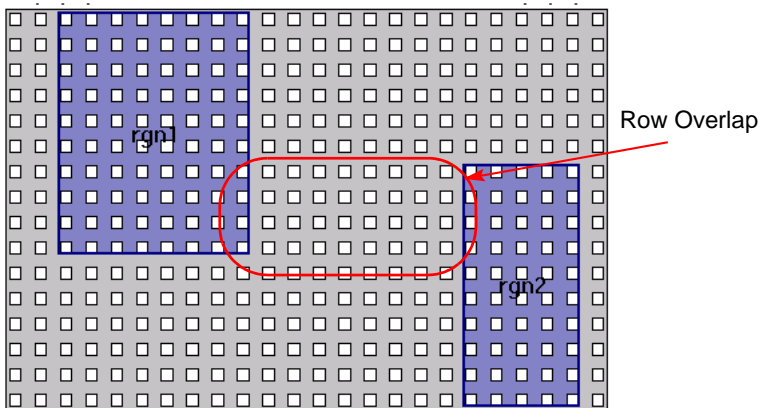
## Creating Regions for Xilinx Designs

The following vendor-specific guidelines are intended to supplement the procedure described in [Creating Regions, on page 505](#). Use the following recommendations to help you design plan regions in the Design Plan Editor for Xilinx devices:

- Base your placement on the Configurable Logic Block (CLB) coordinate system, which varies with the device:

| Family                                                  | Row1 Col1 Location |
|---------------------------------------------------------|--------------------|
| Virtex, Virtex-E                                        | Top-left corner    |
| Virtex-5, Virtex-4, Virtex-II Pro, Virtex-II, Spartan-3 | Bottom-left corner |

- The number of CLB rows in a region must be greater than the length of the cascade/carry chain logic assigned to it. Refer to [Handling Critical Paths with Cascading Cells or Carry Chain Logic](#), on page 530.
- Avoid overlapping regions on Xilinx devices. Although the Design Planner supports overlapping regions, the Xilinx place-and-route tool cannot place some designs with overlapping regions and can result in an error.
- For optimum region utilization in Spartan-3 architectures, create regions that are at least 6 x 6 CLBs.
- Regions that drive a global bus signal must overlap other regions that drive the same global bus signal either in rows or columns. For Virtex family architectures, all tristates feeding the same bus signal are required to be on the same CLB row or column (4 bus signals per row/column).



- If multiple regions drive a global bus signal make sure that there is some overlap between all the regions that drive the signal.

## Assigning Objects to Xilinx Regions

The following provide more information about assigning objects to Xilinx regions:

- [Assigning Xilinx Critical Paths to Design Planner Regions](#), on page 527
- [Assigning Xilinx Block RAMs to Regions](#), on page 534
- [Assigning Xilinx Block Multipliers to Regions](#), on page 539
- [Assigning Xilinx DSP Blocks to Regions](#), on page 540

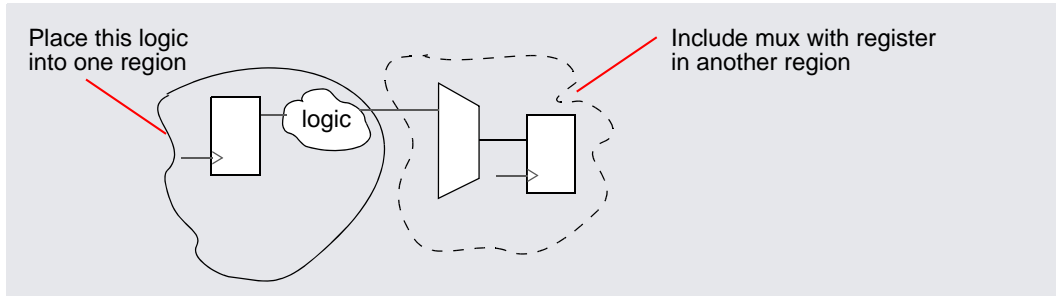
## Assigning Xilinx Critical Paths to Design Planner Regions

The following describe techniques for using Design Planner to handle Xilinx critical paths:

- [Splitting a Critical Path into Multiple Logic Regions](#), on page 527
- [Dividing Long Critical Paths into Smaller Regions](#), on page 528
- [Handling Virtex Critical Paths with High Fanout Nets](#), on page 529
- [Handling Critical Paths with Cascading Cells or Carry Chain Logic](#), on page 530
- [Handling Critical Paths with Bit Slicing](#), on page 531
- [Handling Critical Paths with Pipelining](#), on page 532
- [Handling Designs with Multiple Critical Paths](#), on page 533
- [Handling Critical Paths with Large Multiplexers](#), on page 533

### Splitting a Critical Path into Multiple Logic Regions

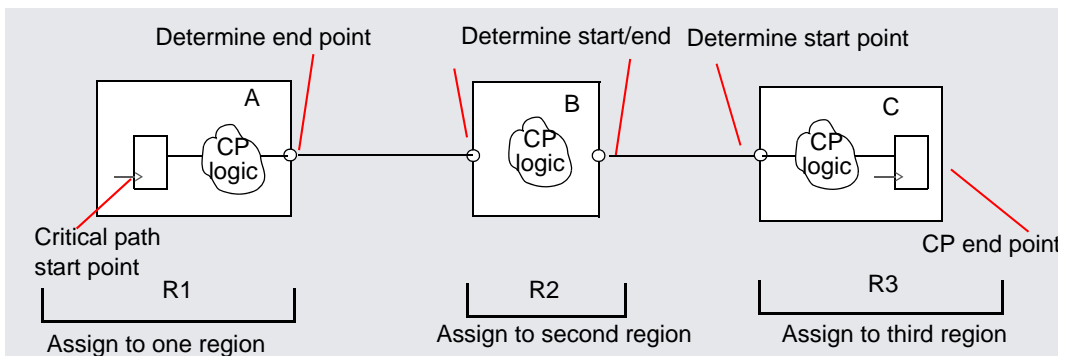
If a critical path contains logic that should be placed closely together, you can use Design Planner to split a critical path into multiple regions that contain common logic. For instance, if a critical path ends with a large multiplexer feeding a register, you might find that the large mux is decomposed and spread out throughout the region. To prevent the mux from spreading, you can split the region into two, then constrain the mux and the register to one region and the rest of the logic to the other region.



## Dividing Long Critical Paths into Smaller Regions

For the most optimal timing results, you must divide the logic of the modules containing the critical path into smaller regions. Assign logic to each of the regions so that it follows the dataflow of the design. This example uses a critical path that runs through modules A, B, and C.

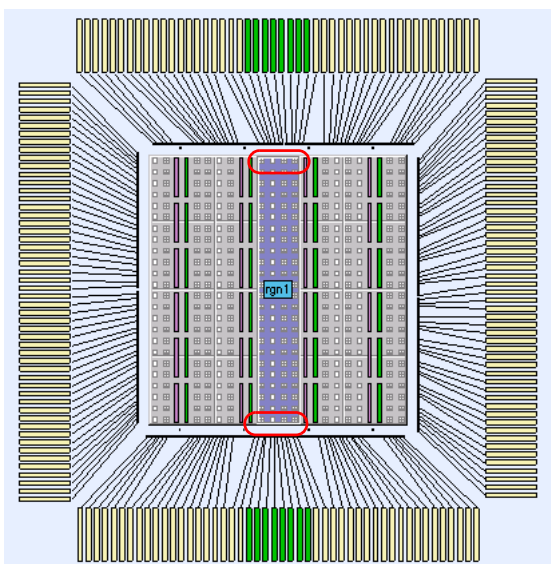
1. Assign the first section to one region.
  - Determine the start point of the critical path in module A and the end point in module A.
  - Assign this portion of the critical path to one region.
2. Assign the second section to another region.
  - Determine the start and end points of the critical path in module B.
  - Assign this portion of the critical path to a second region.
3. Repeat the previous step for module C and assign it to a third region.



## Handling Virtex Critical Paths with High Fanout Nets

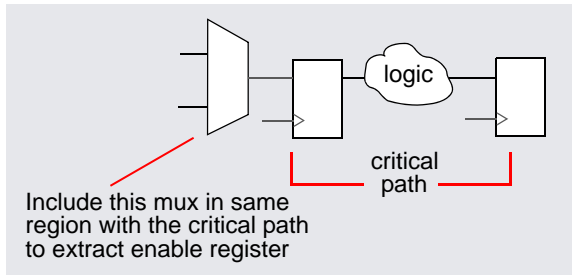
The Virtex family of devices contain secondary routing buffers on the first and last CLB rows of the device. These routing buffers can only be accessed from the first and last CLB row close to the vertical center of the device.

When a critical path contains a design with high fanout nets, it is best to place that critical path in a region that includes the first or last CLB row near the middle column of the device, so that the tool can access the routing buffers.

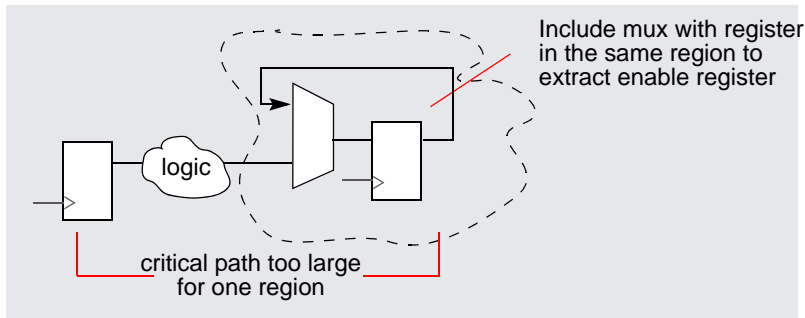


## Extracting Enable Registers

For the Design Planner software to extract an enable register, the mux and the register must be kept together. If a register of a critical path is fed by a mux that is outside the critical path, you must constrain the mux along with the register and the rest of the critical path logic into the same region.



If the critical path is divided between two or more regions, place the register and the associated mux in the same region to ensure that the enable register is extracted.



## Handling Critical Paths with Cascading Cells or Carry Chain Logic

If your critical path contains cascading cells or carry chains, use the following procedure to create regions.

1. Create a region that is large enough to support the cascade/carry chain assignment:

If a region is not large enough, the Xilinx place-and-route tool fails. The area requirements vary with the technology. For an 8-bit adder in a Virtex design you must create a region with at least 4 CLBs in the vertical direction to accommodate the carry chain. The following table shows the area requirements.

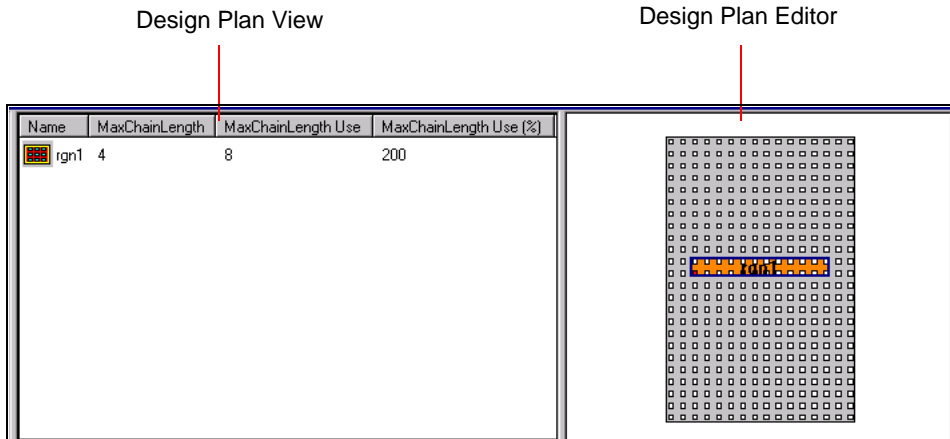
|                                                      |                  |
|------------------------------------------------------|------------------|
| Virtex-5 (repackaged device),<br>Virtex-E and Virtex | 2 bit slices/CLB |
| Virtex-4, Virtex-II Pro, Virtex-II, and Spartan-3    | 4 bit slices/CLB |

## 2. Assign the cascade/carry chain logic to the region.

The tool implements a carry chain DRC (design rule check) to ensure that the region is large enough. The Design Plan view displays information like the following:

|                        |                                                                                                                                                                                                                                                                                       |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MaxChainLength         | Length of the longest cascade/carry chain that can fit into the specified region.                                                                                                                                                                                                     |
| MaxChainLength Use     | Length of the longest cascade/carry chain assigned to the region, obtained after you estimate the region utilization.                                                                                                                                                                 |
| MaxChainLength Use (%) | Percentage of the length of the longest cascade/carry chain assigned to the region and the longest cascade/carry chain that fits into the region, after you run region estimation. This percentage can show that the cascade/carry chain exceeds its capacity to fit into the region. |

If you have a DRC violation, the region is orange. You must resize the region to avoid a place-and-route failure.

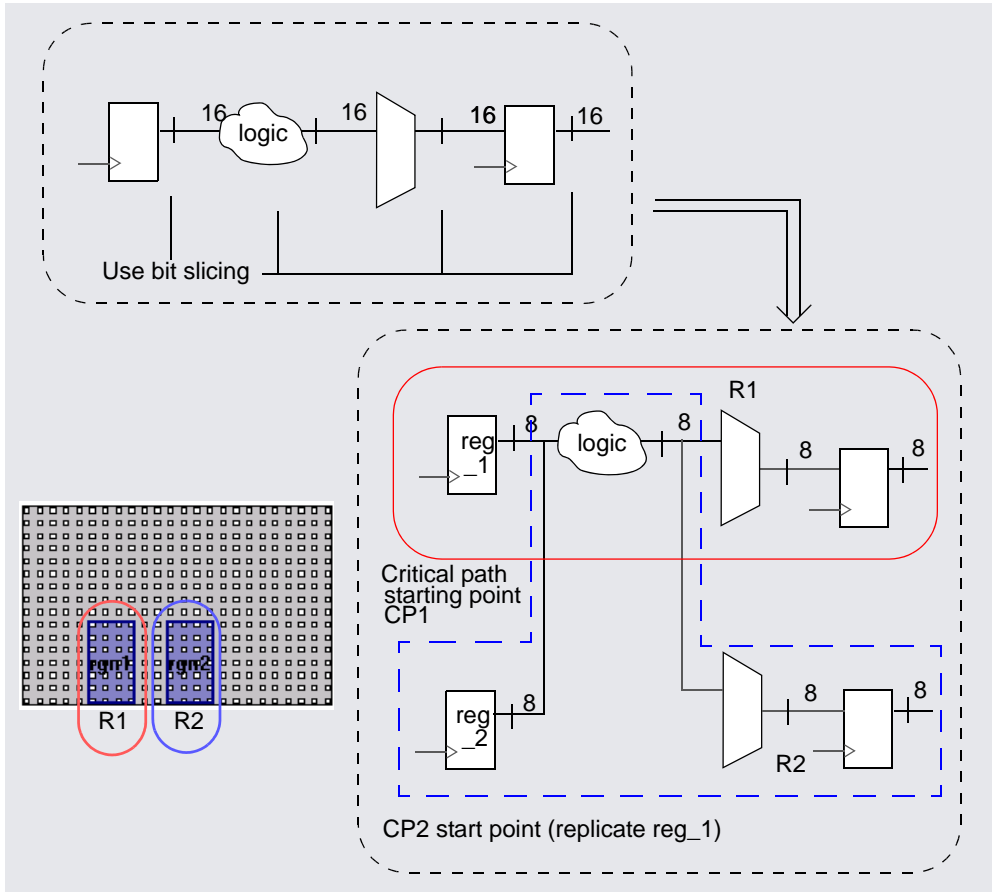


## Handling Critical Paths with Bit Slicing

If the critical path involves a datapath that is too wide for one region, you can use bit-slicing to divide the datapath.

1. Use bit-slicing to divide the datapath.

2. Replicate the register and place with the common logic of the critical path.
3. Place one half of the critical path in one region and the other half of the critical path in another region.

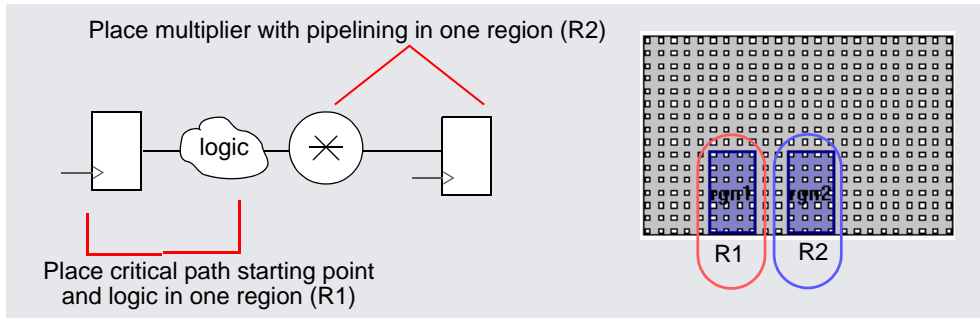


## Handling Critical Paths with Pipelining

If the critical path contains a register that follows either a multiplier or ROM that is pipelined, then create two regions as follows:

1. Place the critical path starting point and logic in one region.
2. Place the multiplier or ROM with the pipeline register in another region.

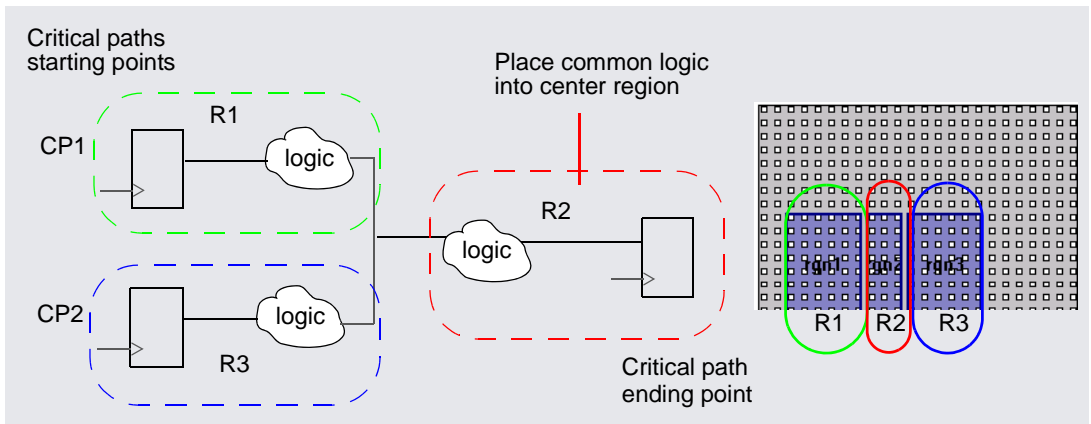




### Handling Designs with Multiple Critical Paths

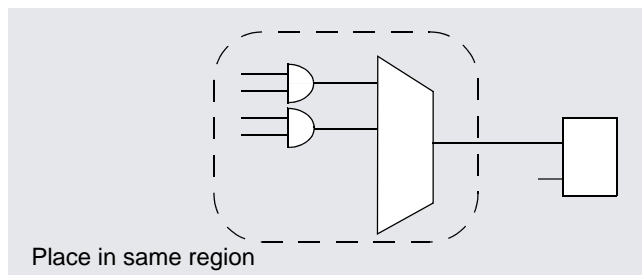
If a design has multiple critical paths that do not share the same start and end points, place them in separate regions.

If a design has multiple critical paths that share the same end point, try to constrain only the common logic in a separate region and constrain the other logic that is connected to each starting point registers in separate regions for each path. Place the common logic region between these regions to minimize distances.



### Handling Critical Paths with Large Multiplexers

If the critical path contains a large multiplexer, make sure that the region containing the mux also includes the control logic for that mux.



## Assigning Xilinx Block RAMs to Regions

Block RAM resources are limited. The Design Planner tool allows you to view the resources and create block RAM regions in the Design Plan Editor. This section includes information about the following:

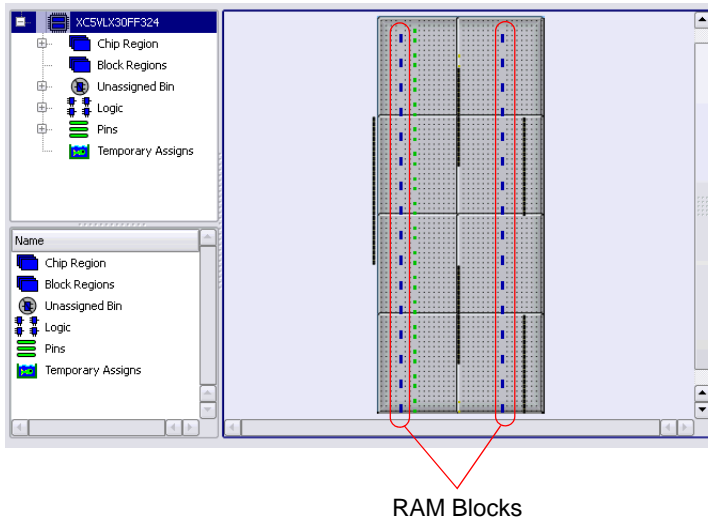
- [Viewing Block RAM Resources](#), on page 534
- [Creating Block RAM Regions](#), on page 535
- [Assigning RAM to Block RAM Regions](#), on page 536

### Viewing Block RAM Resources

The target FPGA device footprint displays the following resources: CLBs, Block RAMs (RAMB36\_FIFO36, RAMB16\_FIFO16, and BRAMs), and I/O pins. The coordinate system for block RAMs and CLBs starts at row=0, col=0. The height and width of the block RAMs is similar to the view in the Xilinx place-and-route tool.

| Family                                                       | Block RAM Height |
|--------------------------------------------------------------|------------------|
| Virtex-5 (RAMB36_FIFO36) and Virtex-4 (RAMB16_FIFO16) blocks | 2 CLB rows       |
| Virtex-II Pro, Virtex-II, and Spartan-3                      | 4 CLB rows       |
| Virtex and Virtex-E                                          | 4 CLB rows       |

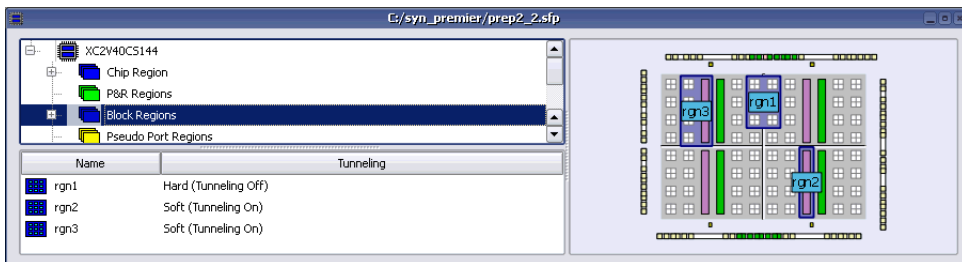
The following figure displays the RAMB36\_FIFO36 blocks on a Virtex-5 device.



To view information about a block RAM, use the tooltip, or right-click and select Properties when you have the RAM selected.

## Creating Block RAM Regions

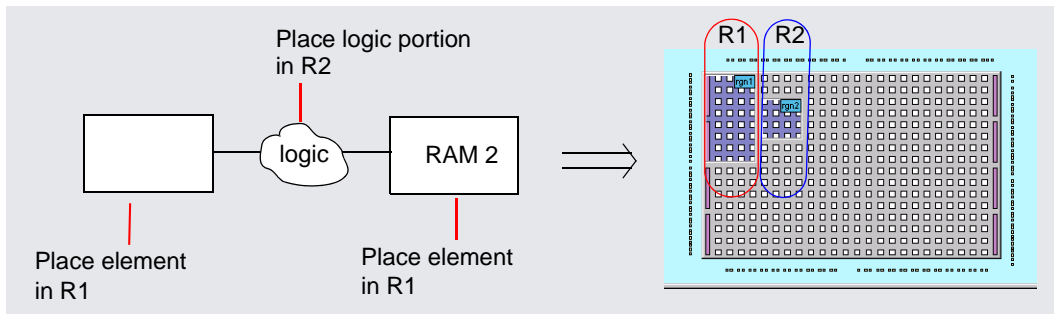
To create a block RAM region, use the procedure described in [Creating Regions, on page 505](#) and the Block Region Tool command. A tool tip displays the coordinate locations for CLBs and BRAMs and the capacity of the region, when you drag the cursor over these locations.



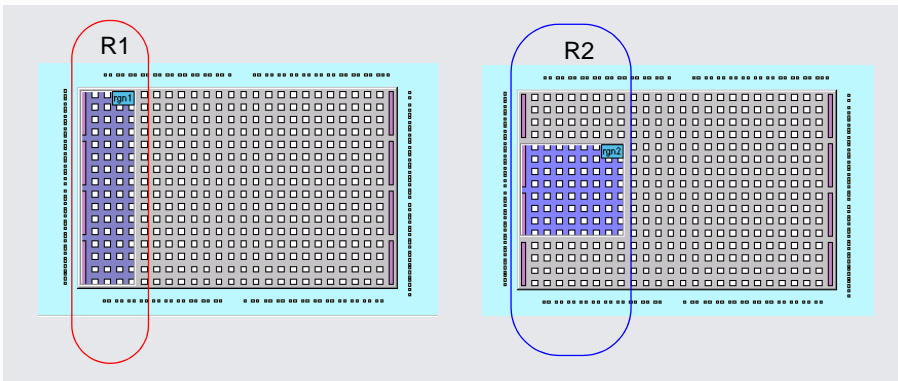
Note the following when you create and place block RAM regions:

- Block RAM regions can consist only of CLBs, only block RAMs, or a combination of CLBs and block RAMs.
- Block RAM regions can overlap.

- You can calculate region-to-region delay based on the CLB location. For regions consisting only of block RAMs, region-to-region delay can be calculated from a representative CLB location.
- If a critical path includes block RAMs, make sure that the region containing the critical path is close to or includes a sufficient number of block RAMs. Select a block RAM that is within or close to the region containing the rest of the critical path logic.
- If a critical path contains several RAMs, place block RAMs in one region and place standard logic in another region.



- If the block RAMs span all CLB rows, select the wider region R2 in the following figure instead of R1. R2 allows more area for routing



## Assigning RAM to Block RAM Regions

1. Turn on the rats nesting option so you can view the interconnect.
2. Create the block RAM region.

3. Assign the RAM by dragging and dropping the RAM modules from the HDL Analyst RTL view to the Design Planner view.
  - To automatically infer block RAM, assign the register driving the RAM and the block RAM to the same region.
  - To automatically infer block RAM in Virtex-II and Spartan-3 designs, assign the block RAM and its output register to the same region.
  - To automatically infer block RAM from instantiated block RAMs with non-Xilinx primitive names, specify the `syn_resources` "Blockrams=value" attribute in the HDL source code.

Once you have assigned the RAM, the resulting implementation from the Design Plan Editor is saved to a Design Plan file (.sfp). The location constraints for block RAM regions are written to a Xilinx netlist constraint file (.ncf), and these constraints are honored by the Xilinx place-and-route tool.

4. If you are using the `syn_ramstyle` attribute, follow these specification guidelines
  - Avoid mismatches between attributes specified in the SCOPE editor and the HDL code. If there is a conflict, SCOPE attributes take precedence.
  - To ensure that the RAM is assigned to the region and implemented in the way that you want, specify the attribute value appropriately:

**To implement the RAM as ...    Use this `syn_ramstyle` value...**

|                                           |                                                                           |
|-------------------------------------------|---------------------------------------------------------------------------|
| Block RAM                                 | <code>block_ram</code><br>Or, do not specify the attribute.               |
| Registers                                 | <code>registers</code><br>This floats RAM logic and generates a warning.  |
| Select RAM                                | <code>select_ram</code><br>This floats RAM logic and generates a warning. |
| Block RAM without doing read/write checks | <code>no_rw_check</code>                                                  |

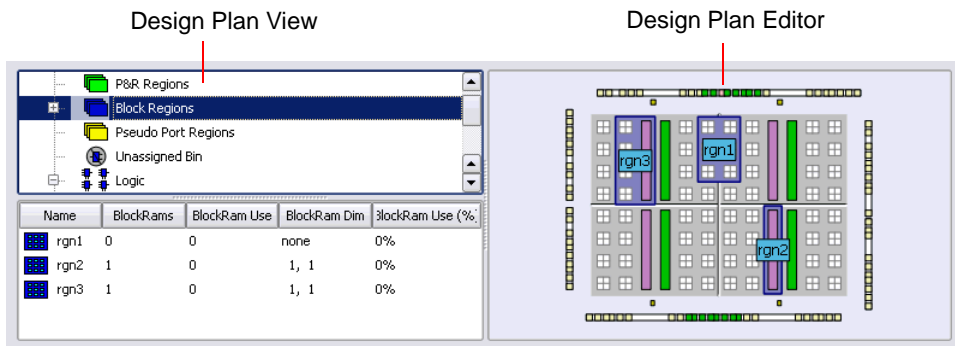
If you do not specify an attribute, logic RAMs assigned to a block RAM region are implemented as block RAMs.

- Estimate the area of the block RAM region to ensure that the logic RAM can fit into the region in the form in which they are implemented. See [Checking Utilization, on page 517](#) for a procedure.

| RAM Implementation                                               | Region Area Estimation Needs                                                                                                                            |
|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Block RAM                                                        | Ensure that block RAMs can fit into that region.                                                                                                        |
| Standard logic                                                   | The logic is not constrained and can float anywhere on the device. You must estimate the block RAM region to ensure all logic can fit into that region. |
| Block RAMs without standard logic, assigned to a CLB-only region | The block RAM logic is not constrained and can float anywhere on the device.                                                                            |

Over-utilized areas are displayed in orange.

- To display information like the dimensions of CLBs and BRAMs, the number of BRAMs, BRAM usage, and the percentage of BRAM usage for each block RAM region, do the following in the Design Plan view:
  - Right-click in the Design Plan view and select Show/Hide columns...
  - Select the information you want to display from the Select Columns dialog box. The Design Plan view displays the selected information.



## Assigning Xilinx Block Multipliers to Regions

Block mult resources are limited. If a critical path includes multipliers, make sure that the region containing the critical path is close to or includes a sufficient number of block mult resources. The Design Planner tool allows you to create block Mult regions in the Design Plan Editor. This can help you visualize where block Mults are placed on the device.

The following describe how to work with Xilinx block multipliers.

- [Viewing Xilinx Block Mult Resources](#), on page 539
- [Creating Block Mult Regions](#), on page 539
- [Assigning Multipliers to Block Mult Regions](#), on page 539
- [Assigning Xilinx DSP Blocks to Regions](#), on page 540

### Viewing Xilinx Block Mult Resources

Block mult resources have their unique coordinate system starting at location (row=0, col=0). The Design Planner lets you view block mult resources for Xilinx Virtex-II Pro, Virtex-II, and Spartan-3 devices. You can move the mouse cursor over any resource on the device to display a tool tip identifying its description. Once you create and assign logic to a block mult region, you can also display capacity and utilization results for these resources.

### Creating Block Mult Regions

To create a block mult region, use the procedure described in [Creating Regions, on page 505](#) and the Block RegionTool command. You can create block mult regions that consist only of CLBs, only block mults, or a combination of CLBs and block mults. Block mult regions can overlap.

Once you have created the region, you can calculate region-to-region delay based on the CLB location. For regions consisting only of block Mults, calculate region-to-region delay from a representative CLB location.

### Assigning Multipliers to Block Mult Regions

1. Turn on the rats nesting option so you can view the interconnect.
2. Create the block mult region.

3. Assign the mult by dragging and dropping it from the HDL Analyst RTL view to the Design Planner view.
4. Run area estimation by right-clicking in the Design Plan Editor and selecting Estimate Regions or Estimate All Regions. See [Checking Utilization, on page 517](#) for a detailed procedure.
5. Right-click in the Design Plan view and select Show/Hide Columns from the pull-down menu. Select the region usage criteria you want to display: BlockMults, Block Mult Use, and Block Mult Use (%).

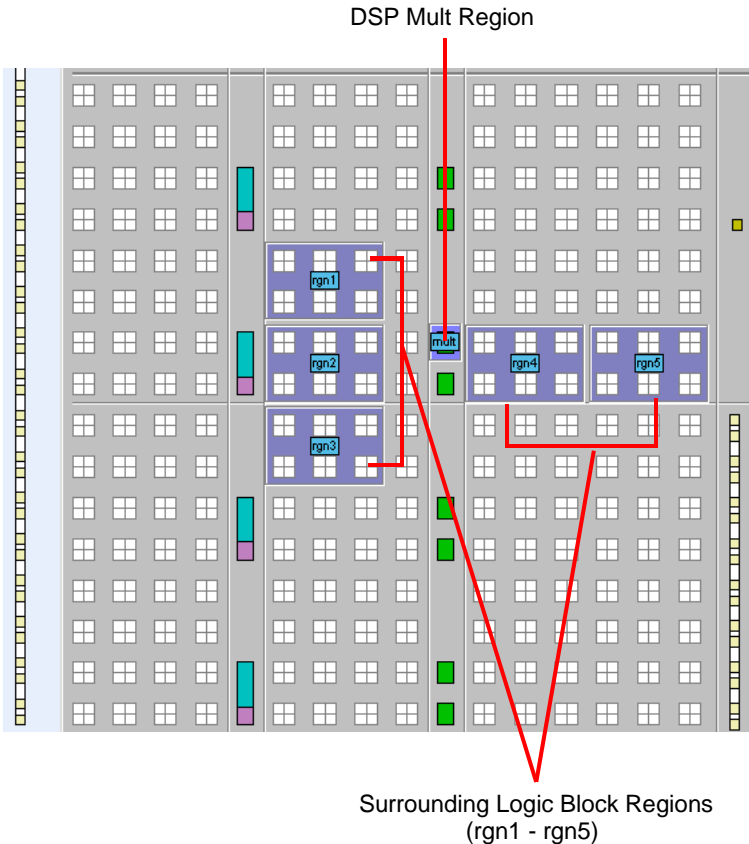
## Assigning Xilinx DSP Blocks to Regions

The DSP48 slices support many independent functions which include multipliers, multiplier accumulators (MACs), multipliers followed by an adder/subtractor, three-input adders, wide bus multiplexers, magnitude comparators, and wide counters.

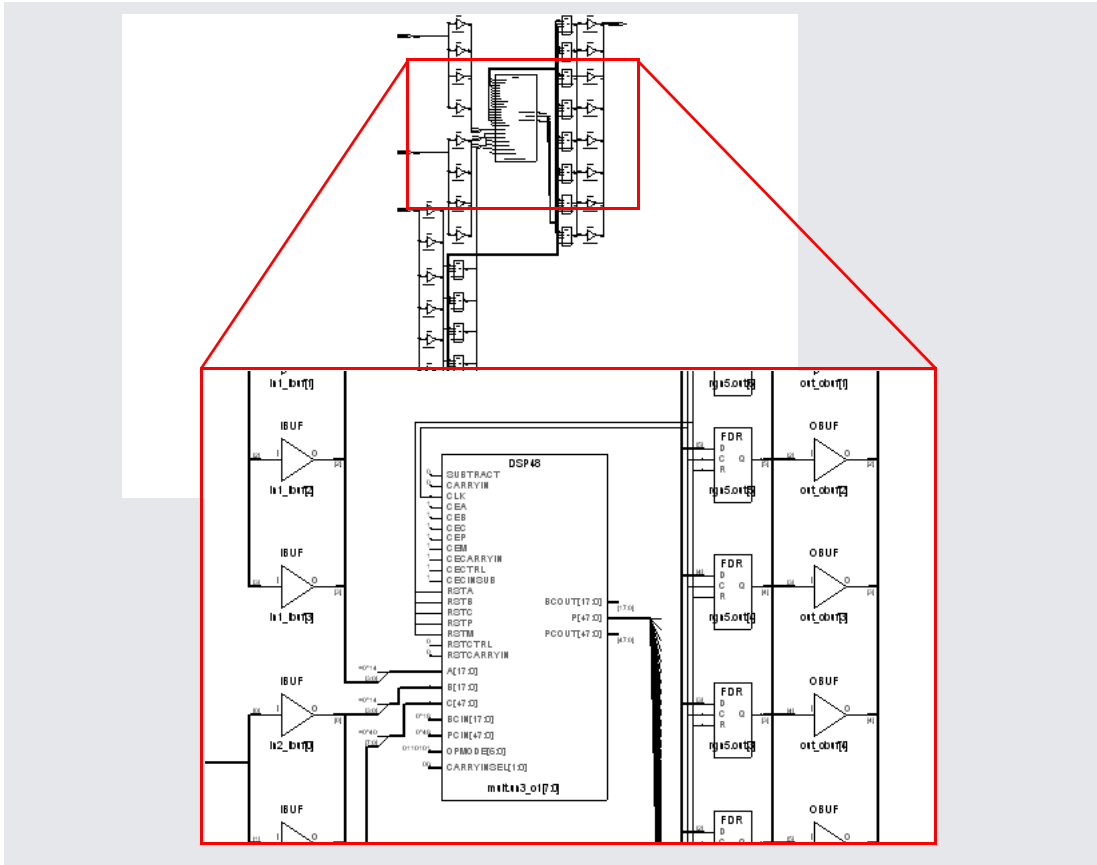
The Design Planner tool allows you to create DSP block regions and assign these components to them to constrain the logic. For example, you can assign a multiplier and its surrounding logic to a DSP region to constrain it to a region for synthesis.

The following figure shows part of the floorplan for a Virtex-4 or Virtex-5 device, where a multiplier is assigned to the mult region and its surrounding logic is assigned to regions rgn1 to rgn5.





After synthesis, the HDL Analyst Technology view shows that the multiplier and its surrounding logic are constrained to the DSP48 module.



## Using Process-Level Hierarchy

Depending on the technology you use, process-level hierarchy can affect design performance positively or negatively. The tendency is to affect Xilinx designs positively and Altera designs slightly negatively.

Process-level hierarchy is turned off by default in the Synplify Premier UI. The mapper treats designs with and without process-level hierarchy in the same way. However, if you have process-level hierarchy, there are extensive name changes, which can affect the mappers and the place-and-route tools.

# Bit Slicing

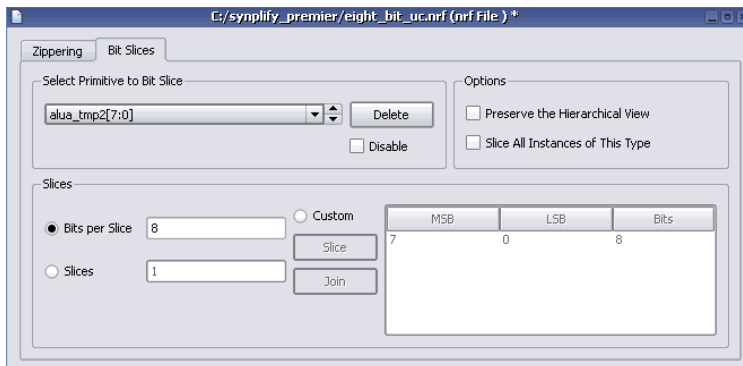
Bit slicing is a technique you can use when a primitive is too large to fit into a region, or when you want more granularity to control placement. It allows you to break up large primitives into smaller ones, which you can then place in different regions. To divide instances, use the zippering technique described in [Zippering, on page 550](#). If you are going to use both zippering and bit slicing, see the guidelines described in [Zippering Guidelines, on page 551](#).

The following describe bit slicing in more detail

- [Using Bit Slicing, on page 543](#)
- [Bit Slice Examples, on page 547](#)
- [Zippering, on page 550](#)

## Using Bit Slicing

1. In the Synplify Premier project window, open a new (File->New->Netlist Restructure File) or an existing .nrf file, then click the Bit Slices tab.



The .nrf file is a netlist restructure file that defines the logical division of primitive outputs. The tool reads the `slice_primitive` commands in this file which define the division. For information about this command and its use in a script, see `slice_primitive` in the [Chapter 14, Tcl Commands and Scripts](#) of the *Reference Manual*.

If you have an existing `.tcl` file, you can view it in this graphical interface by renaming it with a `.nrf` extension, and then opening it as described.

2. Type in or drag and drop the instance to slice from the RTL view into this tab.

For bit slicing, you can only divide bus primitives of the following types:

|          |     |          |
|----------|-----|----------|
| buf      | or  | register |
| inv      | xor | mux      |
| tristate | and | latch    |

If you used zippering on a module before bit slicing a primitive within the module, use the post-zippering name of the instance in the bit slicing command. To do this, run Compile Only (F7) after using zippering on a module, and then open the RTL view to get the new module instance name. Drag and drop the element to be bit sliced from the new RTL view.

3. Set bit-slicing preferences.
  - To slice an instance by a specified number of bits per slice or by a specified number of slices, see the details in [Slicing an Instance into a Specified Number of Slices, on page 545](#).
  - To divide an instance into slices of varying widths, see the procedure in [Custom Slicing, on page 545](#).
  - To globally bit slice all instances of the same type in the netlist, select Slice all instances of this type.

4. Save the file.

The Project view now shows the netlist restructure folder.

5. Select (Project->Implementation Options) and click the Netlist Restructure tab. Make sure that the netlist restructure file that you just created is checked in the Netlist Restructure Files section, and click OK.
6. Select Run->Compile Only (F7) to run netlist restructuring on your design. The sections of the sliced element are displayed and can now be individually assigned.

## Slicing an Instance into a Specified Number of Slices

The following procedure shows you how to slice an instance by a specified number of bits per slice or into a specified number of slices. To slice into varying bit widths, see [Custom Slicing, on page 545](#).

1. Open the Netlist Restructure tab, as described in [Using Bit Slicing, on page 543](#).
2. Enter a value.

| To create...                                     | Do this..                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Slices with a specified number of bits per slice | Click the <b>Bits per Slice</b> button and enter a value for the number of bits.<br>The tool allocates n instance for each group of bits, and allocates any remaining bits to the last instance. For an example, see <a href="#">Slicing into Primitives of Equal Size, on page 547</a> .              |
| A specific number of slices                      | Click the <b>Slices</b> button and enter a value for the number of slices. For an example, see <a href="#">Slicing into Predefined Primitives, on page 548</a> .<br>The tool divides the bits equally between the specified number of instances, and assigns any partial numbers to the last instance. |

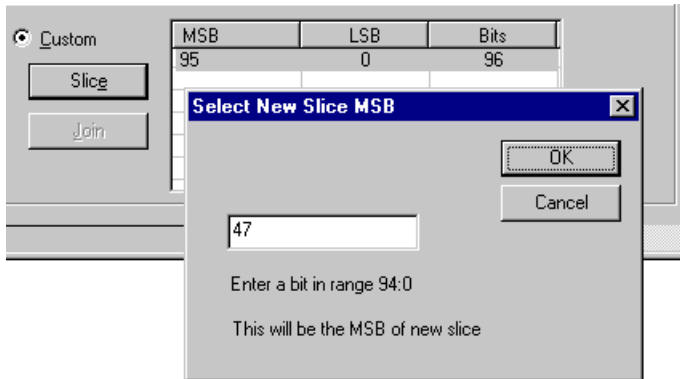
You can now return to the rest of the bit-slicing procedure described in [Using Bit Slicing, on page 543](#).

## Custom Slicing

The following procedure shows you how to define slices of varying widths. For a specified number of slices, see [Slicing an Instance into a Specified Number of Slices, on page 545](#).

1. Open the Netlist Restructure tab, as described in [Using Bit Slicing, on page 543](#).
2. Click the Custom button. This enables the MSB/LSB table and the Slice button.

3. To define a slice, do the following:
  - Select the top entry in the table, then click on the Slice button. This displays the Select New Slice MSB.



- Either click OK to slice the number of bits into two or enter the starting MSB for the next slice.

The upper limit of the bit range is always one less than the previously assigned MSB so that each slice is at least one bit wide. When you click OK, the table is updated and the Slice button is again enabled, so you can define a new slice.

4. Continue to select entries in the table and click Slice to redisplay the Select New Slice MSB popup menu and define the additional slices.

See [Slicing into Predefined Primitives, on page 548](#) for an example of custom slicing.

5. To undo an entry, merge the entries by doing the following:
  - Select two (or more) adjacent slice definitions by holding down the Ctrl key and clicking the table entries to select them.
  - Click Join.

You can now return to the rest of the bit-slicing procedure described in [Using Bit Slicing, on page 543](#).

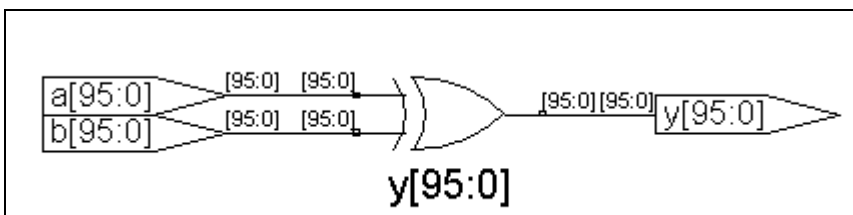
## Setting Viewing Options for Bit Slices

This procedure shows you how to set some viewing options.

1. To keep the current level as it is and only view the effects of bit slicing one level down in the design hierarchy, go to the Bit Slices tab of the netlist restructure file GUI and enable Preserve the Hierarchical View.
2. To view information about the bits, do the following:
  - Select a group of bits in an HDL Analyst view.
  - Right-click and select Properties. A dialog box displays the bit slicing properties for the primitive. Click OK to dismiss this dialog box. For additional information, see [Zippering Guidelines, on page 551](#).

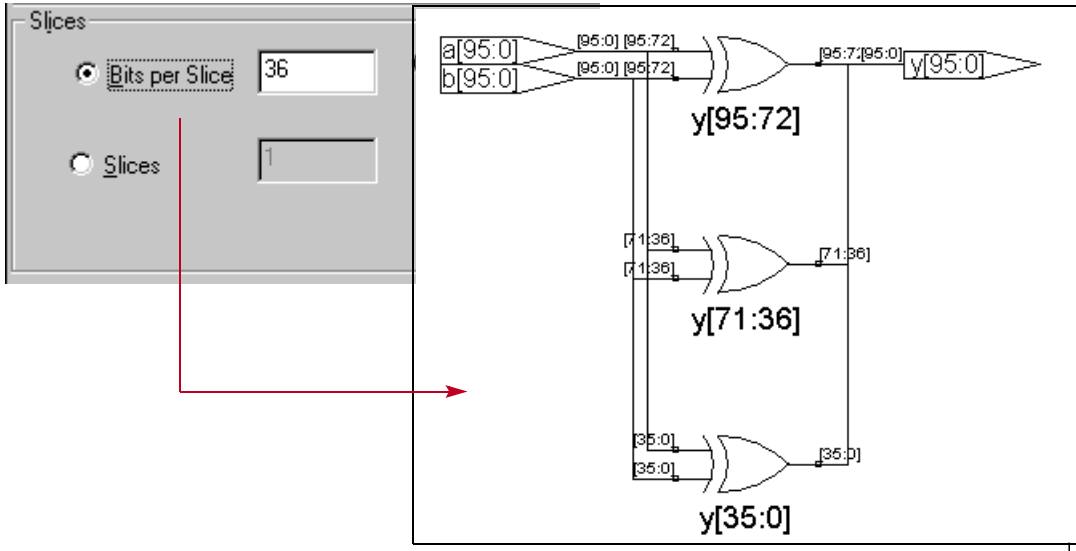
## Bit Slice Examples

The following examples illustrate two different cases of bit slicing a 96-bit bus XOR primitive. The following figure shows the primitive before bit slicing.



## Slicing into Primitives of Equal Size

In this example, the Bits per Slice value is set to 36. The tool divides the output of the  $y[95:0]$  primitive into three individual primitives. The first two primitives each contain the requested 36 bits; and the last primitive contains the remaining 24 bits ( $y[95:72]$ ). The following figure shows the results of this bit slicing RTL view.



## Slicing into Predefined Primitives

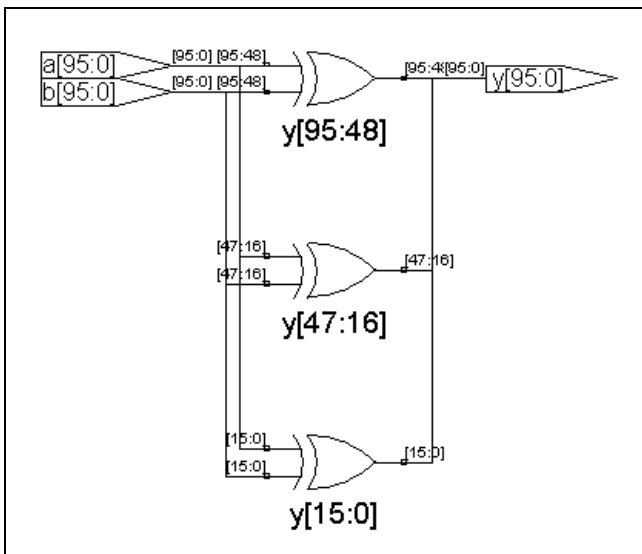
In this example, the Custom setting is used to define three individual primitives with widths of 48, 32, and 16. For more explanation about defining custom slices, see [Custom Slicing, on page 545](#).

The screenshot shows the 'Custom' configuration dialog with a table defining three slices. The 'Slice' button is highlighted.

| MSB | LSB | Bits |
|-----|-----|------|
| 95  | 48  | 48   |
| 47  | 16  | 32   |
| 15  | 0   | 16   |
|     |     |      |
|     |     |      |
|     |     |      |

The RTL view for this bit slicing example is shown in the following figure.





## Zippering

Zippering is a technique that divides a logic block that is too large to fit into a region into a number of smaller instances. The instances can then be placed in separate regions. To divide large primitives, use the bit slicing technique described in [Bit Slicing, on page 543](#).

Zippering works by dividing the outputs of a block into groups. Once divided, a “cone-of-logic” is traced back through the hierarchy to the input pins to create instances containing only the requisite logic for that cone. The tool replicates logic while calculating the cone of logic, based on the number of inputs in the cone.

This section contains information about the following:

- [Zippering Guidelines](#), on page 551
- [Using Zippering](#), on page 551
- [Zippering Example](#), on page 555

## Zippering Guidelines

The following guidelines apply if you are using both bit slicing and zippering:

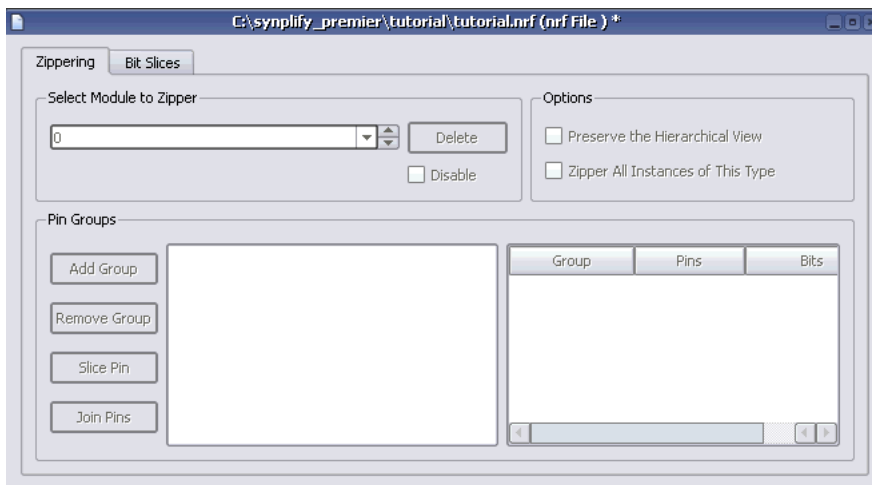
- You can combine zippering and bit slicing commands in a single `.nrf` file. Bit slicing commands are automatically placed before zippering commands, so as the file is read line-by-line, the primitives are sliced before any outputs are zippered.
- If you zipper a block before bit slicing a primitive in the block, the bit slicing command must use the post-zippering name of the instance. A simple way to do this is to recompile after zippering a block, push down into the RTL view of the new hierarchical block with the primitive, and then drag the primitive to the bit slicing UI.

Follow these guidelines for zippering:

- Zippering usually increases overall area utilization. Area can increase dramatically if you randomly select output groups.
- You can zipper at any level in the hierarchy above the leaf level. You must specify the full hierarchical instance name.
- After zippering, individual instances may not include all of the contents of the original instance.
- Hierarchical instances cannot be used when the Zipper all instances of this type box is checked (or the `-nl` option is used with the `zipper_inst_hier` command).

## Using Zippering

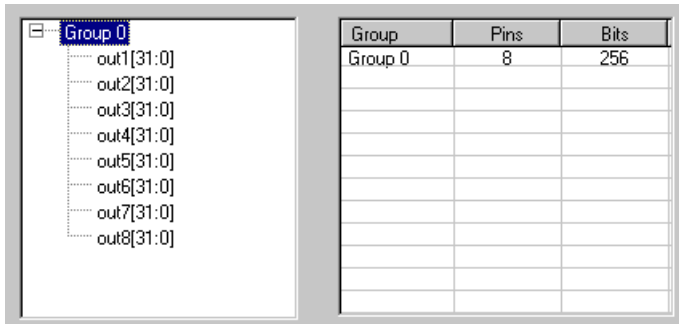
1. In the Synplify Premier project view, create a new file (File->New->Netlist Restructure File) or open an existing `.nrf` file. Then click the Zippering tab.



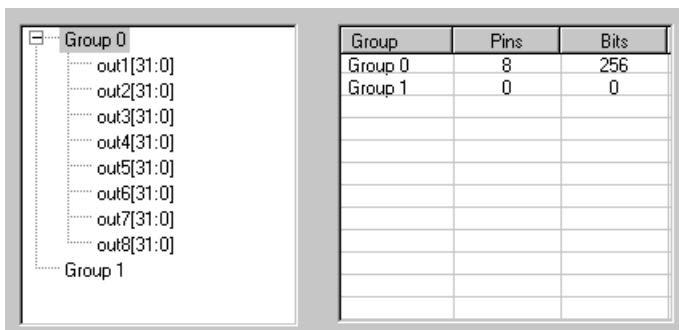
The `.nrf` file is a netlist restructure file that defines the logical division of primitive outputs. The tool reads the `zipper_inst_hier` commands in this file to determine the logical divisions for a module. Division boundaries are specified by identifying groups of output signals. For information about this command and its use in a script, see `zipper_inst_hier` in the [Chapter 14, Tcl Commands and Scripts](#) of the *Reference Manual*.

If you have an existing `.tcl` file, you can view it in this graphical interface by renaming it with a `.nrf` extension, and then opening it as described.

2. Drag and drop the block to be zippered from the RTL view to the UI.
3. Specify the input groups.
  - Click on the “+” sign in the Pin Groups window to expand Group 0. This displays the output nets.

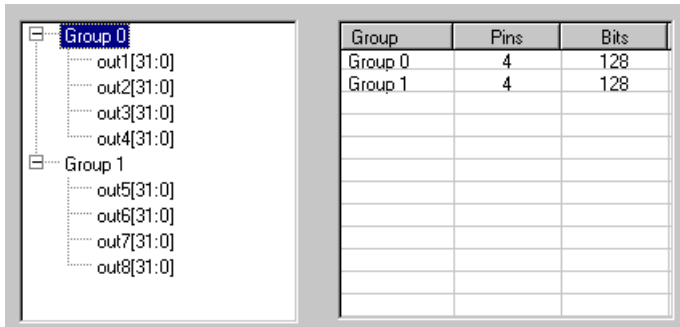


- To define a section for zippering, click on Add Group. This adds an empty group to the Pin Groups window. Each group represents a zippered section. Continue to add additional groups until you have the number you need.

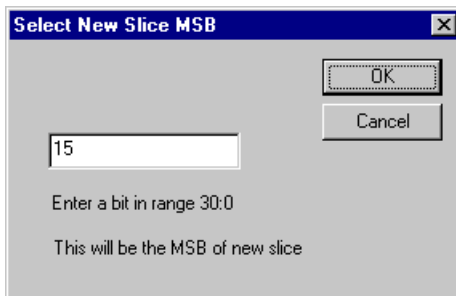


#### 4. Assign nets to groups.

- Click on a net in group 0 and drag the net to the new group. Use the Ctrl and Shift keys to select more than one net. The following figure shows net out5[31:0], out6[31:0], out7[31:0], and out8[31:0] dragged from Group 0 to Group 1. You can click the + sign to expand Group 1 and view the new assignment.



- To slice a bus nets and divide it into separate groups, click Slice Pin. To divide the bus into two slices, click OK in the resulting dialog box. To create more slices or specify custom widths, enter an MSB for the new (second) slice and click OK. You can then accept the displayed MSB for the new (next) slice or enter another MSB for the next slice. The upper limit of the bit range displayed is always one less than the MSB of the parent slice so that each slice is at least one bit wide.



If you split a bus net incorrectly, you can undo the split by selecting the nets using the Ctrl or Shift key and clicking Join Pins.

- Continue to arrange the groups by dragging nets to the individual groups. Make sure that you specify the groups for zippering judiciously, as sub-optimal zippering can significantly increase design size because of logic replication. For an example, see [Zippering Example, on page 555](#).
5. When all the nets are arranged in groups, save the file.
  6. Add the file to the project.

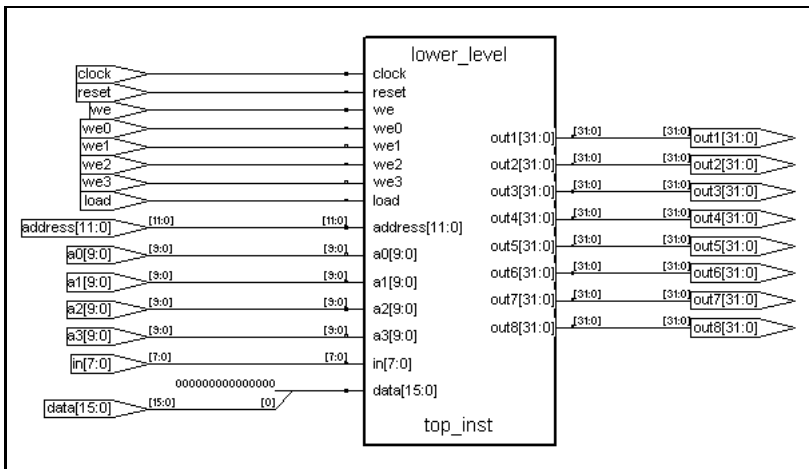
## 7. Assign zippered elements.

- Close the view and return to the Project view.
- Click Implementation Options and select the Netlist Restructure tab. Make sure that the netlist restructure file is checked.
- Run Compile Only (F7) on your design with the netlist restructure file and open the updated RTL view, which reflects the divided logic.
- Press F9 to rerun area estimation. Because of logic replication, zippering increases the total area of the design.
- Assign parts of the zippered element individually to different regions.

To view properties, select a group of instance pins, right-click and select Properties. A dialog box displays the zippering pin group properties of the module.

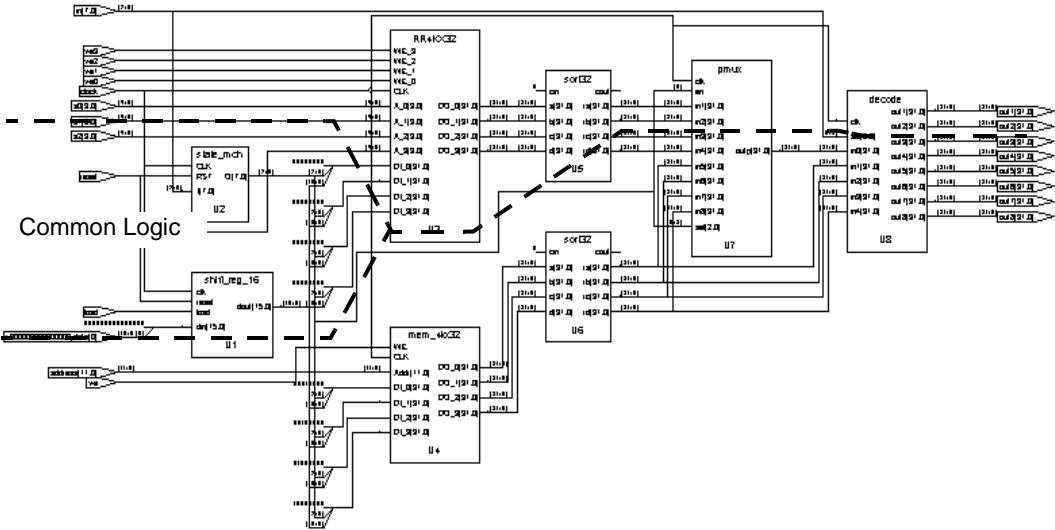
## Zippering Example

Zippering requires careful examination of your design to logically divide the block into an optimal number of instances and logical signal boundaries. This hierarchical block simple example, requires 325 I/O pins with 256 output pins for the eight output buses and 69 input pins.



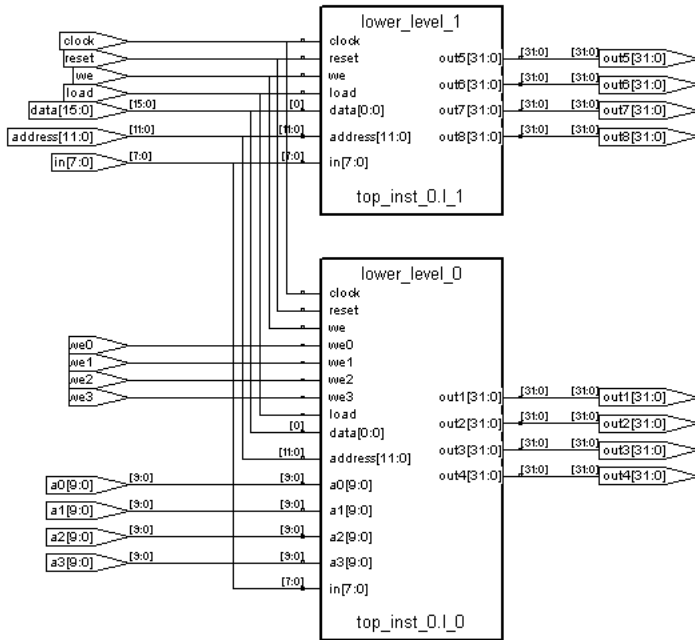
The number of I/O pins required for the block might be too large to fit into one region. If you go down one level and examine the hierarchy, you see that you can logically divide the block into two instances as shown by the broken

line in the next figure. The outputs from module top\_inst can be divided between two instances, which out1, out2, out3, and out4 in one instance and out5, out6, out7, and out8 in the other instance. Each instance would require a smaller number of output pins for the split buses, with some input pins and some replication of common logic. The two instances can be assigned to different regions.



The following figure shows the RTL view results after zippering. The original block is split into two instances. The first instance, top\_inst\_0.1\_1, contains the out5 through out8 buses, and the second instance, top\_inst\_0.1\_0, contains the out1 through out4 buses. The first instance requires 153 I/O pins, and the second instance requires 197 I/O pins.







**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 12

# Running Logical Compile Points

---

The following sections describe how to use the Logical Compile Point iterative flows:

- [Logical Compile-Point Synthesis](#), on page 560
- [About Compile Points](#), on page 562
- [Compile Point Synthesis](#), on page 571
- [Using Compile-point Synthesis](#), on page 573
- [Xilinx Compile-point Synthesis Flow](#), on page 583

# Logical Compile-Point Synthesis

This section describes the compile-point synthesis flow, which automates the traditional bottom-up flow for large designs. This feature is available with the Synplify Pro and Synplify Premier products, for use with certain technology families. This section includes the following topics:

- [Overview](#), on page 560
- [About Compile Points](#), on page 562
- [Compile Point Synthesis](#), on page 571
- [Using Compile-point Synthesis](#), on page 573

For information about technology-specific flows, see the following:

- [Quartus II Incremental Compilation](#), on page 857
- [Xilinx Compile-point Synthesis Flow](#), on page 583
- [Working with Xilinx Incremental Flows](#), on page 862

## Overview

The compile-point synthesis flow addresses the need for overall stability of a design while portions of the design evolve, as well as provides better runtime performance of the place-and-route tools. These needs are met by dividing design requirements into parts or *points* that can be processed separately.

For example:

- Portions of a design can be isolated to stabilize results. These parts are frozen as they are completed, while work continues, independently, on the rest of the design.
- To improve runtime place-and-route performance when processing large designs, some place-and-route tools let you decompose your design into portions that are processed incrementally.

Traditionally, to accomplish these tasks designers have resorted to bottom-up design and synthesis. Previously, designers who used bottom up flows have been limited in the following ways:

- Top-down synthesis produces better results, because it can optimize intelligently, taking into account the relations between the parts of a design.

- Bottom-up design has required designers to write and maintain sets of long, complex, error-prone scripts to direct synthesis and keep track of design dependencies. Typical scripts involve stitching, modeling, and ordering of compilations.

Compile-point synthesis is designed to handle these issues. For more details, see [Traditional Bottom-up Design and Compile Points](#), on page 561.

## Traditional Bottom-up Design and Compile Points

In a traditional bottom-up flow, a design is divided into parts that can be processed independently. Traditionally, this approach has been used in the following cases:

- Where parts of the design need to be isolated to stabilize results. The design team can freeze portions of the design as they are completed, while continuing to work independently on the rest of the design.
- To process large designs where a top-down approach is not possible because of memory and runtime limits. The bottom-up flow permits partial recompiles and multiprocessing to speed up design compilation.

For certain device technologies, the compile-point synthesis flow lets you design incrementally and synthesize designs that exceed runtime limits for top-down synthesis. For details about the supported compile-points flows and their respective generic flow diagrams, see [Logical Compile-Point Synthesis](#), on page 560.

Compile-point synthesis is a top-down flow that lets you choose the exact mix of incremental synthesis you need — and lets you change that mix at any time. You choose which parts of your design to synthesize and place-and-route independently. The compile-point synthesis lets you break down a design into smaller synthesis units or *compile points*. The software treats each compile point as a block for incremental mapping, and the design team can work on individual compile points independently of the rest of the design. A design can have any number of compile points, and compile points can be nested.

Because constraints are not automatically budgeted, manual time budgeting is important. Compile-point constraints directly affect quality of results. You must provide reasonably accurate timing constraints for each compile point. Then, because the synthesis tool manages design dependencies for you, working with compile points is no more difficult than working without them.

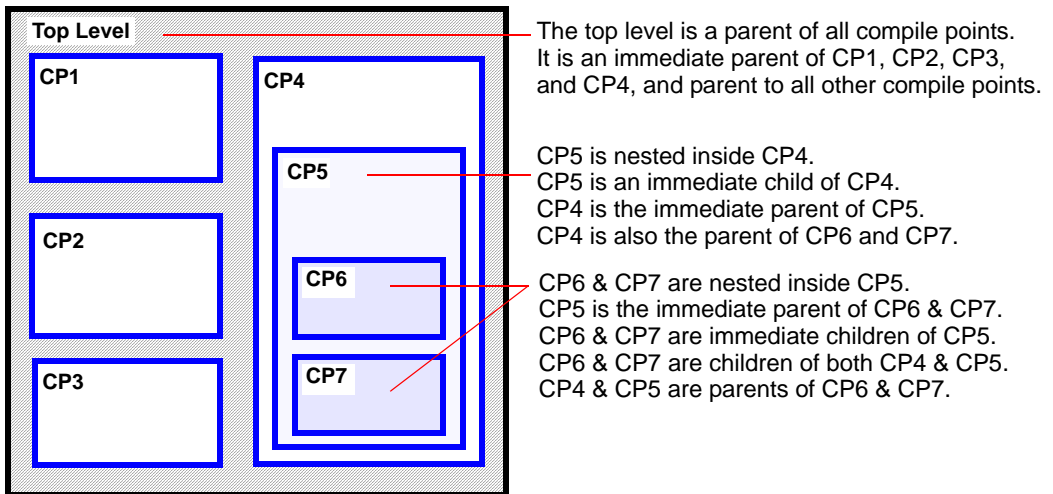
## About Compile Points

Compile points are parts of a design 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. Compile point topics include:

- [Nesting: Child and Parent Compile Points](#)
- [Advantages of Using Compile Points](#)
- [Compile Point Types](#)
- [Compile Point Feature Summary](#)
- [Using `syn\_hier` with Compile Points](#)
- [Using `syn\_allowed\_resources` with Compile Points](#)
- [`define\_compile\_point` and `define\_current\_design`](#)
- [About Interface Logic Models \(ILMs\)](#)

### Nesting: Child and Parent Compile Points

A design can have any number of compile points, and compile points can be nested inside other compile points. In the figure below, compile point CP6 is nested inside compile point CP5, which is nested inside compile point CP4.



To simplify things, the term *child* is used here to refer to a compile point that is contained inside another compile point; the term *parent* is used to refer to the compile point that contains the child. These terms are not used here 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.

## Advantages of Using Compile Points

Some advantages of using compile points in the design flow include:

- Smaller memory consumption compared to top-down processing without compile points.
- Incremental synthesis which maintains design stability and reduces run times.
- Runtime advantage for multiple instantiations of a compile point.

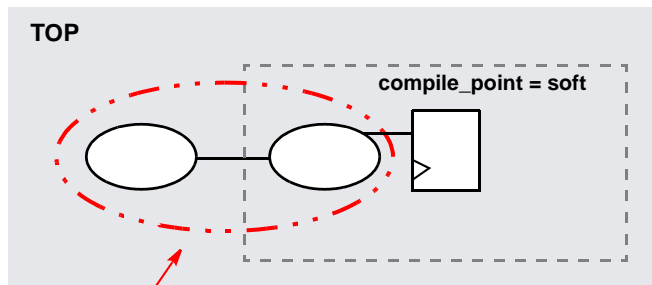
## Compile Point Types

You can control the amount of boundary optimizations for compile points using compile point types:

- [Soft](#)
- [Hard](#)
- [Locked](#) (default)
- [Locked, Partition](#)

### Soft

With type `soft`, the boundary of a compile point can be reoptimized during top-level mapping. Timing optimizations such as sizing and buffering and DRC logic optimizations can modify boundary instances of the compile point and combine the instances with functions from the containing design. The interface of the compile point can also be modified. Multiple instances are uniquified. Any optimization changes can propagate both ways: into the compile point and from the compile point to its parent.



Optimization of entire logic cone across boundary

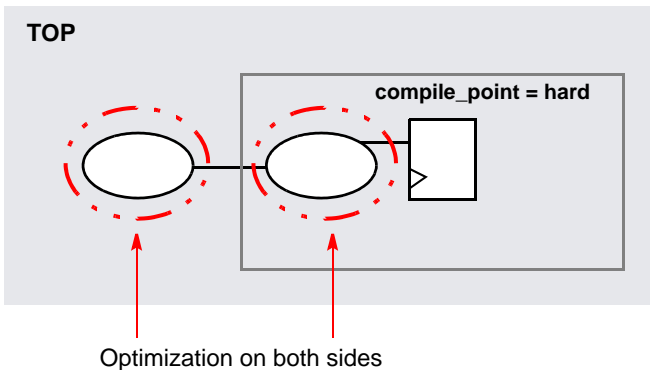
In the figure above, `compile_point` is represented with a dotted boundary to show that logic can be moved in or out of the compile point.

One advantage of using `soft` mode is that it usually yields the best quality of results compared to the other compile point types because the software is allowed to utilize boundary optimizations. However, a design using `soft` compile points can have longer runtime than the same design using `hard` or `locked` compile points.



## Hard

With type `hard`, 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 unquified.



In the figure above, `compile_point` is shown with a solid boundary to emphasize that no logic can be moved in or out of the compile point.

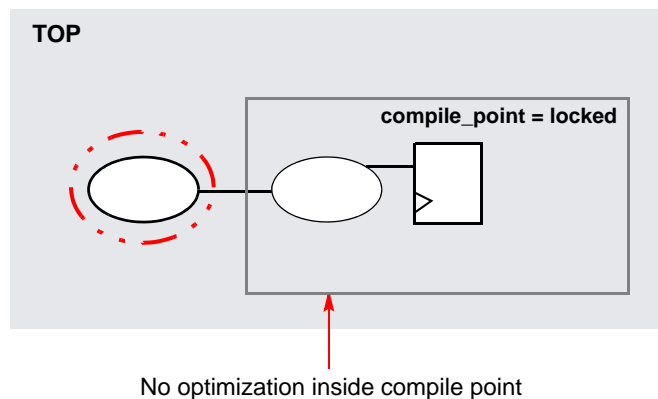
One advantage of using `hard` compile point type is that it allows for optimizations on both sides of the boundary without changing the boundary. There is some trade-off of quality of results to keep the boundaries which is usually done for verification of the sub-blocks. Using `hard` also allows for hierarchical equivalence checking for the compile point module.

## Locked

The locked compile point type is the default. When the compile point type is locked, there are no interface changes or reoptimization performed on the compile point during top-level mapping. An interface logic model (ILM) of the compile point is created (see [About Interface Logic Models \(ILMs\)](#), on page 570) and included for the top-level mapping. The ILM requires less memory than the whole netlist because the model contains only the paths necessary to provide an accurate timing model for top-level mapping. This 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, the multiple instances of the compile point module remain identical even though the compile point is unquified.

You will see unique names for multiple instances of the compile points in the Technology view (.srm file). However, in the final Verilog netlist (.vma file) the original module names for the multiple instances are restored.

Timing optimization can modify only instances outside the compile point. Although used to time the top-level netlist, changes do not propagate into or out of a locked compile point.



In the figure above, `compile_point` is shown with a solid boundary to emphasize that no logic is moved in or out of the compile point during top-level mapping.

Some advantages of using locked mode include:

- Consumes smallest amount of memory. Also used for large designs because of this memory advantage.
- Provides most runtime advantage compared to other compile point types.
- Allows for obtaining stable results for a completed part of the design.
- Allows for hierarchical place and route with multiple output netlists for each compile point and the top-level output netlist.
- Allows for hierarchical simulation.

Some limitations of using locked mode include:

- Automatic time budgeting
- Gated clocks or generated clocks
- Constant propagation
- Tristate pads embedded within compile points

- BUFG insertion
- GSR hookup
- IO pads, such as IBUF/OBUF buffers, should not be instantiated within compile points

This mode has the largest trade-off of quality of results because no boundary optimizations can occur. For this reason, it is very important to provide accurate constraints for the locked compile point.

### Locked, Partition

The locked compile point type can also be used with the partition option. When you specify type as locked, partition, the synthesis tool generates a netlist file for the compile points that are defined. Each compile point also includes a timestamp, for example, when the module was last synthesized.

This mode provides place and route runtime advantages and allows for obtaining stable results for a completed design. However, this mode has the largest trade-off of quality of results because no boundary optimizations can occur.

## Compile Point Feature Summary

The following table provides a summary of how compile points are handled 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          |

| Features                             | Compile Point Type |     |                                                              |
|--------------------------------------|--------------------|-----|--------------------------------------------------------------|
| Hierarchical Equivalence Checking    | no                 | yes | yes                                                          |
| Interface Logic Model (created/used) | no                 | no  | yes                                                          |
| Final Netlist                        | one                | one | one top-level netlist and one netlist for each compile point |

## Using `syn_hier` with Compile Points

For `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.

## Using `syn_allowed_resources` with Compile Points

Apply the `syn_allowed_resources` attribute 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 is adequate resources available during the top-down flow.

For more information about this attribute, see [syn\\_allowed\\_resources Attribute, on page 957](#).

## `define_compile_point` and `define_current_design`

You can only set a compile point through the SCOPE interface, which creates the `.sdc` file, or directly in the `.sdc` file. The `define_compile_point` command is automatically written to the top-level constraint file for each compile point you define. The following figure provides an example:

```

constraint\eight_bit_uc.sdc (constraint)
00058 # Attributes
00059 #
00060 define_attribute {n:resetn} syn_ideal_net {1}
00061
00062 #
00063 # Compile Points
00064 #
00065 define_compile_point {v:work.prgm_cntr} -type {locked} |
00066
00067 #
00068 # Other Constraints
00069 #

```

This top-level .sdc file has one locked compile point, `prgm_cntr` and uses the following syntax for defining the compile point:

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

The first command in a compile point constraint file is `define_current_design` which specifies the compile point module for the contained constraints. For example:

```

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 define_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}

```

```
define_current_design {prgm_cntr}
```

When running synthesis this command sets the context for the constraint file. The remainder of the file is similar to the top-level constraint file.

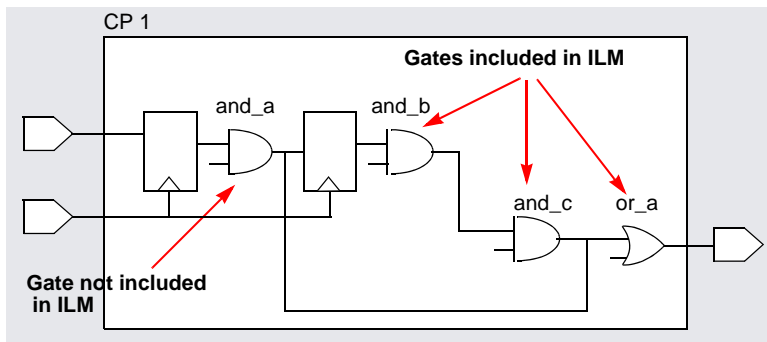
## About Interface Logic Models (ILMs)

An interface logic model (ILM) of a locked compile point is a timing model that contains only the interface logic necessary to provide accurate timing. An ILM is a partial gate-level netlist and is output in Verilog to a file with a `.vmd` extension. This partial netlist is a smaller file than the full netlist, which is helpful in managing capacity, especially for larger designs. Using ILMs improves the runtime for static timing analysis without compromising timing accuracy. An ILM represents the original design accurately while requiring less memory during mapping.

No timing optimizations are done 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:

- input/inout port to internal register
- internal register to output/inout port
- input/inout port to output/inout port

Internal register to internal register paths are removed from the model. The following figure provides an 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.

# Compile Point Synthesis

Synthesis consists of two phases: compiling and mapping. During synthesis, the design is first compiled, then mapped starting with the compile points at the lowest level of hierarchy in the design. After the compile points are mapped, the top-level is mapped. For more information, see:

- [Compile Point Optimization](#), on page 571
- [Forward-annotation of Compile-point Timing Constraints](#), on page 572

## Compile Point Optimization

Compile points are optimized separately from their parent environments (the containing compile-point or top-level), so they are unaffected by critical paths or constraints in those environments. A compile point stands on its own, with its own individual constraints.

You must set constraints on individual locked compile points, since top-level constraints do not propagate down to them. However, compile point timing models are taken into account when synthesizing higher levels, so you need not duplicate compile-point timing constraints at the top level.

During synthesis, any compile points that have not yet been synthesized are synthesized before the top level. Nested compile points are synthesized before the parent compile points that contain them. In a previous figure above, CP6 is synthesized before CP5, which is synthesized before CP4.

A compile point that has already been synthesized is not resynthesized, unless at least one of the following is true:

- You change the HDL source code defining the compile point in such a way that the design logic is changed.
- You change the constraints applied to the compile point.
- You change any of the options on the Device panel of the Implementation Options dialog box (except Update Compile Point Timing Data). In this case the entire design is resynthesized, including all compile points. See [Device Panel, on page 140](#).
- You intentionally *force* the resynthesis of your entire design, including all compile points – see Run -> Resynthesize All, [Run Menu, on page 181](#).

- 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. *Note:* This includes the possibility that the child was remapped earlier, while the option was disabled. The newly enabled option then requires that the updated timing model of the child be taken into account, by resynthesizing the parent.

The synthesis tool automatically detects design changes, and it resynthesizes compile points only when necessary. A compile point is not resynthesized, for example, just because you add or change a source code comment; since such a change does not really affect the design.

Incremental synthesis of a compile point results in the creation of intermediate mapping files (.srd), which are located in a subdirectory named after the compile point. You need not be concerned with these files. They are used to save mapping information for subsequent synthesis runs. (If you happen to delete them, 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 an associated constraint file. When constraints are forward-annotated to placement and routing, they are included from all these files. However, constraints on ports in the interface of a compile point are not forward annotated.

More precisely, the constraints defined in the constraint file for a compile point are of two kinds:

- Constraints applied to the interface (ports and bit ports) of the compile point. These include `input_delays`, `output_delays`, and clock definitions on the ports. Such constraints are only used when mapping the compile point itself, not its parents. They are not used in the final timing report, and they are not forward -nnotated.
- Constraints applied to instances inside the compile point, such as timing exceptions and internal clocks. Such constraints are used when mapping the compile point and its parents. They are used in the final timing report, and they are forward-annotated.

Constraints on top-level ports are always forward annotated.



## Using Compile-point Synthesis

This section provides information on how to use logical compile-point synthesis with the Synplify Pro or Synplify Premier products. Note that compile points are not available with the Synplify tool.

For a general description of the flow, see the [Synplify Pro and Synplify Premier Compile-point Flow](#), on page 573.

For vendor-specific flows, see:

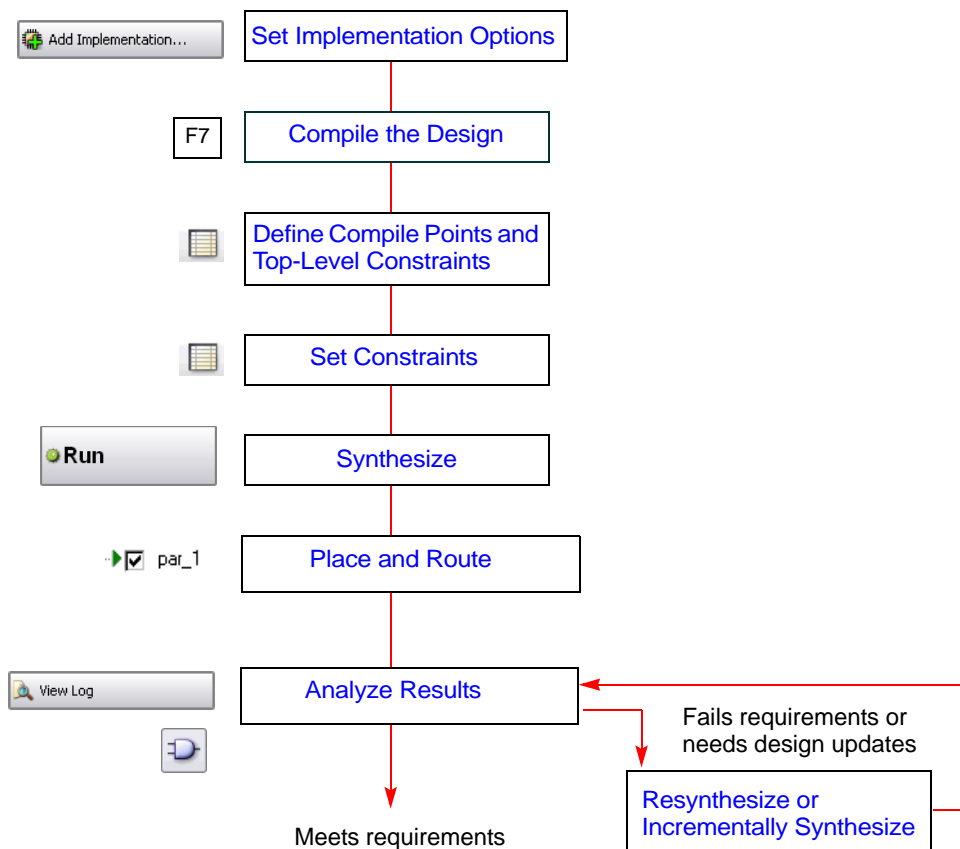
- [Quartus II Incremental Compilation](#), on page 857
- [Xilinx Compile-point Synthesis Flow](#), on page 583
- [Working with Xilinx Incremental Flows](#), on page 862

### Synplify Pro and Synplify Premier Compile-point Flow

This section describes the compile-point- synthesis flow, which contains the following steps to automate the traditional bottom-up flow for large designs.

- [Set Implementation Options](#), on page 574
- [Compile the Design](#), on page 575
- [Define Compile Points and Top-Level Constraints](#), on page 575
- [Set Constraints](#), on page 577
- [Synthesize](#), on page 580
- [Analyze Results](#), on page 580
- [Resynthesize or Incrementally Synthesize](#), on page 581

The following figure shows the generic procedure for using the Synplify Pro or Synplify Premier compile-point flow. For Actel designs, follow this generic flow.



## Set Implementation Options

The first step in compile-point synthesis is to set the implementation options, just as with the regular design flow.

1. Start the Synplify Pro tool or Synplify Premier; set up a design project for the compile-point flow and open the project for the top-level design.
2. Press the Implementation Options button in the Project view to open the Implementation Options dialog box.
3. Set the following:

- Select a technology that supports compile points and set the device, part and speed grade options.
- Set the global frequency, and any other optimization options.

For the Synplify Premier tool:

- On the Netlist Restructure tab, disable the Netlist Optimization Options.

*Note:* Synplify Premier MultiPoint synthesis ignores the following optimizations for compile points: Feedthrough Optimization, Constant Propagation, and Create Always/Process Level Hierarchy.

- For certain Altera devices, disable Create MAC Hierarchy as well.

You are now ready to compile the design ([Compile the Design, on page 575](#)).

## Compile the Design

After setting the implementation options, you must compile the design. This is the second step in the design flow.

1. Open the project for the top-level design.
2. Press F7 or select Run->Compile Only.

This compiles the design and enables the SCOPE constraints file to be initialized, which is important for the defining the compile points and their constraints, later in the flow.

The next step is to define compile points ([Define Compile Points and Top-Level Constraints, on page 575](#)).

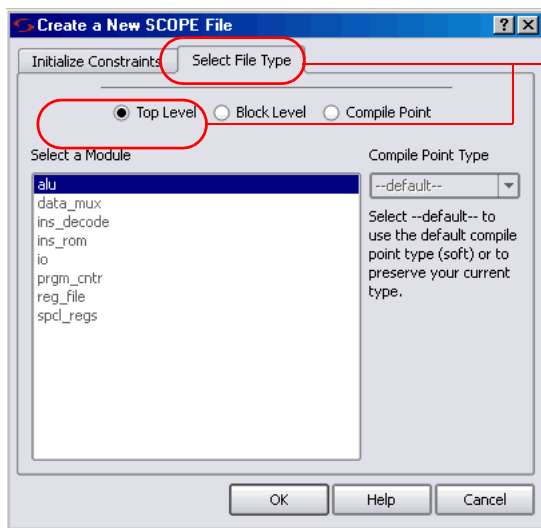
## Define Compile Points and Top-Level Constraints

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 keeps the two steps separate.

You define compile points in a top-level constraint file. See [About Compile Points, on page 562](#) for details about compile points. You can add the compile point definitions to an existing top-level .sdc file or create a new file.

1. Open a SCOPE window for the top-level 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. Select the Select File Type tab, click Top Level, and click OK.



Click and select.

Alternatively, you can create a new top-level constraint file when you create the module-level constraint files, as described in [Create Compile Point and Top-Level Constraint Files](#), on page 578.

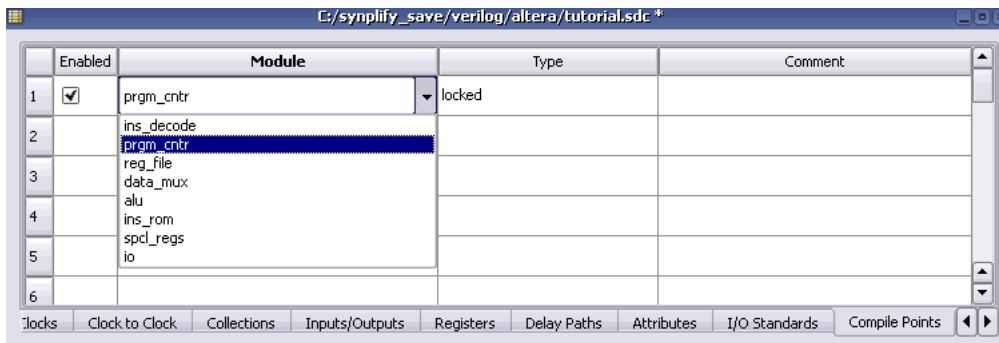
The SCOPE window opens.

## 2. Click the Compile Points tab.

- Set the module you want as a compile point using either of these methods: select a module from the drop down list in the Module column, or drag the instance from the HDL Analyst RTL view to the Module column.
- The Type can be specified as locked, locked,partition, hard, or soft.

For a description of these types, see [Compile Point Types](#), on page 564.

This tags the module as a compile point. The following figure shows the the prgm\_cntr module set as a locked compile point in the design flow.



3. Set any other top-level constraints like input/output delays, clock frequencies or multicycle paths.

The parent level includes lower-level constraints. The software considers the lower-level constraints when it maps the top level.

4. Save the top-level .sdc file.

You can now set constraints as described in [Set Constraints, on page 577](#).

## Set Constraints

You can specify constraints for each compile point in individual .sdc files, as well as set separate top-level constraints for the entire design. You need a compile point constraint file for each locked compile point, and a constraint file for the top level. Do not define the compile point constraints in the same file as the top-level constraints.


If you supply a constraint file for soft and hard compile points, the compile point timing models are taken into account and optimized separately during this bottom-up synthesis. However, further optimizations can occur during the top-down synthesis to help improve timing performance and overall design results.

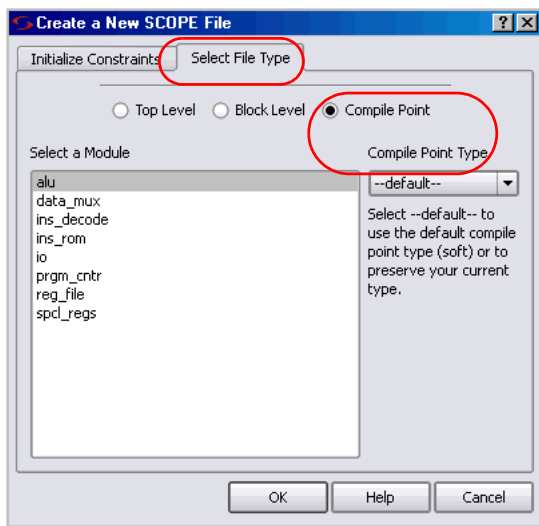
See the following sections for details about compile point constraints:

- [Create Compile Point and Top-Level Constraint Files](#), on page 578
- [Set Compile Point Constraints](#), on page 579

## Create Compile Point and Top-Level Constraint Files

You can create a module (compile point) constraint file as follows. Optionally, you can generate a top-level constraint file at the same time that you define the compile points.

1. In an open project, click the SCOPE icon (  ). The Create a New SCOPE File dialog box opens.
2. Click the Select File Type tab and click the Compile Point option.



3. Select the module you want to make a compile point.
4. Click OK.

If you do not have a top-level file, you are prompted to create one. If you have multiple top-level files, you can choose one or create a new one by clicking New. For information about defining compile points in a top-level file, see [Define Compile Points and Top-Level Constraints, on page 575](#).

5. Click OK to exit the prompt box, and then click OK again in the Create a New SCOPE File dialog box to initialize the constraints.

Two SCOPE windows open, one for the top-level and one for the compile point constraint file. You must define constraints for both the top-level and the compile point. See [Set Compile Point Constraints, on page 579](#)

for details about setting compile point constraints. Set top-level constraints as in a normal design flow.

## Set Compile Point Constraints

To create or modify the compile point constraints, do the following:

1. If needed, open the SCOPE window for the compile point constraint file by double-clicking the file in the Project view.

This opens the constraint file for the compile point. The name of the compile point file appears in the banner of the SCOPE window. Note that there is no Compile Point tab in the SCOPE UI 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 |
|---|-------------------------------------|--------------|-------------|-----------------|-------------|--------------------|--------------|--------------|----------------|------------|--------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | clock        |             | 120             | 8.33333     | default_clkgroup_0 |              |              |                |            | <input type="checkbox"/> |         |
| 2 |                                     |              |             |                 |             |                    |              |              |                |            |                          |         |
| 3 |                                     |              |             |                 |             |                    |              |              |                |            |                          |         |

2. 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.

You must set compile point constraints because parent constraints do not propagate down to the compile points. However, compile point constraints are considered while mapping the parent, so you do not need to duplicate compile point constraints at the top level. Compile point port constraints are not used at the parent level, because compile point ports do not exist at that level.

If you want to use the `syn_hier` attribute with a compile point, the only valid value is `flatten`. The software ignores any other value of `syn_hier` for compile points. The `syn_hier` attribute behaves normally for all other module boundaries that are not defined as compile points.

3. Save the file. When prompted, click **Yes** to add the constraint file to the top-level design project.

The software writes a file *name\_cp\_number.sdc* to the current directory.

## Synthesize

After you have set up the compile points and the constraints, you can synthesize the design.

1. Click **Run** and synthesize the top-level design.

The design is synthesized in two phases:

- First, compile points are synthesized from the bottom up, starting with the compile point at the lowest level of hierarchy in the design. Each compile point is synthesized independently. For each compile point, the software creates a subdirectory named after the compile point, in which it stores intermediate files for the compile point: RTL netlist, mapped netlist, and model file. The model file contains the hierarchical interface timing and resource information that is used to synthesize the next level.

When a design is resynthesized, compile points are resynthesized only if source code logic or constraints have been changed. If a compile point has not changed, the model file from the previous run is used. Once generated, the model file is not updated unless there is an interface design change or you explicitly specify it.

- After all the compile points are synthesized, the software synthesizes the design from the top down, using the model information for each compile point.

The software writes out a single output netlist and one constraint file for the entire design.

## Analyze Results

The software writes timing and area results to one 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 Log 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.
  - 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.
  - 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 the RTL and Technology view schematics for a graphic view of the design logic.

Note that 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.

## **Resynthesize or Incrementally Synthesize**

This is an optional step. You can resynthesize a locked compile point or synthesize your design incrementally. To obtain the best results, you should also define any required constraints and set the proper implementation options for the compile point before resynthesizing.

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).
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.

3. 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.

4. 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.

# Xilinx Compile-point Synthesis Flow

You can use the Synplify Pro or Synplify Premier compile-point synthesis flow in conjunction with the Xilinx place-and-route tool to design and lock down a design, one block at a time. For compile-point synthesis, the design is divided into compile points, which are hierarchical logic blocks (modules) that you can optimize independently from the rest of the design. The design team can work on individual modules separately and concurrently, and then integrate them into the top-level design using the compile-point synthesis flow. See [Using Xilinx Compile-point Synthesis, on page 583](#), for the following compile-point synthesis procedure specific to Xilinx.

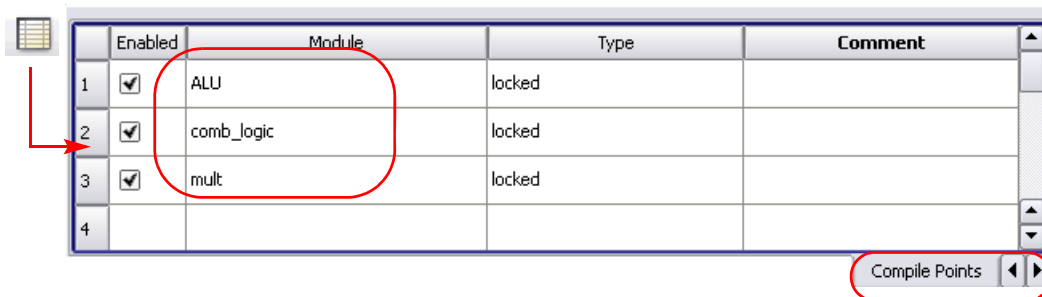
See Also:

- The Xilinx partition flow for incremental design changes is used in conjunction with the compile-point synthesis flow (see [Working with Xilinx Incremental Flows, on page 862](#)).
- For a generic procedure (that is not vendor-specific) on how to use the compile-point flow, see [Synplify Pro and Synplify Premier Compile-point Flow, on page 573](#).

## Using Xilinx Compile-point Synthesis

To implement Synplify Pro or Synplify Premier compile-point synthesis with the Xilinx compile-point synthesis flow, follow these steps:

1. Set up a project, set implementation options, and compile the project
  - Set up a project as usual, select the Xilinx target device.
  - Set the implementation options.  
On the Netlist Restructure tab of the Synplify Premier tool, make sure to disable all the Netlist Optimization Options.
  - Compile the design.
2. Define compile points in the top-level .sdc file.
  - Click the Compile Points tab, and set compile points. The following example shows three compile points set: ALU, comb\_logic, and mult.

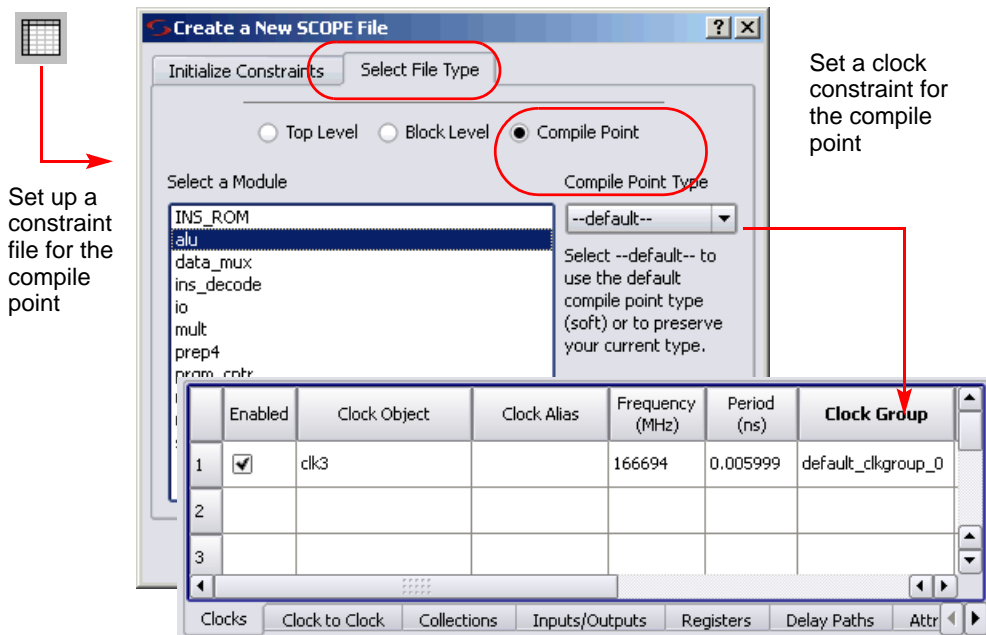


|   | Enabled                             | Module     | Type   | Comment |
|---|-------------------------------------|------------|--------|---------|
| 1 | <input checked="" type="checkbox"/> | ALU        | locked |         |
| 2 | <input checked="" type="checkbox"/> | comb_logic | locked |         |
| 3 | <input checked="" type="checkbox"/> | mult       | locked |         |
| 4 |                                     |            |        |         |

Compile Points

A compile point is a module that is treated as a block for incremental mapping. In subsequent synthesis iterations, the software does not resynthesize the compile point unless the original RTL netlist for the compile point changes.

3. Create a compile point constraint file for each compile point.
  - Click the SCOPE icon.
  - In the Create a New SCOPE File dialog box, click the Select File Type tab, then click Compile Point, and select the compile point. The following examples shows v:work.alu selected.
  - In the next dialog box, select the top-level .sdc file that defines the compile points.
  - Set the clock constraint for the compile point. This can be the same as the top level. Save the file.



4. Synthesize your design and check the compile point summary in the log file.

The software synthesizes the design from the bottom up, starting with the compile point at the lowest level.

5. Place and route the design.
6. To synthesize the design incrementally, do the following:
  - Make the design changes needed in the compile points.
  - Click Run to resynthesize your design incrementally.

The synthesis software runs incrementally, only resynthesizing compile points whose logic, implementation options, or timing constraints have changed.

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.

### 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
=====
```

Syntax changes only; not resynthesized

Logic changes; compile  
point resynthesized

### Incremental Run Log Summary

```
Summary of Compile Points
Name Status Reason

mult Unchanged -
comb_logic Remapped Design changed
alu Unchanged -
eight_bit_uc Unchanged -
=====
```



#### Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
www.solvnet.com

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 13

# Using Multiprocessing

---

The following sections describe how to use multiprocessing to run parallel synthesis jobs and improve runtime:

- [Multiprocessing With Compile Points](#), on page 588
  - [Setting Maximum Parallel Jobs](#), on page 588
  - [License Utilization](#), on page 589

# Multiprocessing With Compile Points

Use the Configure Compile Point Process command to run multiprocessing with compile points. This option allows the synthesis software to run multiple, independent compile point jobs simultaneously, providing additional runtime improvements for the logical compile point synthesis flows. On the Configure Compile Point Process dialog box, specify the maximum number of synthesis jobs you can run in parallel. Note, one license is used for each job. For a description of how to set the maximum number of parallel synthesis jobs, see [Setting Maximum Parallel Jobs, on page 588](#).

To use multiprocessing in the Logical Compile Point Synthesis flows for the Synplify Pro and Synplify Premier tools, see [Chapter 12, Running Logical Compile Points](#). For the Synplify Premier tool, the Physical Synthesis switch must be turned off when you run compile points.

## Setting Maximum Parallel Jobs

You can set maximum number of parallel jobs in the following ways:

- [INI variable — MaxParallelJob](#)
- [Tcl variable — max\\_parallel\\_jobs](#)

### INI variable — MaxParallelJob

The maximum number of parallel jobs is set in the product .ini file. The following commands are set in the `<product>.ini` file (for example, `synplify_premier_dp.ini`):

```
[JobSetting]
MaxParallelJobs=<n>
```

The MaxParallelJobs value is used by the UI as well as in batch mode. This value is effective until you specify a new value. To change the number of parallel jobs you can run, use the Options->Configure Compile Point Process command from the Project view menu. On the Configure Compile Point Process dialog box, in the Maximum number of parallel synthesis jobs field you will see the current .ini value. You can specify a new MaxParallelJobs value which is effective until you change it again. Once you click OK, the new value is saved in the .ini file. For a description of the dialog box, see [Configure Compile Point Process, on page 244](#).



## Tcl variable — max\_parallel\_jobs

You can also manually set the maximum number of parallel jobs an override value. To do this, use the Tcl command:

```
set_option -max_parallel_jobs <n>
```

You can choose to:

- Source the Tcl file containing this option.
- Add this option to the Project file.
- Set this option from the Tcl command window.

This max\_parallel\_jobs value is applied to all project files and their respective implementations. This is a global option. The maximum number of parallel jobs remains in effect until you specify a new value. This new value takes affect immediately going forward. However, when you set this option from the Tcl command window, the max\_parallel\_jobs value is not saved and will be lost when you exit the application.

## License Utilization

When you decide to run parallel synthesis jobs, a license is used for each compile point job that runs. For example, if you set the Maximum number of parallel synthesis jobs to 4, then the synthesis tool consumes one license and three additional licenses are utilized to run the parallel jobs if they are available for your computing environment. Licenses are released as jobs complete, and then consumed by new jobs which need to run.

The actual number of licenses utilized depends on the:

1. Synthesis software scheme for the compile point requirements used to determine the maximum number of parallel jobs or licenses a particular design tries to use.
2. Value set on the Configure Compile Point Process dialog box.
3. Number of licenses actually available. You can use Help->Preferred License Selection to check the number of available license. If you need to increase the number of available licenses, you can specify multiple license types. For more information, see [Specifying License Types, on page 590](#).

Note that factors 1 and 3 above can change during a single synthesis run. The number of jobs equals the number of licenses; which then equates the lowest value of these three factors.

## Specifying License Types

You can specify multiple license types to increase the total number of licenses available for multiprocessing. To do this, you can either:

- Use the command line option `-licensetype` when you execute your tool.

For example, suppose you have two `synplifypremier` licenses, two `synplifypremier_allvendor` licenses, and three `synplifypremier_xilinx` licenses. Type the following at the command line:

```
synplify_premier.exe -licensetype
"synplifypremier:synplifypremier_allvendor:synplifypremier_xilinx"
```

- Use one of the following environment variables specified with the license type:
  - `SYNPLIFYPRO_LICENSE_TYPE` (Synplify Pro tool)
  - `SYNPLIFYPREMIER_LICENSE_TYPE` (Synplify Premier and Synplify Premier with Design Planner tools)

```
setenv SYNPLIFYPREMIER_LICENSE_TYPE=
"synplifypremier:synplifypremier_allvendor:synplifypremier_xilinx"
```

Multiprocessing can access any of these seven license types for additional licenses.



### Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA  
 Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 14

# Synthesizing and Analyzing the Log File

---

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 592
- [Checking Log Results](#), on page 596
- [Handling Messages](#), on page 602
- [Validating Logic Synthesis for Physical Synthesis](#), on page 609

## 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 592
- [Running Physical Synthesis](#), on page 592

### 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.

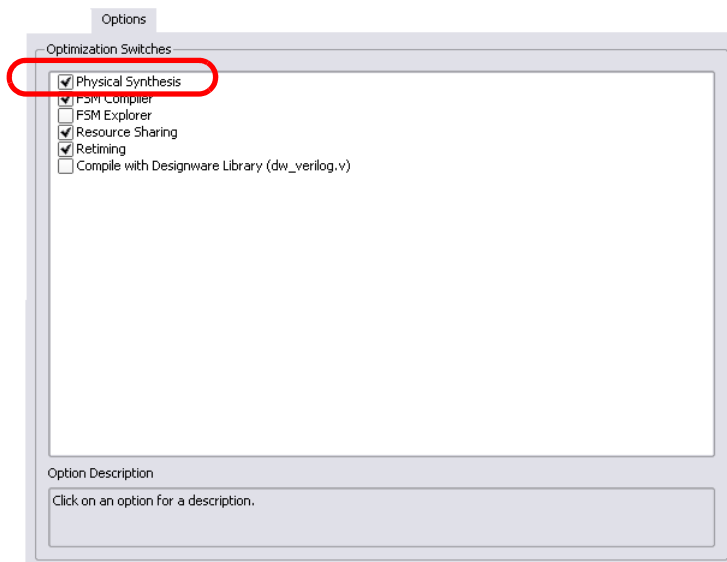
2. To synthesize the logic, set all the options and attributes you want, and then click Run.
3. To run logic synthesis as the initial phase of physical synthesis, see [Running Physical Synthesis](#), on page 592.

You can now run physical synthesis as described in [Running Physical Synthesis](#), on page 592.

### Running Physical Synthesis

When you run physical synthesis, the tool not only compiles the design and maps it to the technology target you selected, but also uses placement information to concurrently optimize and synthesize your design. Regardless of the flow you are using, run physical synthesis in two phases. First, run logic synthesis and fix any issues that come up. Then run physical synthesis.

1. Run logic synthesis as the initial phase of physical synthesis, by doing the following:
  - Set the options and attributes you want for physical synthesis, making sure to set up P&R to run automatically after synthesis.
  - Disable the Physical Synthesis switch either in the Project view or from the Implementation Options dialog box (Implementation Options->Options).



- Click Run to run logic synthesis.

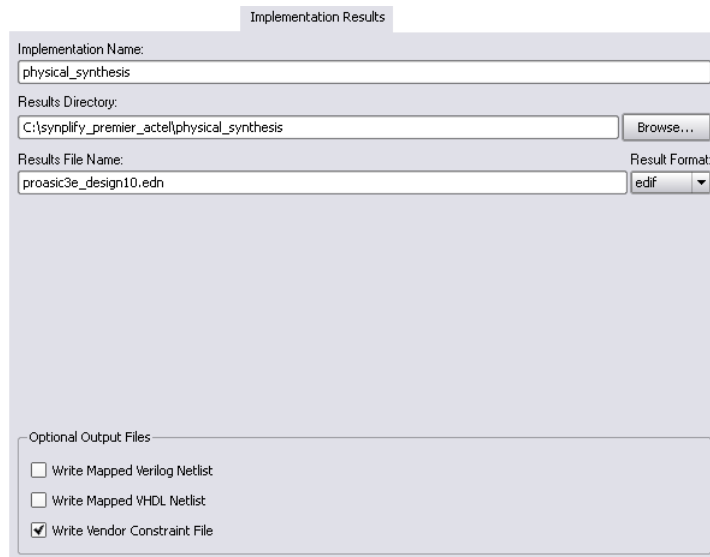
The Synplify Premier tool goes through *compiling* and *mapping* phases. When logical synthesis completes, Done! (or Warnings!) displays in the Project view. Output results files are shown in the right pane of the Project view.

2. Make adjustments to your design as needed.
  - Check the output files and analyze the results.
  - Fix any errors.

See [Validating Logic Synthesis for Physical Synthesis](#), on page 609 for details.

3. Set options for the physical synthesis run.
  - Set any other physical constraints.

- Enable the Physical Synthesis switch either in the Project view or from the Implementation Options dialog box (Implementation Options->Options).
- If you want a different directory for your physical synthesis results, click on the Implementation Options->Implementation Results tab and specify a new name for the implementation.



Implementation Results

Implementation Name:  
physical\_synthesis

Results Directory:  
C:\synplify\_premier\_actel\physical\_synthesis Browse...

Results File Name:  
proasic3e\_design10.edn

Result Format:  
edif

Optional Output Files

Write Mapped Verilog Netlist

Write Mapped VHDL Netlist

Write Vendor Constraint File

- Make sure the place-and-route implementation is enabled (Implementation Options->Place and Route tab).
- If you are using a Design Planner flow, click on the Design Planning tab and enable the desired design plan file (.sfp) if needed.

You do not need a design plan file to run graph-based physical synthesis. However, if you are using a graph-based flow and want to use a design plan file, use the procedure described in [Creating and Using a Design Plan File for Physical Synthesis](#), on page 494.

For older Altera technologies, you must create a design plan (.sfp) to run physical synthesis. See [Chapter 11, Floorplanning with Design Planner](#) for more information.



– Click OK in the Implementation Options dialog box .

#### 4. Run physical synthesis by clicking Run.

The tool performs optimizations using placement-aware synthesis. Synthesis and placement are integrated by performing concurrent placement and optimization based on timing constraints and device technology.

## Checking Log Results

You can check the log file for information about the synthesis run. In addition, the Synplify Pro and Synplify Premier interfaces have 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 the Log File](#), on page 596
- [Analyzing Results Using the Log File Reports](#), on page 599
- [Using the Log Watch Window](#), on page 600

### Viewing 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 Synplify Pro or Synplify Premier users who only want to check a few critical performance criteria, it is easier to use the Log Watch window (see [Using the Log Watch Window, on page 600](#)) instead of the log file. For details, read through the log file.

1. To view the log file, do one of the following:
  - To view the log file in the default HTML format, select View->Log File or click the View Log button in the Project window. You see the log file in HTML format. Alternatively you can double-click the *designName\_srr.htm* file in the Implementation Results view to open the HTML log file.
  - To see a text version of the log file, double-click the *designName.srr* file in the Implementation Results view. A Text Editor window opens with the log file.

Alternatively, you can set the default to show the text file version 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, 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 599](#).



The screenshot displays a log file viewer with two panes. The left pane shows a text-based log with the following content:

```

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
17 @I::"C:\Designs\ramCtrl\alu
18 @N: CG346 : "C:\Designs\ramCtrl
19 @I::"C:\Designs\ramCtrl\data
20 @I::"C:\Designs\ramCtrl\eight
21 @I::"C:\Designs\ramCtrl\ins
22 @I::"C:\Designs\ramCtrl\ins
23 @I::"C:\Designs\ramCtrl\io
24 @I::"C:\Designs\ramCtrl\prg
25 @I::"C:\Designs\ramCtrl\reg
26 @I::"C:\Designs\ramCtrl\spc
27 Verilog syntax check success
28 Selecting top level module
29 @N: CG364 : "C:\Designs\ramCtrl

```

The right pane shows an HTML-based log with the following content:

```

#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
Copyright (C) 1994-2007, Synplicity Inc. All Rights Reserve

@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
@N: CD233 : const_pkg.vhd(7) | Using sequential encoding for
@N: CD233 : const_pkg.vhd(6) | Using sequential encoding for
@N: CD233 : const_pkg.vhd(10) | Using sequential encoding for
@N: CD630 : ins_decode.vhd(7) | Synthesizing work.ins_decode
@N: CD233 : const_pkg.vhd(10) | Using sequential encoding for
@N: CD233 : const_pkg.vhd(7) | Using sequential encoding for

```

Annotations in the image include:

- A red arrow pointing to the text-based log on the left, labeled "Log File (Text)".
- A red arrow pointing to the HTML-based log on the right, labeled "Log File (HTML)".

## 2. To navigate in the log file, use the following techniques:

- Use the scroll bars.
- Use the Find command as described in the next step.
- In the HTML file, click the appropriate header to jump to that point in the log file. For example, you can jump to the Starting Points with Worst Slack section.

## 3. To find information in the log file, select Edit->Find or press Ctrl-f. Fill out the criteria in the form and click OK.

For general information about working in an Editing window, including adding bookmarks, see [Editing HDL Source Files with the Built-in Text Editor](#), on page 85.

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 Synplify Pro and Synplify Premier

products also let you generate a report for a path between any two designated points, see [Using the Stand-alone Timing Analyst, on page 736](#). 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                                     |
| Detailed information about slack times, constraints, arrival times, etc. | Interface Information                                   |
| Resource usage                                                           | Resource Usage Report                                   |
| Gated clock conversions                                                  | Gated clock report                                      |

#### 4. 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 602](#) for information. Notes are informational and usually can be ignored. For details about crossprobing and fixing errors, see [Handling Warnings, on page 609](#), [Editing HDL Source Files with the Built-in Text Editor, on page 85](#), and [Crossprobing from the Text Editor Window, on page 649](#).

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.

#### 5. 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 85](#).

6. 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.

## 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 by going to the Performance Summary section of the log file.
  - Check the slack times. See [Handling Negative Slack, on page 756](#) 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 Synplify Premier or Synplify Pro timing information about a path between any two designated points, see [Using the Stand-alone Timing Analyst, on page 736](#). For information about the Synplify Premier island-based timing report, see [Working in the Schematic Views, on page 614](#).

2. To check buffers,
  - 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,
  - Go to the Resource Usage Report section at the end of the log file.
  - Check the number and types of components used to determine if you have used too much of your resources.

## Using the Log Watch Window

The Synplify Pro and Synplify Premier Log 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 the Log File, on page 596](#).

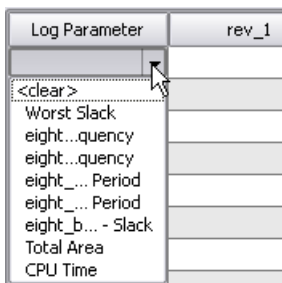
1. Open the Log Watch window, if needed, by checking View->Log Watch Window.

If you open an existing project, the Log 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 Log Watch window as described below.
  - Hold down Ctrl or Shift, click on the window, and move it to a position you want. This makes the Log 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 [Log Watch Window, on page 59](#) 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 **Log Watch** window now shows a column for each implementation you selected.
  - In the **Log 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 426.

| 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 |

## 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 602
- [Filtering Messages in the Message Viewer](#), on page 604
- [Filtering Messages from the Command Line](#), on page 606
- [Automating Message Filtering with a Tcl Script](#), on page 607
- [Handling Warnings](#), on page 609

### Checking Results in the Message Viewer

The Tcl Script window, a Synplify Pro and Synplify Premier feature, 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 63](#) in the *Reference Manual* for a full description of the window.

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

### 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.

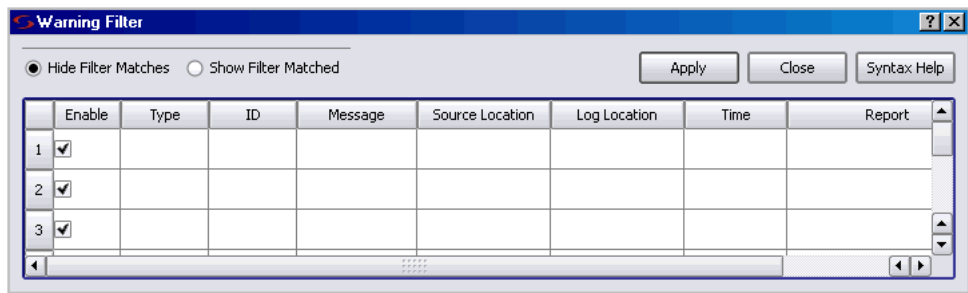
4. To filter the messages, use the procedure described in [Filtering Messages in the Message Viewer, on page 604](#). Crossprobe errors from the message window:
  - If you need more information about how to handle a particular message, click the message ID in the ID column. This opens the documentation for that message.
  - To open the corresponding source code file, click the link in the Source Location column. Correct any errors and rerun synthesis. For warnings, see [Handling Warnings, on page 609](#).
  - 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. It is not available with the Synplify tool. 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 602](#). For the command line equivalent of this procedure, see [Filtering Messages from the Command Line, on page 606](#).

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.

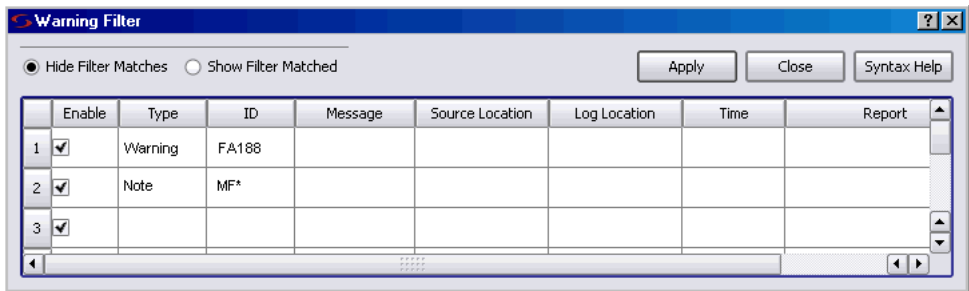


3. Set your display preferences.



- 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 604](#). Message filtering is not available with the Synplify tool.

1. Type your filter expressions in the Tcl window using the `log_filter` command. For details of the syntax, see [log\\_filter Tcl Command, on page 1273](#) in the *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) MO106 Found ROM, 'slaveControlSel_1', 15 words by 1 bits
Mapper Report wishbonebi.v (156) usbHostSlave.srr (820)
05:22:06 Mon Oct 18
@N MO106 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 MO106 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 604](#) or [Filtering Messages from the Command Line, on page 606](#) for details about creating this file.)

```
log_filter -clear
log_filter -hide_matches
log_filter -field report=="VIRTEX2P 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 884.
3. Edit the `synhooks.tcl` file so that it reads like the following example. For syntax details, see [Tcl `synhooks` File Syntax](#), on page 1271 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.

## Handling Warnings

If you get warnings (@W prefix) after a synthesis run, do the following:

- Read the warning message and decide if it is something you need to act on, or whether you can ignore it.
- If the message is not self-explanatory or if you are unsure about how to handle the error, click the message ID in either the message window or HTML log file or double click the message ID in the ASCII text log file. These actions take you to online information about the condition that generated the warning.

## Validating Logic Synthesis for Physical Synthesis

Use the following checklist to validate the results of logic synthesis in your physical design flows. These points apply to physical design flows for Actel, Altera, and Xilinx technologies.

1. Check that the logic synthesis run was successful. Check the following:
  - The Physical Synthesis switch was disabled.
  - Logic synthesis completed successfully.
  - Check the log file, as described in [Checking Log Results, on page 596](#).
2. Check that you used the correct version of the place-and-route tool. See the Release Notes, Help->Online Documents->release\_notes.pdf->*Third Party Tool Versions* for information.
3. Check for black boxes. Search the synthesis .srr log file for black box.

A design that contains black boxes errors out in the tool and should be eliminated from the design.

4. Check for combinational feedback loops. Search the synthesis `.srr` log file for Found combinational loop.

Combinational loops cause random timing analysis results that invalidate any comparison and should be eliminated from the design.

5. Make sure the clock constraints are correct. Check the Clock Relationships table in the `.srr` log file.
6. Check that the forward annotated timing constraints are consistent with the post place-and-route timing constraints.

|                                |                             |
|--------------------------------|-----------------------------|
| Actel forward annotation file  | <code>.tcl</code>           |
| Altera forward annotation file | <code>.tcl</code>           |
| Xilinx forward annotation file | <code>synplicity.ucf</code> |

7. Are the false and multi-cycle paths constraints correctly defined in the `.sdc` file? Ensure that the back-annotation timing report (`.srr` log file in the PAR directory) matches the report file.

The file varies, depending on the vendor:

|                    |                       |
|--------------------|-----------------------|
| Actel report file  | <code>.twr</code>     |
| Altera report file | <code>.tan.rpt</code> |
| Xilinx report file | <code>.twr</code>     |

For Altera and Xilinx designs, there are a couple of additional points to check:

1. Check that the clocks are routed on global resources.
  - Check the Clock Path Skew numbers in the report file.  
Clocks routed on general routing resources usually result in large skews. Because the tool does not take clock skew into account, large skews can degrade the quality of results (QoR) and result in poor timing correlation. The name of the report file varies, depending on the vendor:

|                    |          |
|--------------------|----------|
| Altera report file | .tan.rpt |
| Xilinx report file | .twr     |

For Virtex5 designs, see [Working with Clock Skews in Xilinx Virtex-5 Physical Designs](#), on page 824.

2. For Xilinx designs, check that the DCM parameters correctly defined in the source code or .sdc constraint file. Check the Clock Relationships table in the .srr log file.



**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.



## CHAPTER 15

# Analyzing with HDL Analyst and FSM Viewer

---

This chapter describes how to analyze logic in the HDL Analyst and FSM Viewer. These tools are only available in the Synplify Pro and Synplify Premier products, though you can purchase HDL Analyst as an option to the base Synplify product.

See the following for detailed procedures:

- [Working in the Schematic Views](#), on page 614
- [Exploring Design Hierarchy](#), on page 627
- [Finding Objects](#), on page 635
- [Crossprobing](#), on page 645
- [Analyzing With the HDL Analyst Tool](#), on page 654
- [Using the FSM Viewer](#), on page 670

For information about analyzing timing, see [Chapter 17, Analyzing Timing](#).  
For information about using Synplify Premier Physical Analyst tool, see [Chapter 16, Analyzing Designs in Physical Analyst](#).

## Working in the Schematic Views

The HDL Analyst includes the RTL and Technology views, which are schematic views used to graphically analyze your design. In the Synplify product, these views are part of the optional HDL Analyst package. 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. In the Synplify Premier product, a RTL Floorplan view is available after a floorplan has been created with physical constraint regions and synthesized for the device.

For detailed descriptions of these views, see Chapter 2 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 Views](#), on page 615
- [Opening the Views](#), on page 615
- [Viewing Object Properties](#), on page 617
- [Viewing Object Properties](#), on page 617
- [Selecting Objects in the RTL/Technology Views](#), on page 620
- [Working with Multisheet Schematics](#), on page 621
- [Moving Between Views in a Schematic Window](#), on page 623
- [Setting Schematic View Preferences](#), on page 623
- [Managing Windows](#), on page 625

For information on specific tasks like analyzing critical paths, see the following sections:



- [Exploring Object Hierarchy by Pushing/Popping](#), on page 628
- [Exploring Object Hierarchy of Transparent Instances](#), on page 634
- [Browsing to Find Objects](#), on page 635
- [Crossprobing](#), on page 645
- [Analyzing With the HDL Analyst Tool](#), on page 654

## Differentiating Between the Views

- The difference between the RTL and Technology views is that the RTL view is the view generated after compilation, while the Technology view is the view generated after mapping. The RTL view displays your design as a high-level, technology-independent schematic. At this high level of abstraction, the design is represented with technology-independent components like variable-width adders, registers, large muxes, state machines, and so on. This view corresponds to the `.srs` netlist file generated by the software in the Synopsys proprietary format. For a detailed description, see Chapter 2 of the *Reference Manual*.
- The Technology view contains technology-specific primitives. It shows low-level, vendor-specific components such as look-up tables, cascade and carry chains, muxes, and flip-flops, which can vary with the vendor and the technology. This view corresponds to the `.srm` netlist file, generated by the software in the Synopsys proprietary format. For a detailed description, see Chapter 2 of the *Reference Manual*.
- The Synplify Premier RTL Floorplan view displays a floorplan schematic that includes all the logic assigned to any physical constraint regions created on the device, as well as, all other logic of the design. This view uses the same high-level abstraction and technology-independent components of the RTL view.

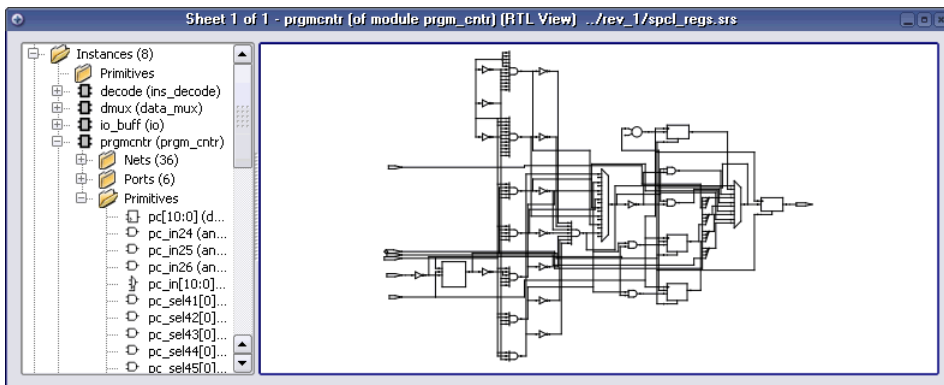
## Opening the Views

The procedure for opening an RTL or Technology view is similar; the main difference is the design stage at which these views are available.

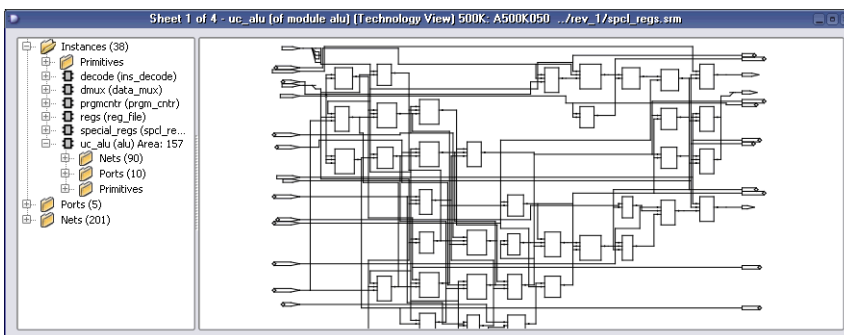
|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| To open an RTL view...       | <p>Start with a compiled design.</p> <p>To open a hierarchical RTL view, do one of the following:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst-&gt;RTL-&gt;Hierarchical View.</li> <li>• Click the RTL View icon (). (a plus sign inside a circle).</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>                                  |
| To open a Technology view... | <p>Start with a mapped (synthesized) design.</p> <p>To open a hierarchical Technology view, do one of the following:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst -&gt;Technology-&gt;Hierarchical View.</li> <li>• Click the Technology View icon (NAND gate icon ).</li> <li>• Double-click the .srm file in the Implementation Results view.</li> </ul> <p>To open a flattened Technology view, select HDL Analyst-&gt;Technology-&gt;Flattened View.</p> |
| To open a Floorplan view     | <p>Start with a synthesized design that has been floorplanned with physical constraint regions.</p> <p>To open a RTL Floorplan view:</p> <ul style="list-style-type: none"> <li>• Select HDL Analyst-&gt;RTL-&gt;Floorplanned View.</li> <li>• Double-click the partitioned netlist (.srp) file from the Implementation Results view.</li> </ul>                                                                                                                                                                                                                  |

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 the window contains the name of the view, the kind of view, hierarchical level, and the number of sheets in the schematic. See [Hierarchy Browser, on page 71](#) in the *Reference Manual* for a description of the Hierarchy Browser.

## RTL View



## Technology View

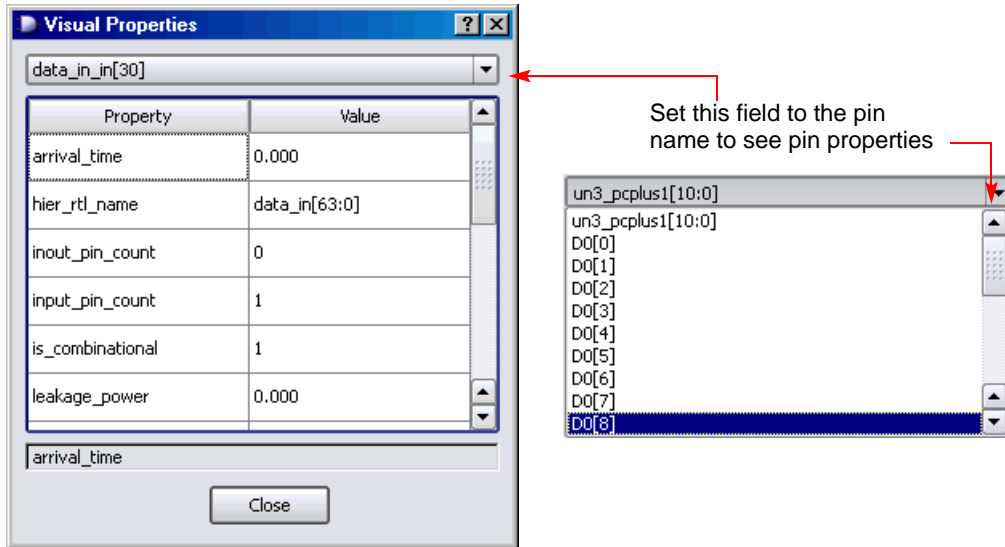


## Viewing Object Properties

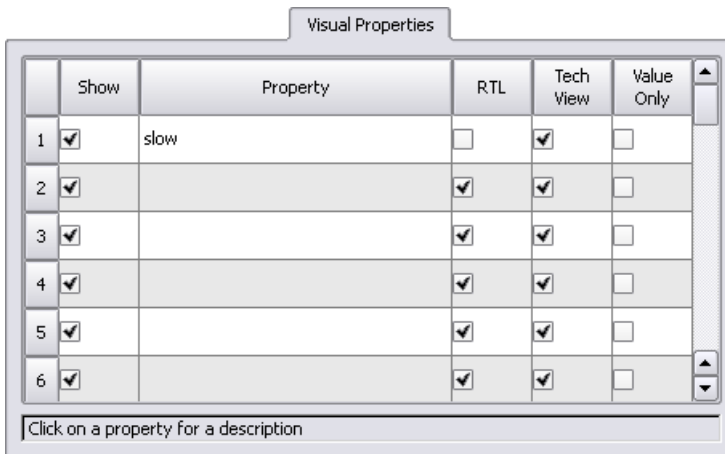
There are a few ways in which you can view the properties of objects.

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.



- To flag objects by property, do the following with an open RTL/Technology view:
  - Set the properties you want to see by selecting Options->HDL Analyst Options->Visual Properties, and selecting the properties from the pull-down list. Some properties are only available in certain views.

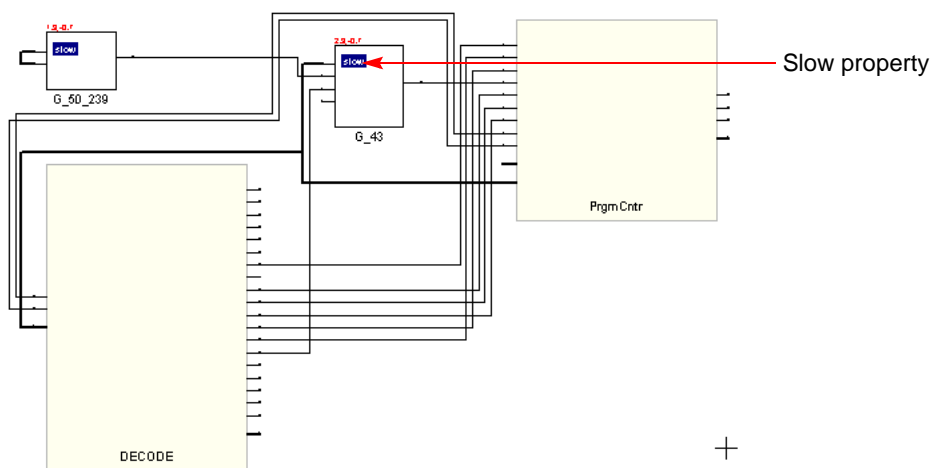


- Close the HDL Analyst Options dialog box.

- Enable View->Visual Properties. If you do not enable this, the software does not display the property flags in the schematics. The HDL Analyst annotates all objects in the current view that have the specified property with a rectangular flag that contains the property name and value. The software uses different colors for different properties, so you can enable and view many properties at the same time.

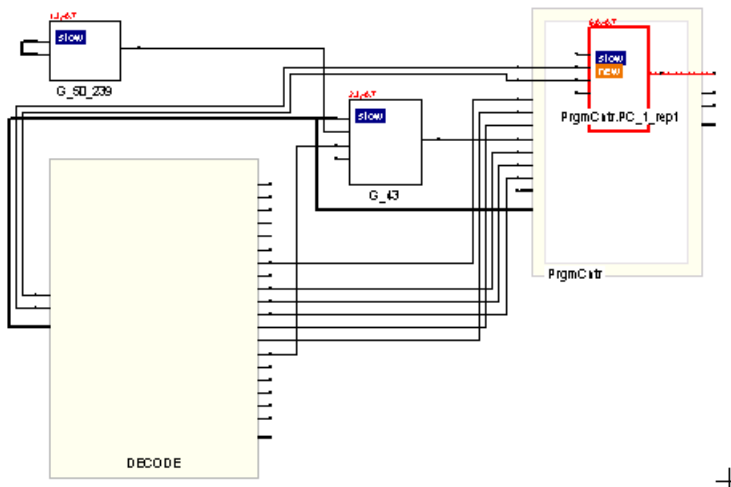
### Example: Slow and New Properties

You can view objects with the slow property when you are analyzing your critical path. All objects with this property do not meet the timing criteria. The following figure shows a filtered view of a critical path, with slow instances flagged in blue.



When you are working with filtered views, you can use the New property to quickly identify 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. This can be useful when you are debugging your design.

The following figure expands one of the pins from the previous filtered view. The new instance added to the view has two flags: new and slow.



## 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                         | Use one of these methods: <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 635</a>.</li> <li>• Use Find to select the objects you want. See <a href="#">Using Find for Hierarchical and Restricted Searches, on page 637</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 636</a> ), or use the Hierarchy Browser, which lists objects by type.                                                                                                                                                                                                                                                                                                                  |



| To select...                                           | Do this...                                                                                                                                                                                                                                                                               |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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, on page 645](#) 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

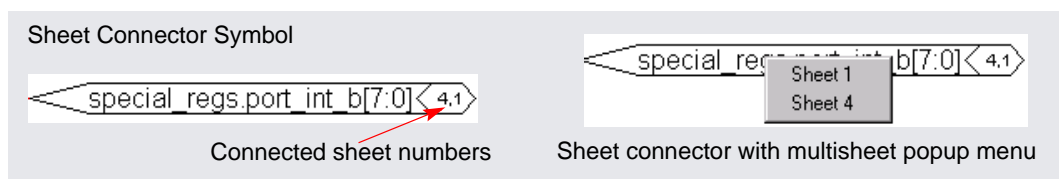
The title bar of the RTL or Technology view indicates the number of sheets in that schematic. In a multisheet schematic, nets that span multiple sheets are indicated by sheet connector symbols, which you can use for navigation.

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                                   | <p>Select View-&gt;Next/Previous Sheet.</p> <p>Press the right mouse button and draw a horizontal mouse stroke (left to right for next sheet, right to left for previous sheet).</p> <p>Click the icons: Next Sheet (👉) or Previous Sheet (👈)</p> <p>Press Shift-right arrow (Next Sheet) or Shift-left arrow (Previous sheet).</p> <p>Navigate with View-&gt;Back and View -&gt;Forward if the next/previous sheets are part of the display history.</p> |
| A specific sheet number                                        | <p>Select View-&gt;View Sheets and select the sheet.</p> <p>Click the right mouse button, select View Sheets from the popup menu, and then select the sheet you want.</p> <p>Press Ctrl-g and select the sheet you want.</p>                                                                                                                                                                                                                              |
| Lower-level logic of a transparent instance on separate sheets | <p>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.</p>                                                                                                                                                                                                                                                                           |
| All objects of a certain type                                  | <p>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.</p> <p>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.</p>                                                                                                                           |
| Selected items only                                            | <p>Filter the schematic as described in <a href="#">Filtering Schematics, on page 658</a>.</p>                                                                                                                                                                                                                                                                                                                                                            |
| A net across sheets                                            | <p>If there are no sheet numbers displayed in a hexagon at the end of the sheet connector, select Options -&gt;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.</p>                                                                                                                                                                |

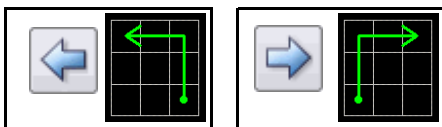


## Moving Between Views in a Schematic Window

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 View Preferences

You can set various preferences for the RTL and Technology views from the user interface.

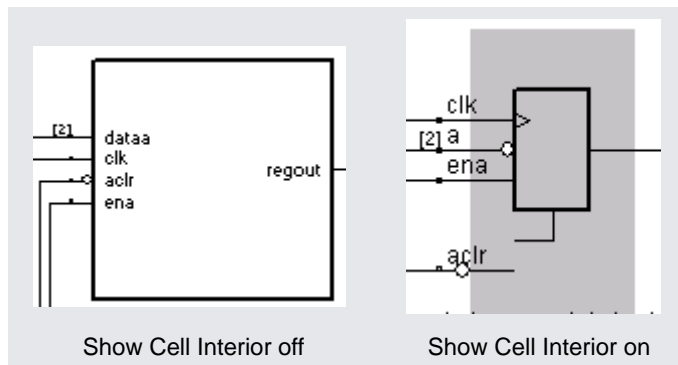
1. Select Options->HDL Analyst Options. For a description of all the options on this form, see [HDL Analyst Options Command](#), on page 255 in the *Reference Manual*.
2. The following table details some common operations:

| 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) |

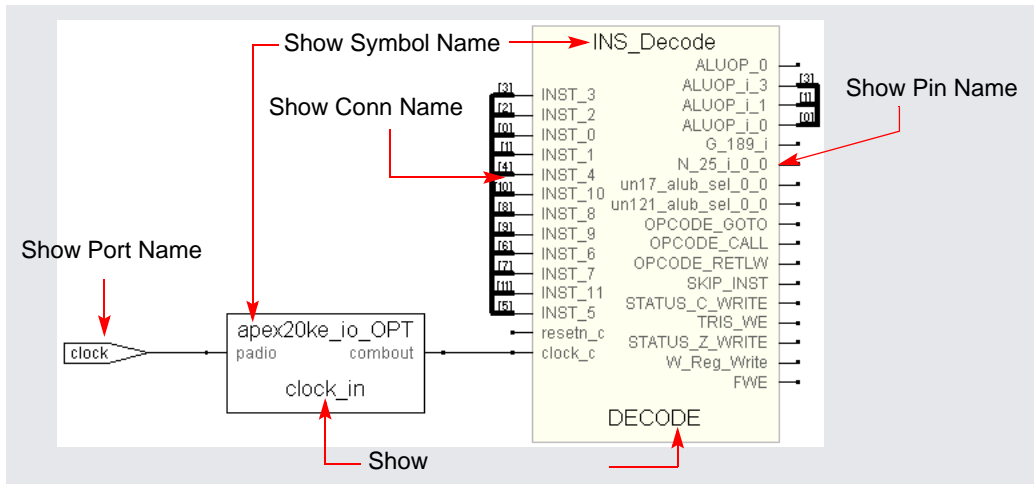
| To...                                                                    | Do this...                                                                                                                                                                                               |
|--------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 view you open.

- 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 351](#) 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 623](#).

- 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

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 HDL Analyst Tool, on page 654](#). The topics include:

- [Traversing Design Hierarchy with the Hierarchy Browser](#), on page 627
- [Exploring Object Hierarchy by Pushing/Popping](#), on page 628
- [Exploring Object Hierarchy of Transparent Instances](#), on page 634

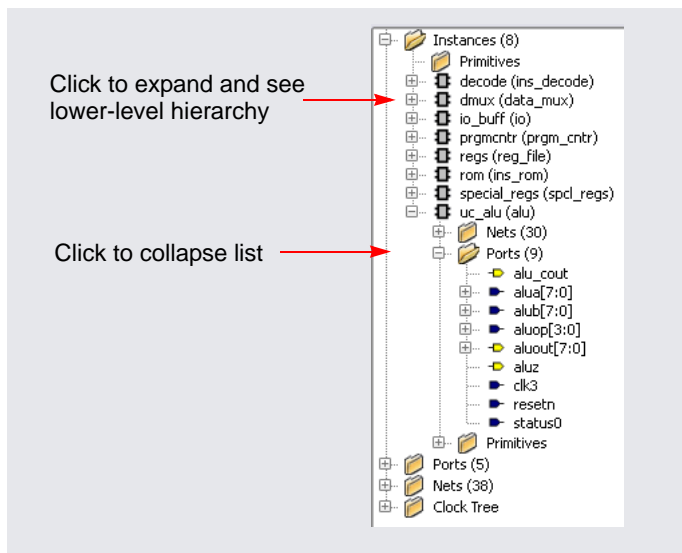
### Traversing Design Hierarchy with the Hierarchy Browser

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 628](#) 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 on the plus sign for the object. To ascend the hierarchy, click on the minus sign.



Refer to [Hierarchy Browser Symbols](#), on page 72 in the *Reference Manual* for an explanation of the symbols.

## Exploring Object Hierarchy by Pushing/Poping

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 627. 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 658, [Expanding Pin and Net Logic](#), on page 660, and [Expanding and Viewing Connections](#), on page 664 for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

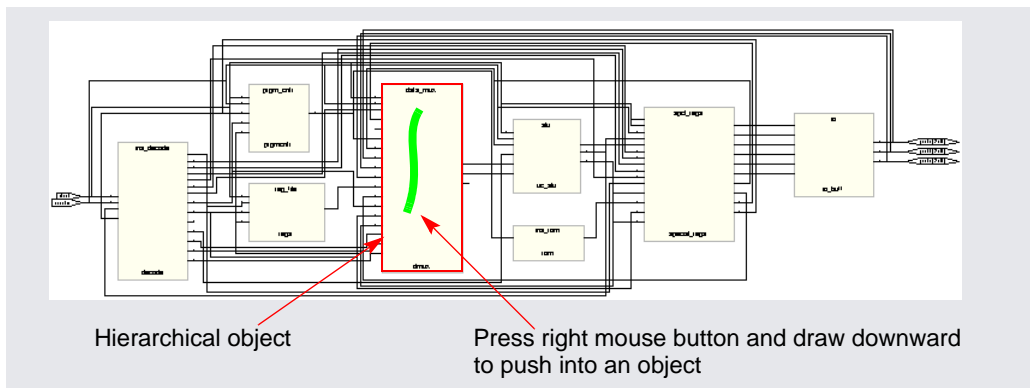
- [Pushing into Objects](#), on page 629, next
- [Popping up a Hierarchical Level](#), on page 632



## Pushing into Objects


In the schematic views, 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).

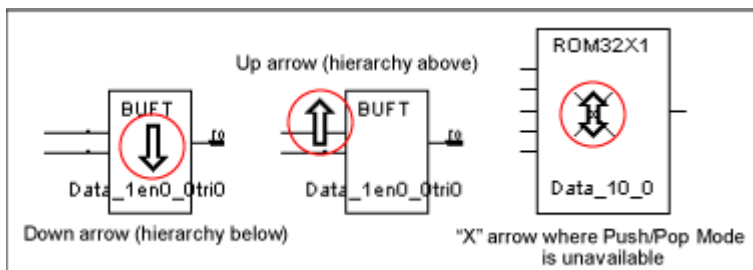


- Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the Technology view, under Instances - Primitives.
- Inferred ROMs and state machines.

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.

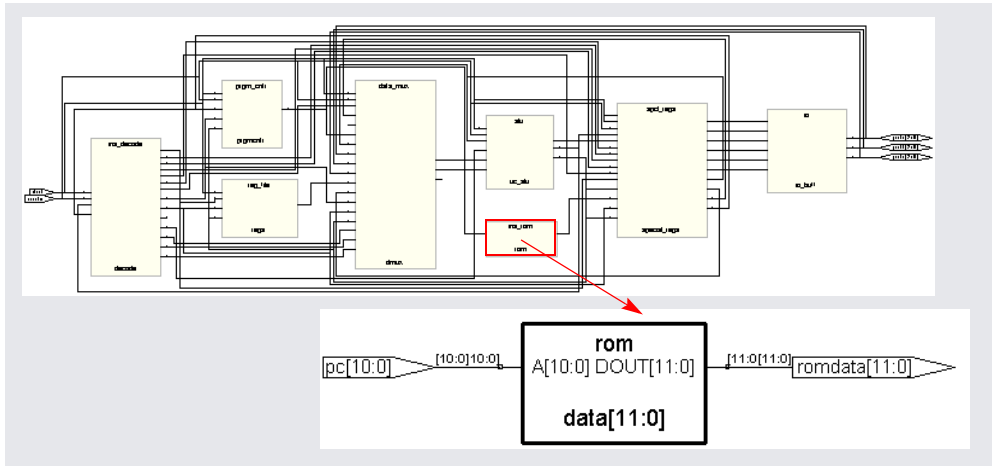
The cursor changes to an arrow. The direction of the arrow indicates the underlying hierarchy, as shown in the following figure. The status bar at the bottom of the window reports information about the objects over which you move your cursor.



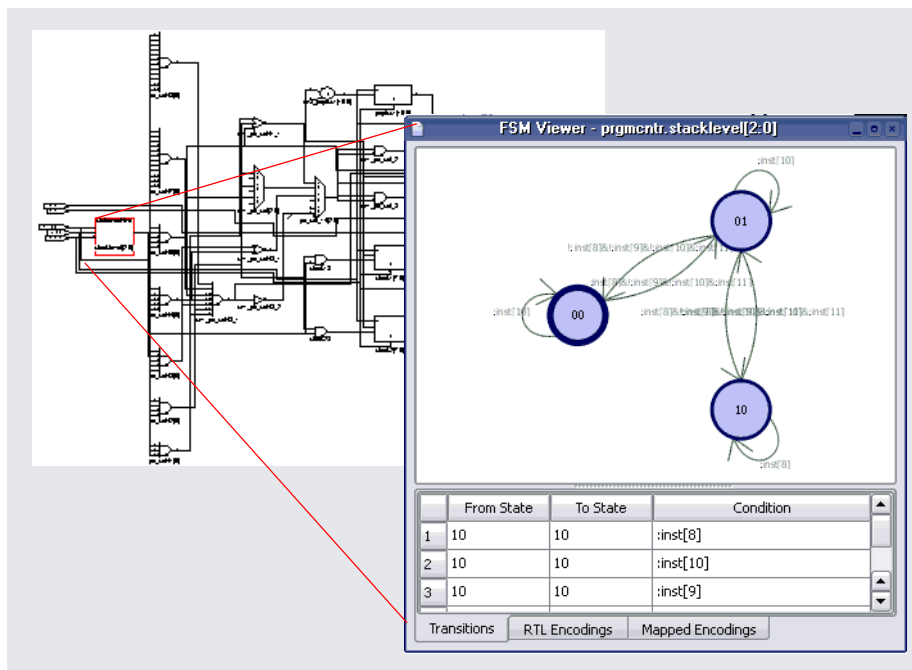
3. To push (descend) into an object, click on the hierarchical object. For a transparent instance, you must click on the pale yellow border. The following figure shows the result of pushing into a ROM.

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. (Synplify users cannot

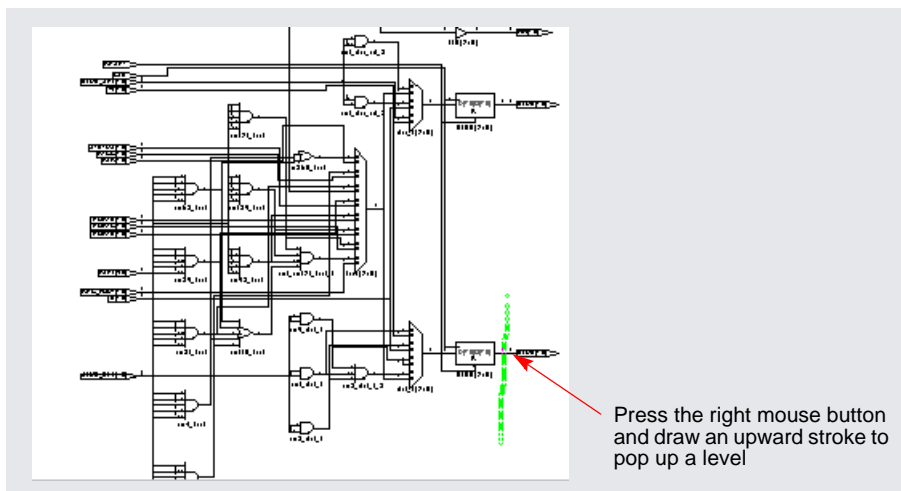


push into state machines.) When you push into an FSM from the RTL view, you open the FSM viewer where you can graphically view the transitions. For more information, see [Using the FSM Viewer, on page 670](#). If you push into a state machine from the Technology view, you see the underlying logic.



## 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 629](#) 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 628](#)).

|                | <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

In the schematic views, you can use the Hierarchy Browser or the Find command to find objects, as explained in these sections:

- [Browsing to Find Objects](#), on page 635
- [Using Find for Hierarchical and Restricted Searches](#), on page 637
- [Using Wildcards with the Find Command](#), on page 640
- [Using Find to Search the Output Netlist](#), on page 643

For information about the Tcl Find command, which you use to locate objects, and create collections, see [Tcl expand Command, on page 1257](#) in the *Reference Manual*.

## Browsing to Find Objects

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.

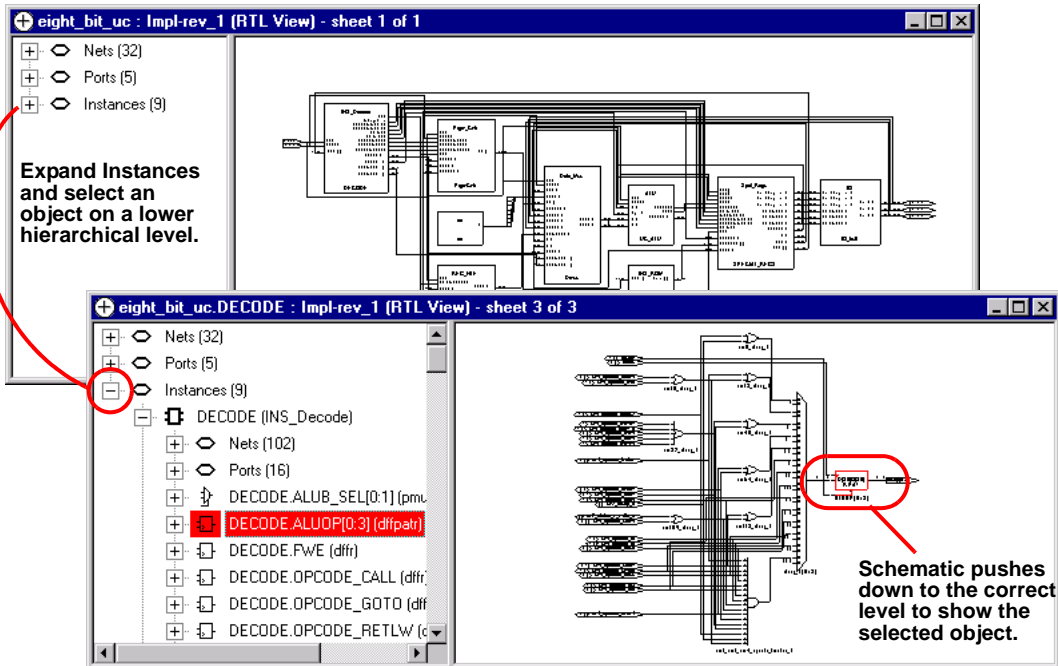
2. To select a range of objects, select the first object in the range. Then, scroll to display the last object in the range. Press and hold the Shift key while clicking the last object in the range.

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 view, 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.

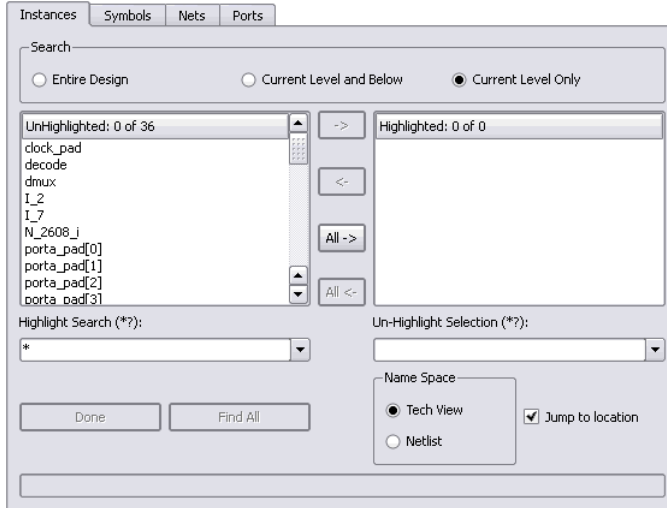
## Using Find for Hierarchical and Restricted Searches

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, on page 635](#)). 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.

1. If needed, restrict the range of the search by filtering the view, hiding instances, or both. See [Viewing Design Hierarchy and Context, on page 655](#) and [Filtering Schematics, on page 658](#) 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. 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.

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 open the Object Query dialog box, 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.



3. 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.

4. 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 Search Range on Wildcard Searches, on page 640](#) for details. Current Level Only or Current Level & Below are useful for searching filtered schematics or critical path schematics.

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.

5. 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 643](#).

6. 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.

7. 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 640](#) 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* |

- 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 124 of the *Reference Manual*.

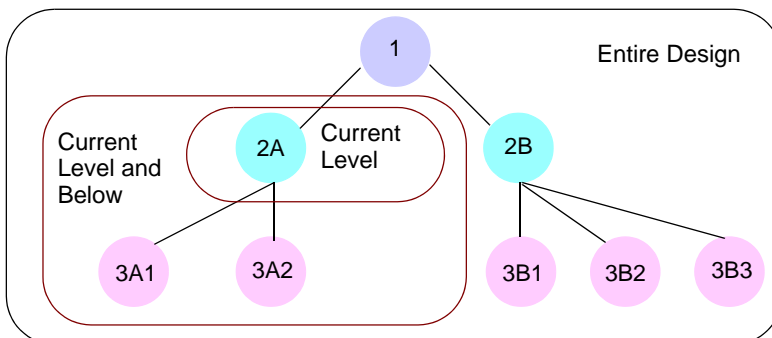
## Using Wildcards with the Find Command

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 Search Range on Wildcard Searches

The asterisk and question mark do not cross hierarchical boundaries. However, the scope of the search determines the starting points for the searches, and this might make it appear as if the wildcards cross hierarchical boundaries in some cases. 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 642 for examples.

## How a Wildcard Search Works

1. 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. |
|---------------|---------------------------------------------------------------------------------------------------------------|

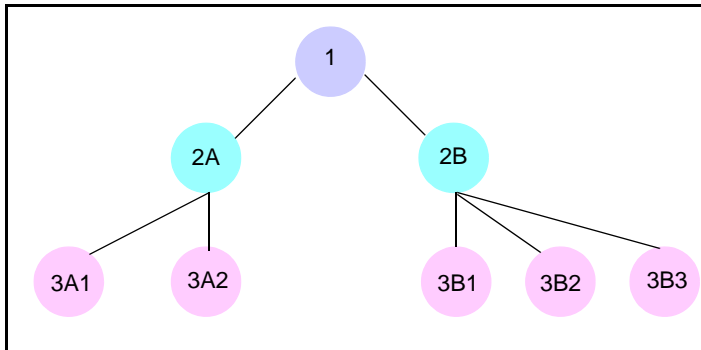
|                         |                                                                                                                                                                                                 |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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. |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

2. 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 (like \*.\* ) are repeated at each level included in the scope. See [Effect of Search Range on Wildcard Searches](#), on page 640 and [Wildcard Search Examples](#), on page 642 for details and examples, respectively. 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.



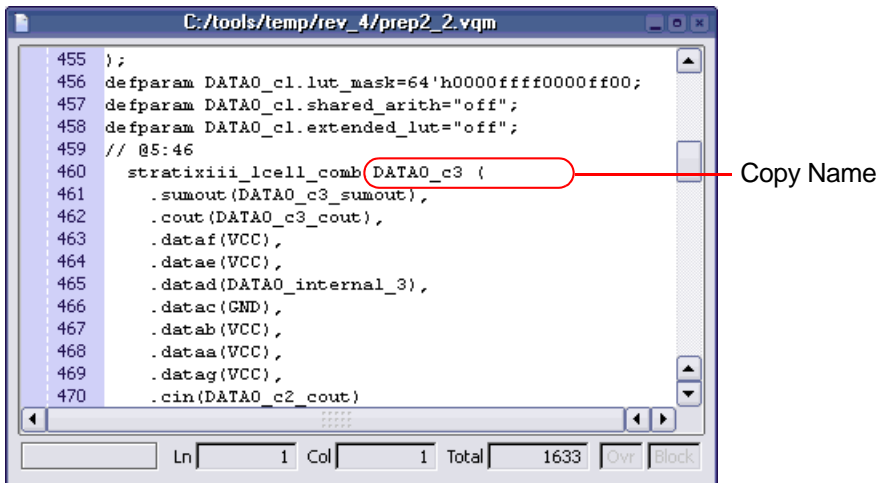
| Scope         | Pattern | Starting Point | Finds Matches in...                                                                                                                                                             |
|---------------|---------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Entire Design | *       | 3A1            | 1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3 (* at all levels)                                                                                                                        |
|               | *.*     | 2B             | 2A and 2B (*.* from 1)<br>3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B)<br>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 <b>Current Level</b> . 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)<br>3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B)<br>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)<br>Search ends because there is no hierarchy two levels below 2A and 2B.            |

## 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.

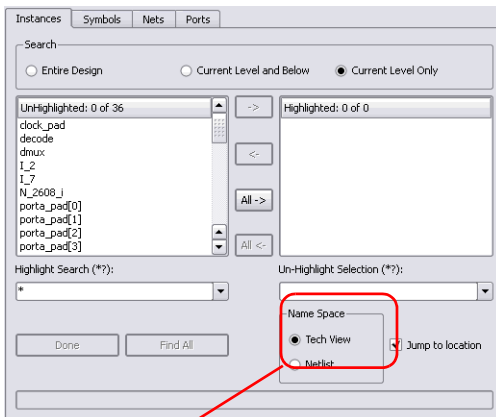


```

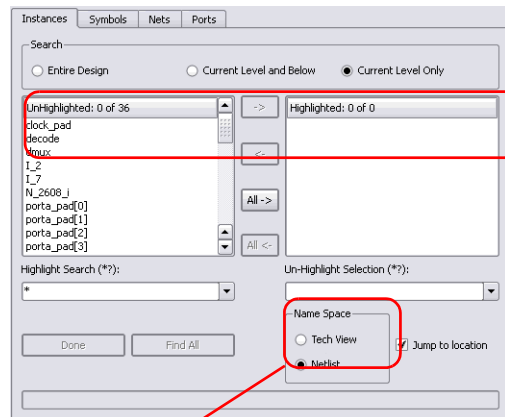
455);
456 defparam DATA0_c1.lut_mask=64'h0000ffff0000ff00;
457 defparam DATA0_c1.shared_arith="off";
458 defparam DATA0_c1.extended_lut="off";
459 // @5:46
460 stratixiii_lcell_comb DATA0_c3 (
461 .sumout(DATA0_c3_sumout),
462 .cout(DATA0_c3_cout),
463 .dataf(VCC),
464 .datae(VCC),
465 .datad(DATA0_internal_3),
466 .datac(GND),
467 .datab(VCC),
468 .dataa(VCC),
469 .datag(VCC),
470 .cin(DATA0_c2_cout)

```

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.



Search by Tech View



Search by Netlist

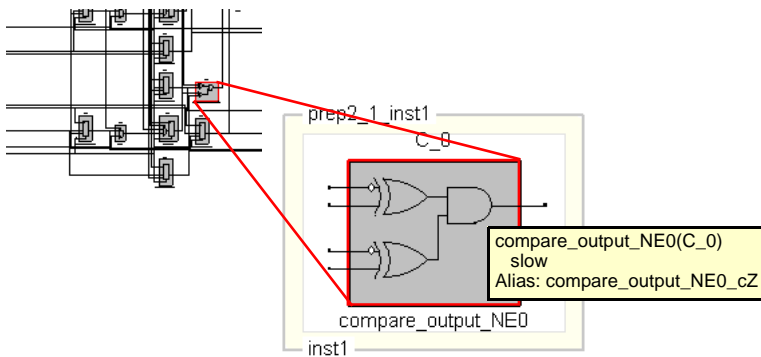


If you leave the Name Space option set to the default of Tech View, the tool does not find the name because it is searching the mapped database instead of the output netlist.

- 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 on the filtered schematic to crossprobe to the corresponding code in the HDL file.

## 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. Highlighting a line of text, for example, highlights the corresponding logic in the schematic views. 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 (not available in the Synplify product), 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:

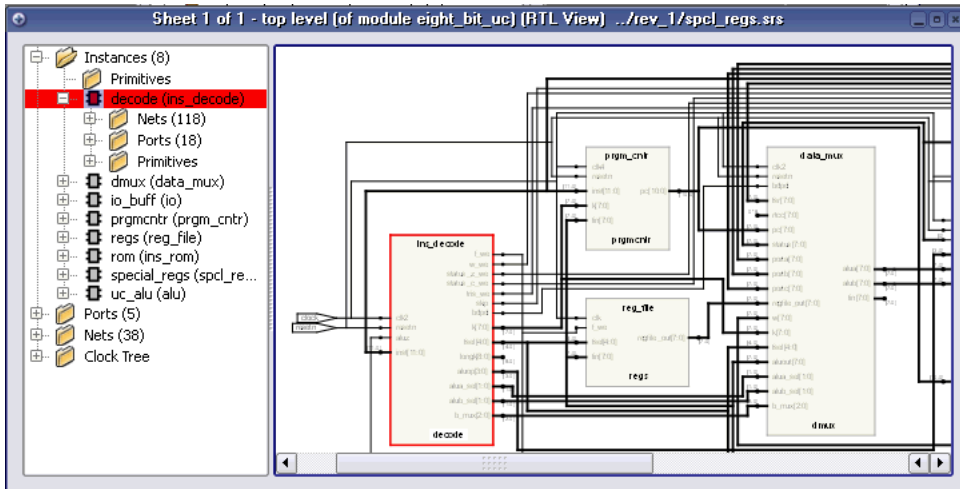
- [Crossprobing within an RTL/Technology View](#), on page 646
- [Crossprobing from the RTL/Technology View](#), on page 647
- [Crossprobing from the Text Editor Window](#), on page 649
- [Crossprobing from the Tcl Script Window](#), on page 652
- [Crossprobing from the FSM Viewer](#), on page 652

## 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.

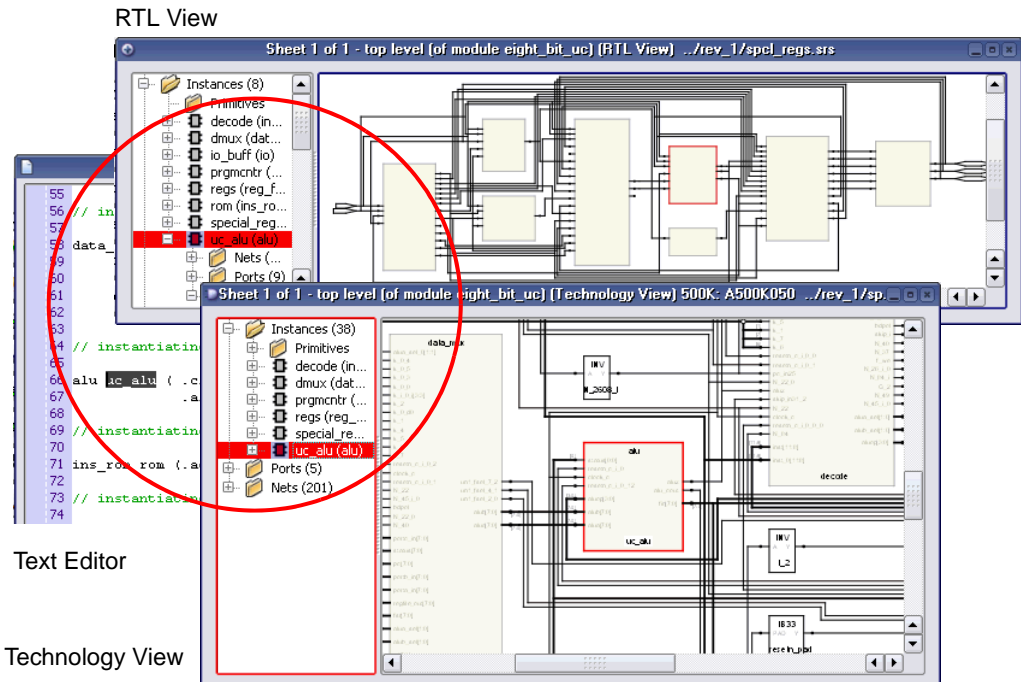


## Crossprobing from the RTL/Technology View

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.



- 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.

| From       | To                           | Procedure                                                                                                                                                                                                                                                 |
|------------|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 (not in Synplify) | 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.<br><br>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 650](#) for a practical example of how to use crossprobing.

1. Open the RTL, FSM, or Technology view to which you want to crossprobe. The FSM view is not a Synplify feature.
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 view (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 Solaris and Linux platforms, use the key to which the Alt function is mapped; this is usually the Meta or Diamond key for Solaris or the Ctrl-Alt key combination for Linux.
  - 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

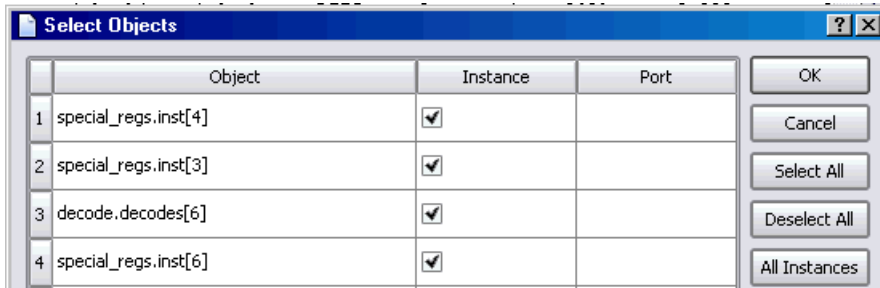
The screenshot displays the HDL Analyst interface. On the left, a 'Text Editor' window shows a 'Clock' table with the following data:

| Clock        |        |      |   |  |
|--------------|--------|------|---|--|
| dmux.alua[0] | System | DFFC | D |  |
| dmux.alua[1] | System | DFFC | D |  |
| dmux.alua[2] | System | DFFC | D |  |
| dmux.alua[3] | System | DFFC | D |  |
| dmux.alua[4] | System | DFFC | D |  |
| dmux.alua[5] | System | DFFC | D |  |
| dmux.alua[6] | System | DFFC | D |  |
| dmux.alua[7] | Sy     |      |   |  |
| dmux.alub[0] | Sy     |      |   |  |
| dmux.alub[1] | Sy     |      |   |  |

Below the table, the 'Log File Link' section shows 'rev\_1' and 'Worst Path Information'.

On the right, the 'Technology View' window shows a complex circuit diagram for 'Sheet 3 of 4 - dmux [of module data\_mux] [Technology View] 500...'. The diagram features a hierarchical tree on the left with categories like 'Instan...', 'Ports (5)', and 'Nets (...)'. The main area displays a dense network of logic blocks and interconnecting lines, with several components highlighted in red.

- 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.



- 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. You cannot do this with the Synplify product, because it does not have the Tcl window feature.

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. The Synplify tool does not support the FSM viewer.

To crossprobe from the FSM Viewer, do the following:

- Open the view to which you want to crossprobe: RTL/Technology view, or the source code file.
- 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 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 Extraction 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. Note that Synplify users do not have access to the FSM viewer. See the following for more information about analysis techniques.

- [Viewing Design Hierarchy and Context](#), on page 655
- [Filtering Schematics](#), on page 658
- [Expanding Pin and Net Logic](#), on page 660
- [Expanding and Viewing Connections](#), on page 664
- [Flattening Schematic Hierarchy](#), on page 665
- [Minimizing Memory Usage While Analyzing Designs](#), on page 670

For additional information about navigating the HDL Analyst views or using other techniques like crossprobing, see the following:

- [Working in the Schematic Views](#), on page 614
- [Exploring Design Hierarchy](#), on page 627
- [Finding Objects](#), on page 635
- [Crossprobing](#), on page 645

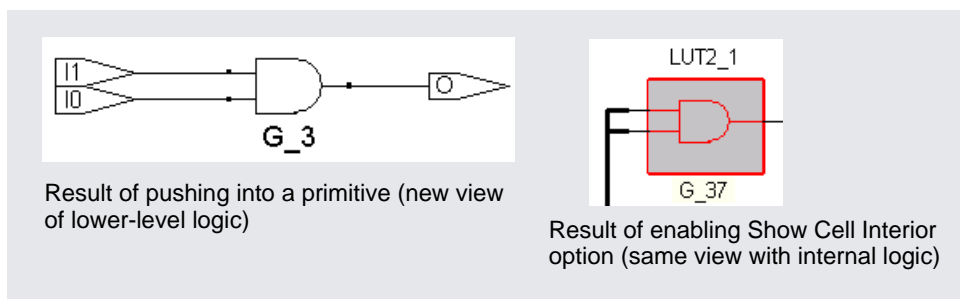
## Viewing Design Hierarchy and Context

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 733](#) 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.

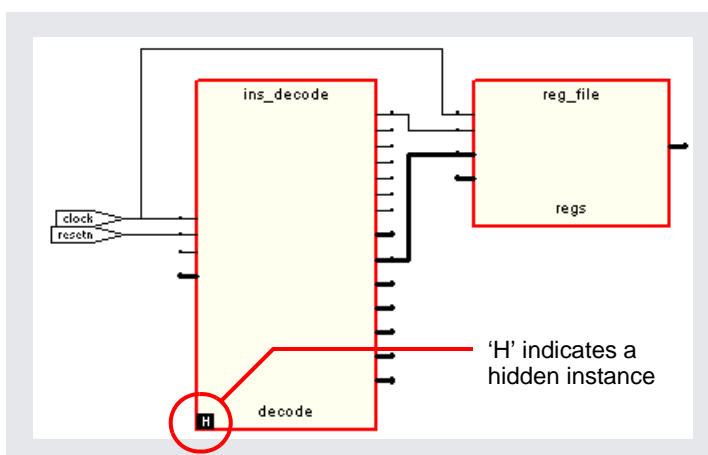
1. To view the internal logic of primitives in your design, do either of the following:
  - To view the logic of an individual primitive, push into it. This generates a new schematic view with the internal details. Click the Back icon to return to the previous view.
  - To view the logic of all primitives 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 view and all subsequent views. If the view is too cluttered with this option on, filter the view (see [Filtering Schematics, on page 658](#)) 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 view.

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 <b>Back</b> . See <a href="#">Exploring Object Hierarchy by Pushing/Popping</a> , on page 628 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</a> , on page 665 for details about flattening designs.<br>Because this is a new view, you cannot use <b>Back</b> to return to the previous view. To return to the top-level unflattened schematic, right-click in the view and select <b>Unflatten Schematic</b> . |
| 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</a> , on page 665 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

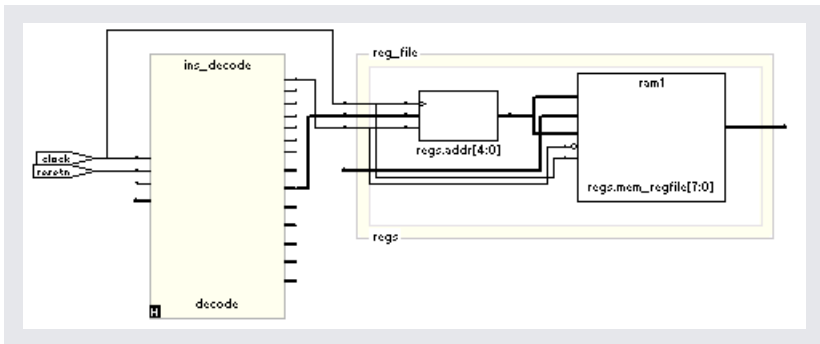
Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter Schematic command to isolate selected 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.

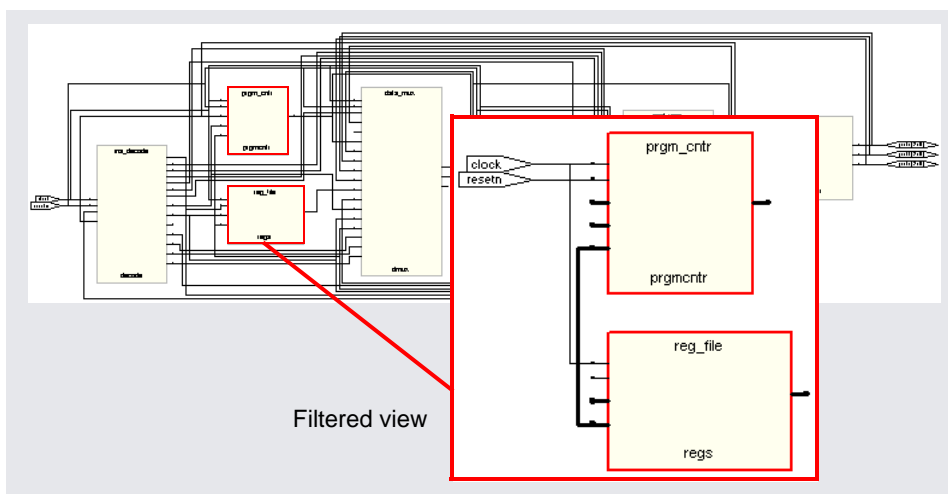
If you filter a hidden instance, the software does not display its internal hierarchy when you filter the design. The following example illustrates this.



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 344 and *Transparent and Opaque Display of Hierarchical Instances*, on page 349 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 655, 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</a> , on page 660 and <a href="#">Expanding and Viewing Connections</a> , on page 664 |
| Filter further                | Select objects and filter again                                                                                                   |
| Find objects                  | See <a href="#">Finding Objects</a> , on page 635                                                                                 |
| Flatten, or hide and flatten  | See <a href="#">Flattening Schematic Hierarchy</a> , on page 665. You can hide transparent or opaque instances.                   |
| Crossprobe from filtered view | See <a href="#">Crossprobing from the RTL/Technology View</a> , on page 647                                                       |

4. To return to the previous schematic view, 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 658 and [Flattening Schematic Hierarchy](#), on page 665.

## 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 664.

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 658 and [Flattening Schematic Hierarchy](#), on page 665 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</a> , on page 662.                                                                                          |
| 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</a> , on page 663.                                                        |
| 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</a> , on page 663. |

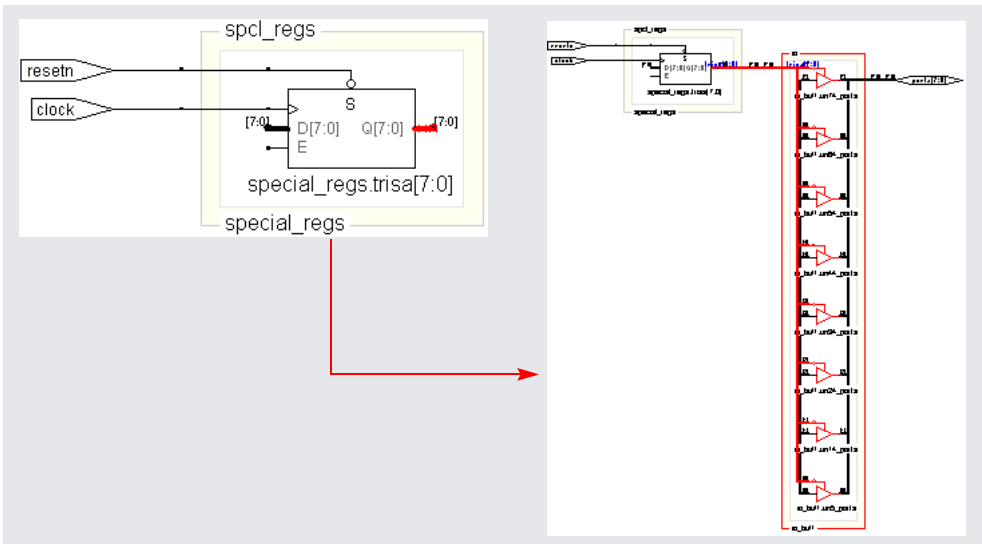
The software expands the logic as specified, working on the current level and below or working up the hierarchy, crossing hierarchical boundaries as needed. Hierarchical levels are shown nested in hollow bounding boxes. The internal hierarchy of hidden instances is not displayed.

For descriptions of the Expand commands, see [HDL Analyst Menu](#), on page 229 of the *Reference Manual*.

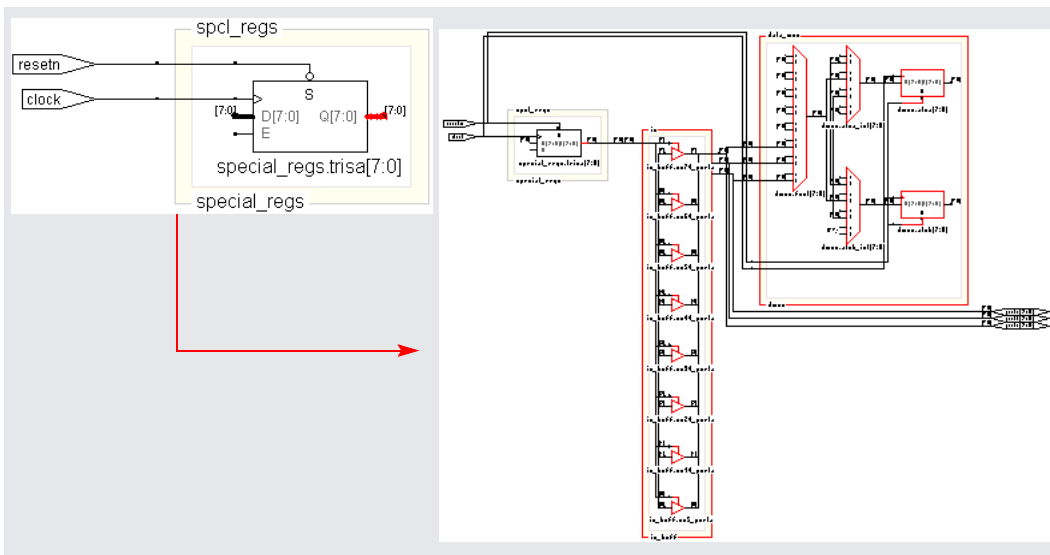
2. To expand logic from a pin at the current level only, do the following:
  - Select a pin, and go to the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.
  - Select Expand or Expand to Register/Ports. The commands work as described in the previous step, but they do not 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 <b>Select Net Driver</b> . The result is a filtered view with the net driver selected ( <i>Selecting the Net Driver Example, on page 664</i> ). |
| Trace the driver, across sheets if needed | Select a net and select <b>Go to Net Driver</b> . The software shows a view that includes the net driver.                                                               |
| Select all instances on a net             | Select a net and select <b>Select Net Instances</b> . You see a filtered view of all instances connected to the selected net.                                           |

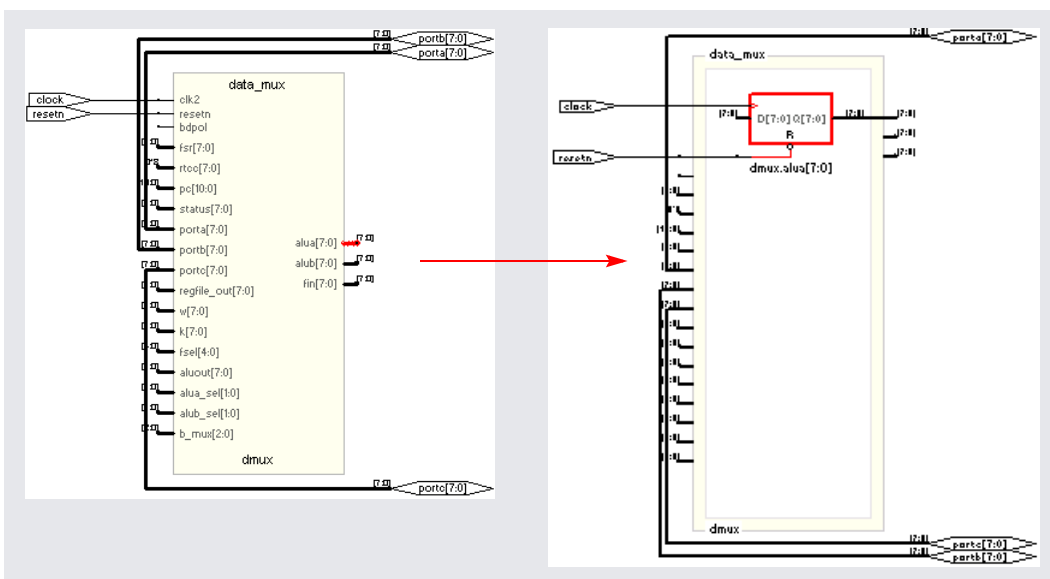
### Expanding Filtered Logic Example



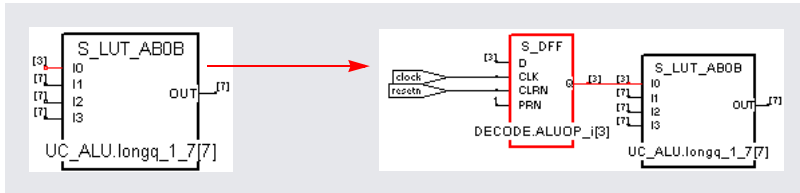
## Expanding Filtered Logic to Register/Port Example



## Expanding Inwards Example



## Selecting the Net Driver Example



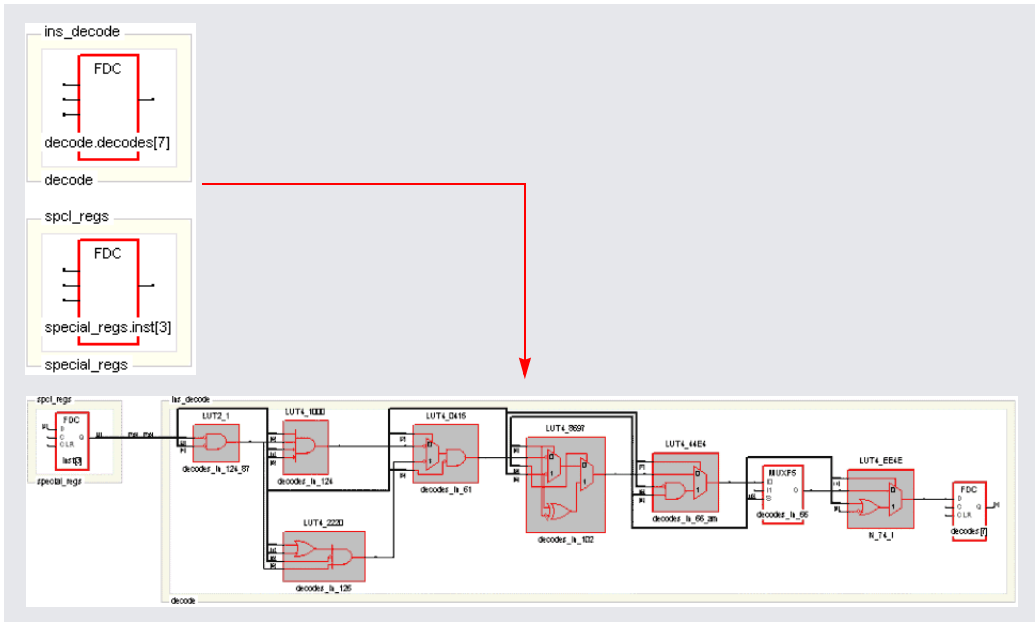
## Expanding and Viewing Connections

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 660](#). 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 730](#).

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 370](#) 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.



- 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.

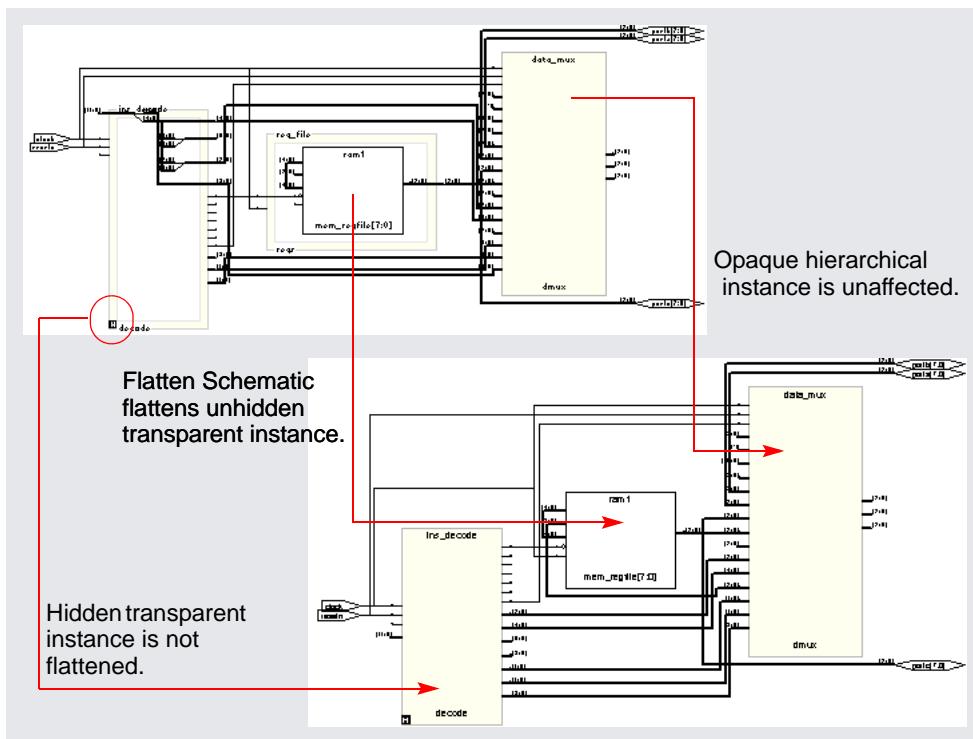
The software flattens the top-level design and displays it in a new window. To return to the top-level design, right-click and select Unflatten Schematic.

Unless you really require the entire design to be flattened, use Push/Pop mode and the filtering commands ([Filtering Schematics, on page 658](#)) to view the hierarchy. Alternatively, you can use one of the selective flattening techniques described in subsequent steps.

2. 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.

- 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.

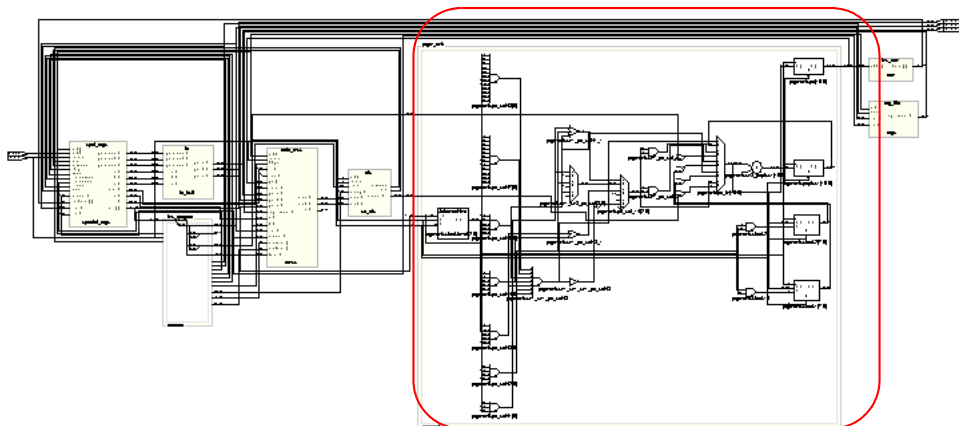
#### **Starting View    Software Generates a...**

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 655</a> . Use the <b>Back</b> 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 <b>Unflatten Schematic</b> to return to the top-level unflattened view. You cannot use the <b>Back</b> button to return to previous views because this is a new view.                                                                                                |

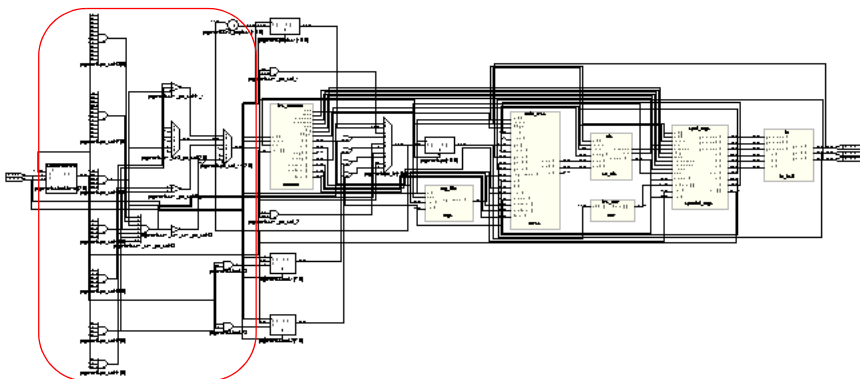


The following figure illustrates this.

Dissolved logic for prgmcntr shown nested when started from filtered view



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 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 665](#) for details. For more information on the Expand Paths and Isolate Paths commands, see [RTL View and Technology View Popup Menu Commands, on page 292](#) 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.

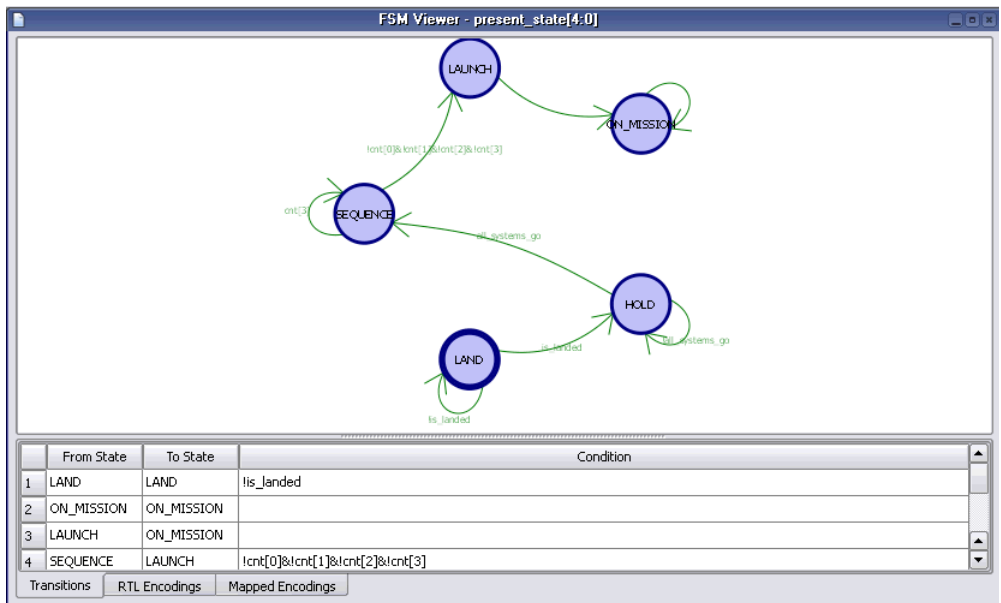
## Using the FSM Viewer

The FSM Viewer option is available only in the Synplify Pro and Synplify Premier tools. 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 either the FSM

Compiler or the FSM Explorer. For more information, see [Running the FSM Compiler, on page 455](#) and [Running the FSM Explorer, on page 458](#), 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).

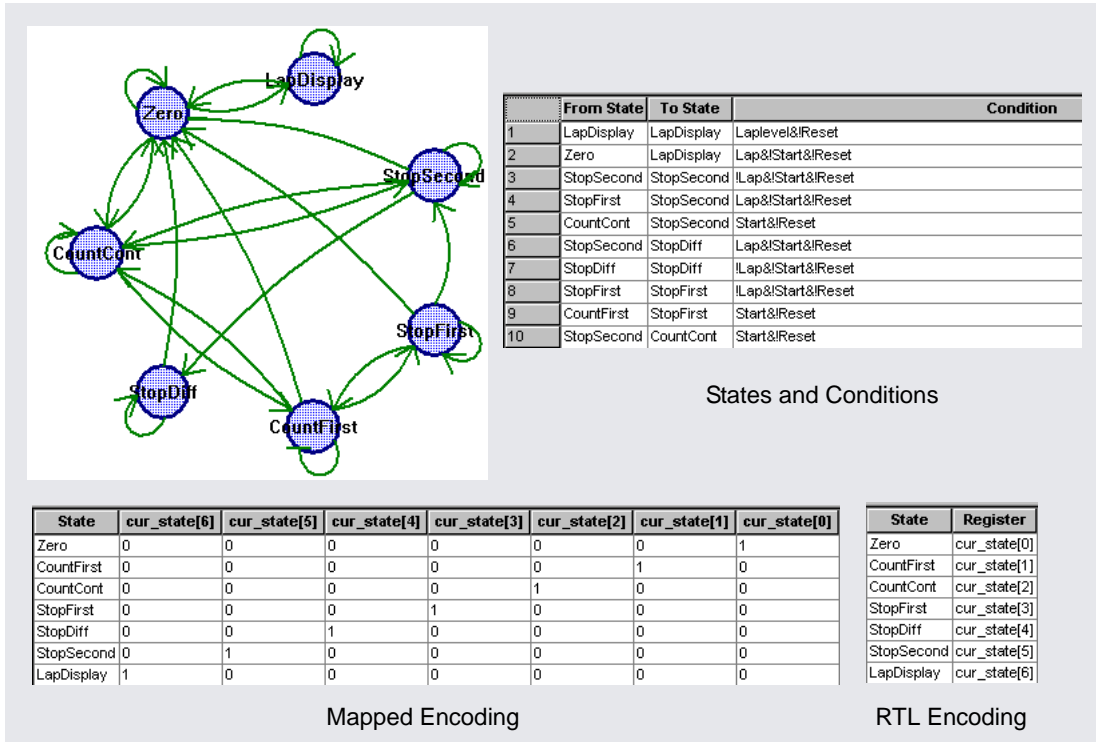
The FSM viewer opens. The viewer consists of a transition bubble diagram and a table for the encodings and transitions. If you used Verilog to define the FSMs, the viewer displays binary values for the state machines if you defined them with the ``define` keyword, and actual names if you used the `parameter` 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).                                          |
| Just 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. |

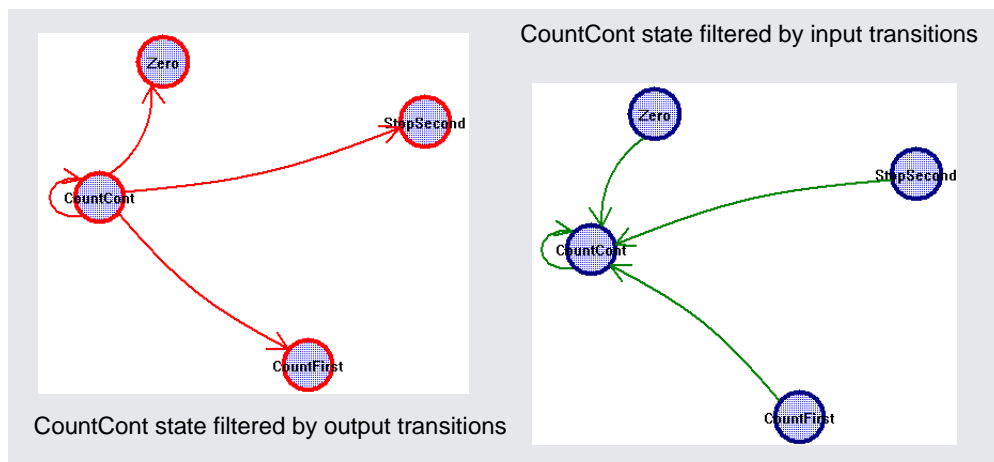
This figure shows you the mapping information for a state machine. The Transitions tab shows you simple equations for conditions for each state. The RTL Encodings tab has a State column that shows the state names in the source code, and a Registers column for the corresponding RTL encoding. The Mapped 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
```



**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.



---

## CHAPTER 16

# Analyzing Designs in Physical Analyst

---

This document describes typical analysis tasks using graphical analysis with the Physical Analyst tool. It covers the following:

- [Analyzing Physical Synthesis Results](#), on page 678
- [Using Physical Analyst](#), on page 683
- [Displaying and Selecting Objects](#), on page 689
- [Querying Physical Analyst Objects](#), on page 699
- [Finding Objects](#), on page 704
- [Crossprobing in Physical Analyst](#), on page 713
- [Analyzing Netlists in Physical Analyst](#), on page 721

For information about analyzing timing in the Synplify Premier Physical Analyst tool, see [Analyzing Timing with Physical Analyst](#), on page 750.

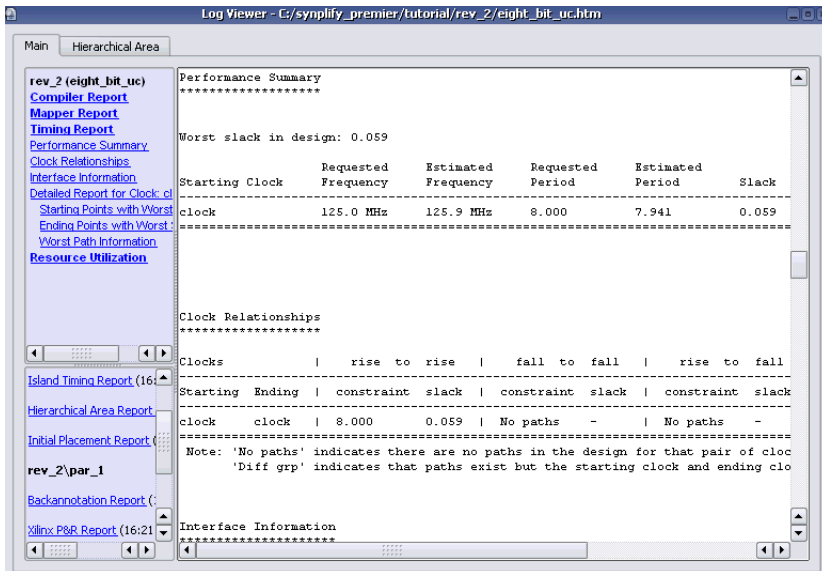
# Analyzing Physical Synthesis Results

This section contains information about tools you can use to analyze physical synthesis results. See the following:

- [Analyzing Physical Synthesis Results Using Various Tools](#), on page 678
- [Comparing Performance Results](#), on page 680
- [Running Multiple Implementations](#), on page 681
- [Checking Altera Pre-Placement Physical Synthesis Results](#), on page 681

## Analyzing Physical Synthesis Results Using Various Tools

Default timing and area reports are presented in the .htm or .srr log file for the design project. To view this information, click the View Log button in the Project view (or View->View Log File). The following .htm log file shows both the Table of Contents and the HTML log file contents for the design.



See [Viewing the Log File](#), on page 596 for complete information on how to interpret the log file results.

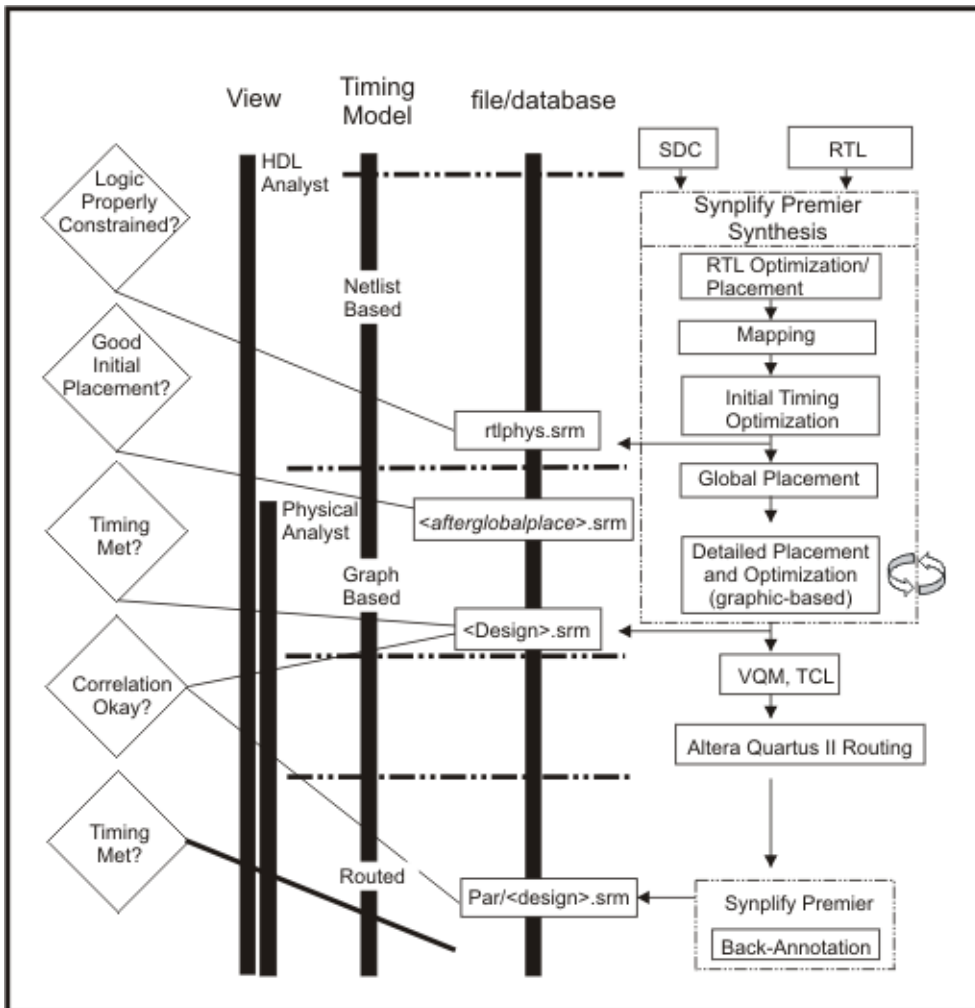
In addition, you can generate a stand-alone timing report to display more or less information than what is provided in the log file. See the following:

- [Using the Stand-alone Timing Analyst](#), on page 736
- [Using the Island Timing Analyst](#), on page 743

Also, check the place-and-route results to determine if further synthesis is required. For example, click on Xilinx P&R Report to check the `xflow_par.log` file to verify that all constraints were met as shown below. For graph-based physical synthesis, you can also click on Initial Placement Report to check the `xflow_gp.log` file. Click on Quartus P&R Report to check the `quartus.log` file for place-and-route results for Altera devices.

## Comparing Performance Results

Synplify Premier physical synthesis provides a timing closure solution that yields more accurate timing correlation and faster timing closure for your design. This section provides details on how to analyze results from logic synthesis, physical synthesis and place and route to show more accurate timing correlation between physical synthesis and final place and route results. The diagram below provides an overview of how to use the Synplify Premier tool and its features to analyze performance.



1. Determine performance improvement between logic synthesis and physical synthesis by comparing, for each clock:
  - The maximum frequency reported in the `.tan.rpt` file with the results from logic synthesis (log file).
  - The maximum frequency reported in the `.tan.rpt` file with the physical synthesis results (log file).
2. Determine the timing correlation by comparing, for the critical clocks:
  - The estimated performance in the `.srr` file.
  - The actual performance in the `.tan.rpt` file.
3. Determine the productivity gain by comparing:
  - The sum of logic synthesis runtime (reported in `.srr` file) with place-and-route runtime (reported in `quartus.log` file).
  - The physical synthesis runtime alone (reported in `.srr` file).

## Running Multiple Implementations

You can create multiple implementations of the same design so that you can compare the results of each implementation and place-and-route run. This lets you experiment with different settings for the same design with different place-and-route options. Implementations are revisions of your design within the context of the Synplify Premier software and do not replace external source code control software and processes.

For the Graph-based physical synthesis with a design plan flow, you can run the first pass using the Synplify Premier software without a design plan file (`.sfp`) to synthesize the design. Placement and routing runs automatically. Then, create a new implementation and apply a design plan for Design plan-based physical synthesis.

See [Working with Multiple Implementations](#), on page 285 for more information.

## Checking Altera Pre-Placement Physical Synthesis Results

In Altera designs, graph-based physical synthesis generates an intermediate file that you can display in the Technology View and use for debugging. Do the following:

1. After physical synthesis, open the `preplace.srm` file by double-clicking or right-clicking the file in the Project view and selecting Open.

The `preplace.srm` file is an intermediate file that captures the netlist after RTL physical synthesis and immediately before global placement, showing the same results as would be obtained from logic synthesis. This file is located in the physical synthesis implementation results directory.

2. To view the corresponding timing slack for all the clocks in the design, open the log file, and check the Pre-placement Timing Snapshot section of the log file for the critical path reflected in the `preplace.srm` file.

```

C:\synplify_premier\tutorial\rev_2\par_1\flow_par.log
284 Constraint | Check | Worst Case | Best Case | Timing | Timing
285 | | | Slack | Achievable | Errors | Score
286 -----
287 TS_clock = PERIOD TIMEGRP "clock" 8 ns HI | SETUP | 0.062ns| 7.938ns| 0|
288 GH 50% | HOLD | 0.513ns| | 0|
289 -----
290 PATH "TS_special_reg_status_7_path" TIG | SETUP | N/A| 9.547ns| N/A|
291 -----
292
293
294 All constraints were met.
295 INFO:Timing:2761 - N/A entries in the Constraints list may indicate that the
296 constraint does not cover any paths or that it has no requested value.
297
298
299 Generating Pad Report.
300
301 All signals are completely routed.
302
303 Total REAL time to PAR completion: 2 mins 5 secs
304 Total CPU time to PAR completion: 41 secs
305
306 Peak Memory Usage: 234 MB
307
308 Placement: Completed - No errors found.
309 Routing: Completed - No errors found.
310 Timing: Completed - No errors found.
311
312 Number of error messages: 0
313 Number of warning messages: 0

```

For more information on analyzing synthesis results graphically, see the following:

- [Using Physical Analyst](#), on page 683
- [Chapter 15, Analyzing with HDL Analyst and FSM Viewer](#)

## Using Physical Analyst

After you have placed and routed your design with backannotated information, you can use the Physical Analyst tool to analyze the placement and global routing. For descriptions of the interface, see the *Reference Manual*.

The Physical Analyst functionality is only available for the Synplify Premier flows:

- Graph-based physical synthesis ([Graph-Based Physical Synthesis](#), on page 42)
- Graph-based physical synthesis with a design plan ([Graph-Based Physical Synthesis with Design Planner](#), on page 46)

For both these flows, you can also include a place-and-route implementation with backannotation. The tool will display the placement information after physical synthesis is run, for the following technologies:


|        |                                                        |
|--------|--------------------------------------------------------|
| Xilinx | Virtex-II Pro, Virtex-4, Virtex-5, and Spartan-3       |
| Altera | Stratix II, Stratix II GX, Stratix III, and Stratix IV |

See the following for more information:

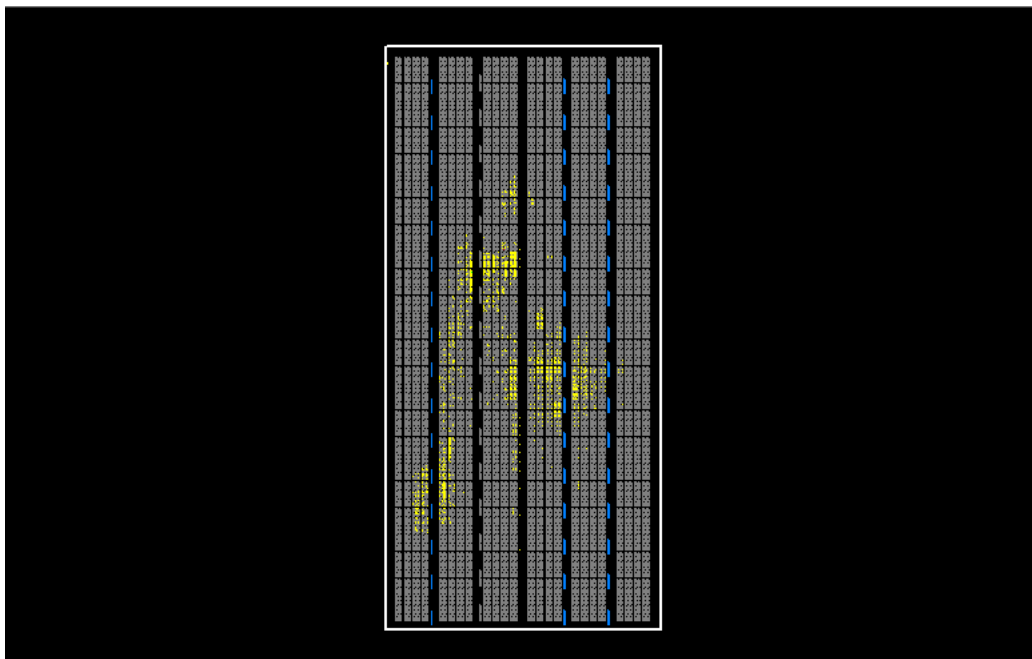
- [Opening the Physical Analyst Interface](#), on page 683
- [Zooming in the Physical Analyst](#), on page 685
- [Moving Between Views in the Physical Analyst](#), on page 686
- [Using the Physical Analyst Context Window](#), on page 687


### Opening the Physical Analyst Interface

The following procedure shows you how to open the tool and the control panel.

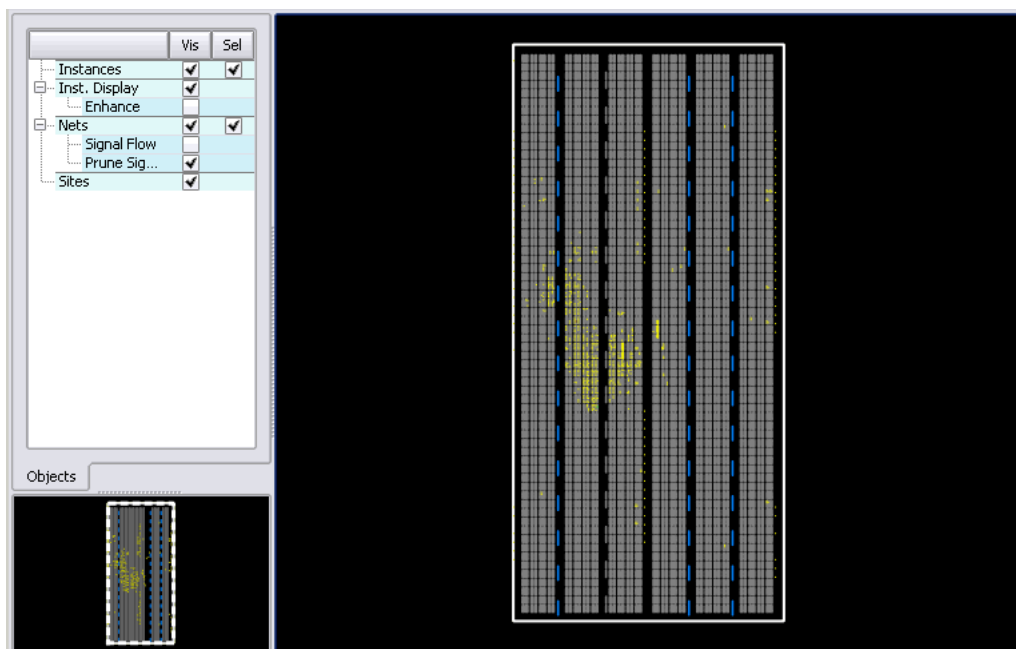
1. Open the Physical Analyst view in any of the following ways:
  - Click on the Physical Analyst icon () from the Physical Analyst toolbar.
  - Select HDL Analyst->Physical Analyst in the Project view.

- Select the .srm file, then right-click and select Open Using Physical Analyst from the popup menu.



2. To display the control panel for the Physical Analyst, do one of the following:
  - Click on the Physical Analyst Control Panel icon (  ) in the Physical Analyst toolbar.
  - Select Options->Physical Analyst Control Panel.
  - Use the keyboard shortcut key Ctrl-k.










3. To close the Physical Analyst control panel, use any of the toggle methods listed in the previous step, or right-click in the control panel and select Hide from the popup menu.

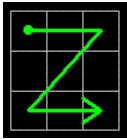
## Zooming in the Physical Analyst

Since the objects displayed in the Physical Analyst full view might be very small, a handy command to use is Zoom Selected. After selecting one or more objects in the Physical Analyst view, you can access this command by

1. If you do not have objects selected, use any of the following global zoom commands to change the display:
  - View->Zoom In from the menu or the Zoom In (  ) icon.
  - View->Zoom Out from the menu or the Zoom Out (  ) icon.
  - View->Full View from the menu or the Full View (  ) icon.
  - View->Normal View from the menu or the Normal View (  ) icon.
  - Appropriate mouse strokes.

For a description of the zoom options, see [View Menu, on page 131](#) in the *Reference Manual*. For a description of the mouse strokes, see [Help->Mouse Stroke Tutor](#).

2. To zoom into a particular object or area, select the objects and then use the Zoom Selected command in one of the following ways:
  - Click the Zoom Selected () icon.
  - Right-click and select Zoom Selected from the popup menu.
  - Use the following mouse stroke.



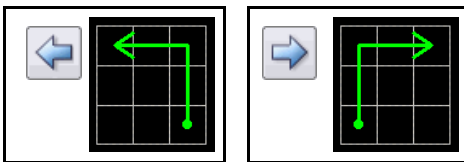
The Zoom Selected command centers the selected object or objects in the view.

## Moving Between Views in the Physical Analyst

When you filter or expand your design, you move through a number of different design views in the same window. For example, you might start with a view of the entire design, zoom in on an area and filter an object, and finally expand a connection in the filtered view, for a total of three views.

1. To move back to the previous view, click the Back icon or draw the appropriate mouse stroke in the Device window.

The software displays the last view, including the zoom factor. This does not work in a newly generated view because there is no history.



2. To move forward again, click the Forward icon or draw the appropriate mouse stroke.

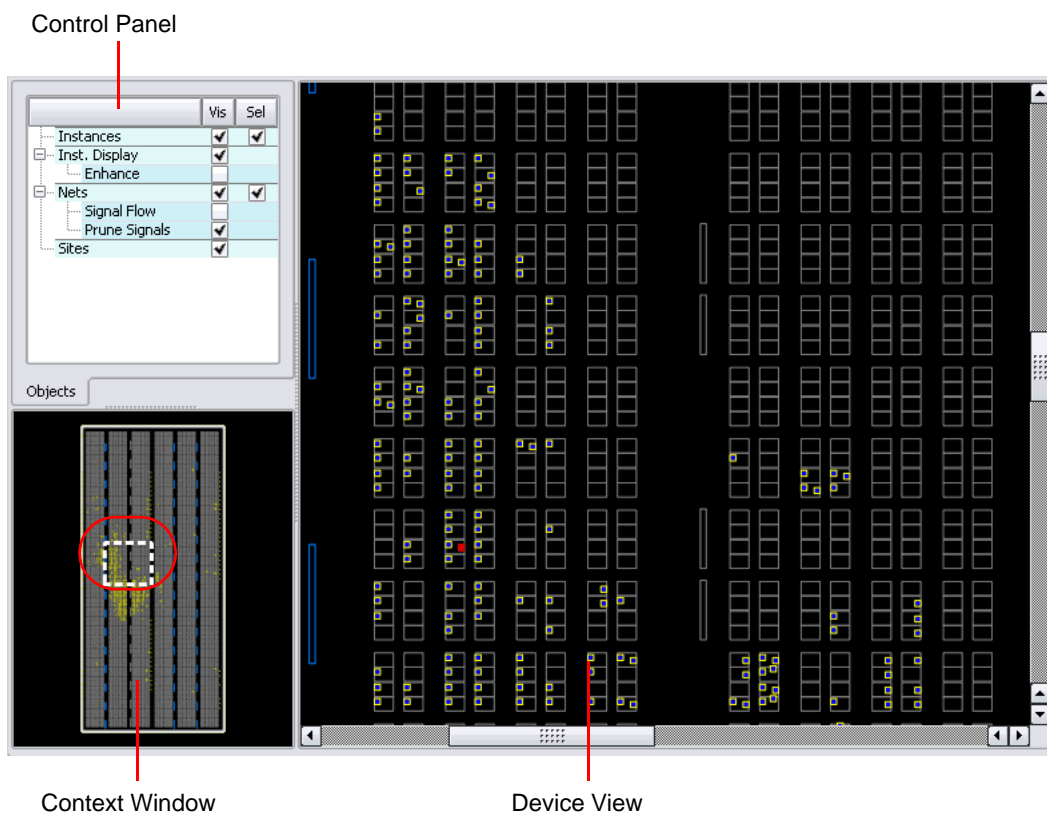
The software displays the next view in the display history.

## Using the Physical Analyst Context Window

The Physical Analyst context window occupies the lower portion of the Control Panel view. The context window provides a point-of-reference to your location on the device.

For example, suppose you:

1. Zoom in the Physical Analyst tool to get a better view of the objects you selected on the device.
2. The context window in the control panel displays a rectangle around the relevant area on the device. This is helpful because you now have a point-of-reference to your location on the device.



3. Once a rectangle area is drawn in the context window, you can then:

- move
- scroll
- stretch/shrink

this rectangle in the context window. This view will be reflected in the Physical Analyst view.

4. To reinstate the full context window view, right-click and select Refresh in this view.

## Displaying and Selecting Objects

This section describes how to display and select objects in the Physical Analyst view.

- [Setting Visibility for Physical Analyst Objects](#), on page 689
- [Displaying Instances and Sites in Physical Analyst](#), on page 690
- [Displaying Nets in Physical Analyst](#), on page 694
- [Selecting Objects in Physical Analyst](#), on page 696

### Setting Visibility for Physical Analyst Objects

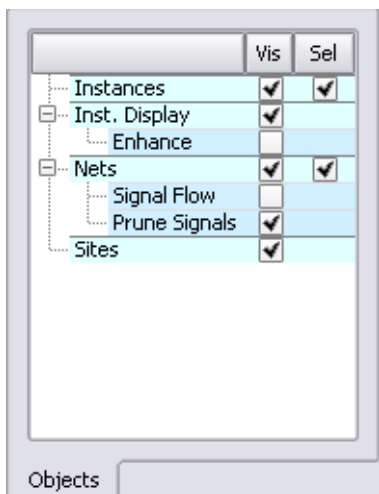
You determine object visibility and selectability by setting it in the control panel.

1. Select Options->Physical Analyst Control Panel to display the control panel.
2. To make an object visible, select the Vis box for that object.

This makes the boundaries for the selected type of object visible in the Device view. You cannot select a visible object, unless it has been made selectable. For details about object selection, see [Displaying Instances and Sites in Physical Analyst, on page 690](#), and [Displaying Nets in Physical Analyst, on page 694](#).

3. To make an object selectable, select the Vis and Sel boxes for that object.

The object must be visible before you can make it selectable. When an object is selectable, you can get detailed information by rolling the mouse over it to get a tool tip.



The Control Panel will display the following in the Physical Analyst View:

- Instances visible and selectable
- Instance internals visible
- Do not show internal signal pins
- Do not show enhanced view for instances
- Signal nets visible and selectable
- Do not show signal flow
- Show pruned signals
- Sites visible

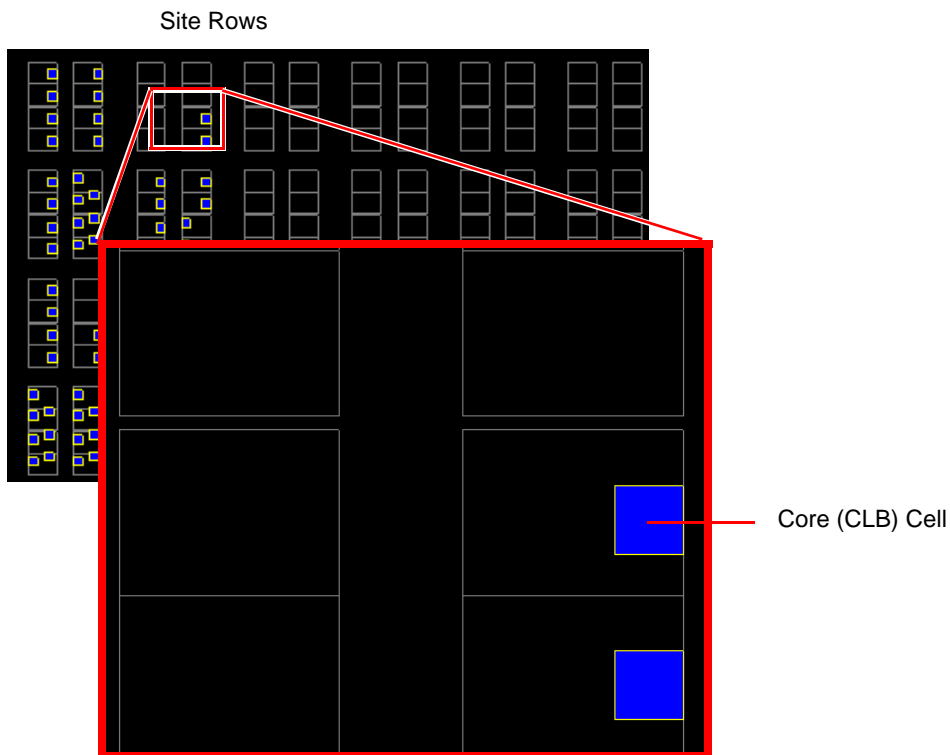
## Displaying Instances and Sites in Physical Analyst

The following procedure shows you how to display objects and sites in the Physical Analyst window.

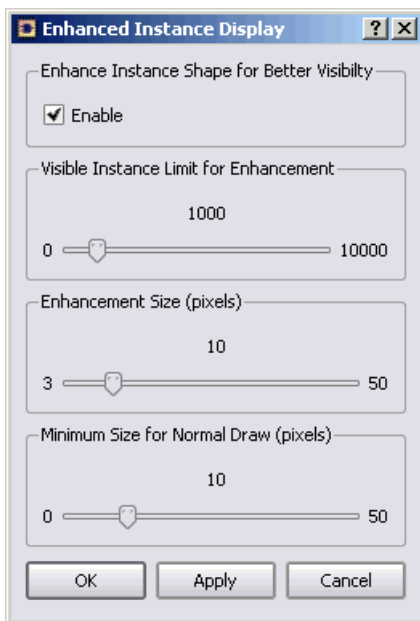
1. To display instances, open the Physical Analyst control panel and select the Vis box for Instances.

The window displays instance bounds, instance locations, and signal pins if all instances with placement information. It does not display the signal pins of the instances.

The following figure shows cell boundaries.



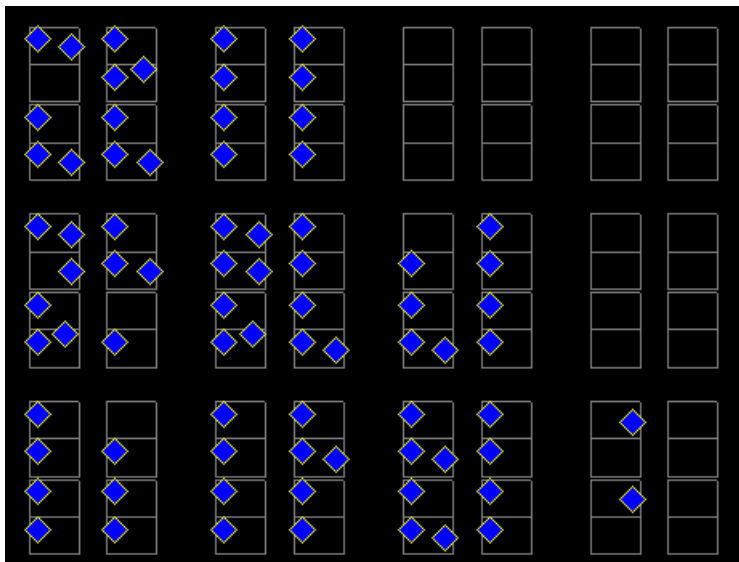
2. To display core cells at a fixed size regardless of zoom level, do the following:
  - With the Physical Analyst window active, select View->Configure Enhanced Instance Display.



- Enable the Enhance Instance Shape for Better Visibility option.
- Set any other options you want and click OK.

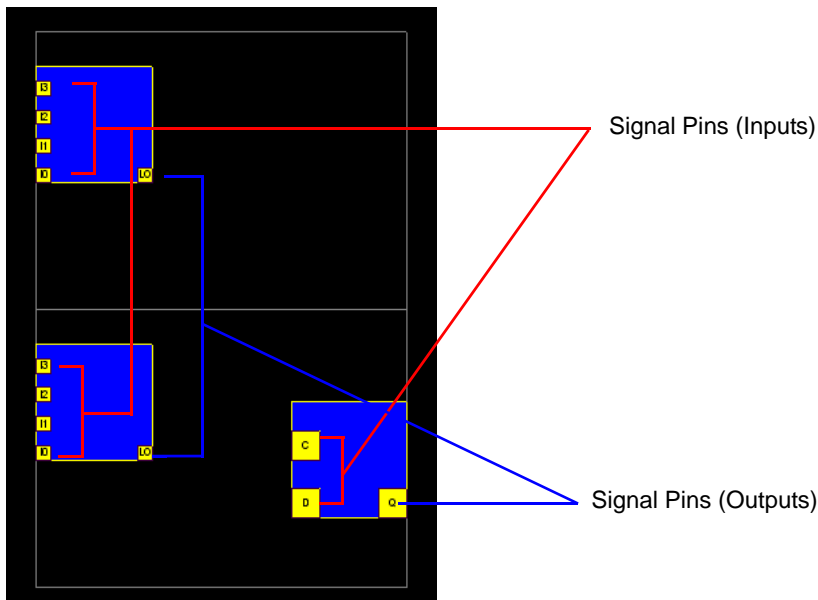
The tool displays the core cells as diamonds of a fixed size.





3. To display instances and their signal pins, select the Vis boxes for Instances, Inst Display.

The Vis box for Signal Pins is selected automatically, and the tool displays the signal pins. The following figure shows signal pins displayed:



4. To display sites, do the following:
  - Select the Vis box for Sites.
  - To view sites more clearly, turn off the visibility of instances.

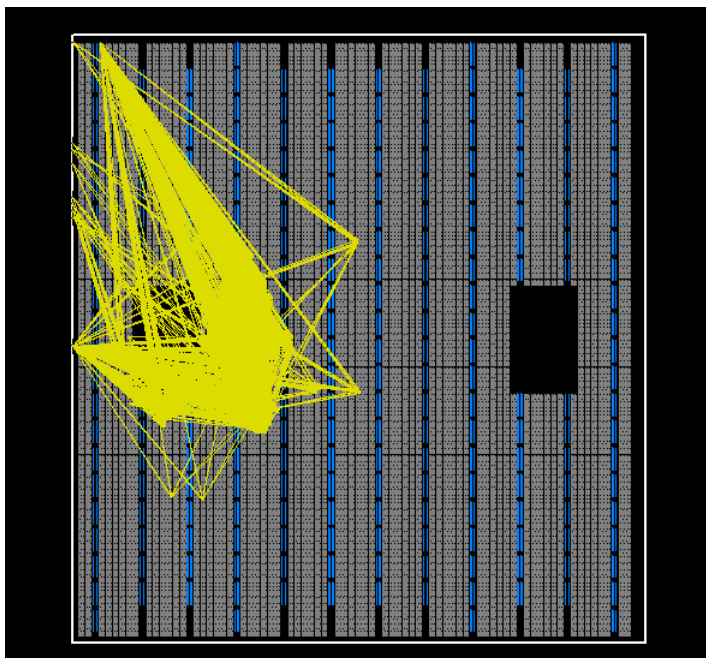
## Displaying Nets in Physical Analyst

When nets are routed, they are connected to their respective instances. Because of the long load time and the limited visibility when nets are super-imposed on the view, net routes are not displayed by default. Nets are routed from output pins to input pins and are shown with their corresponding point-to-point connections on one layer of the device in the Physical Analyst view.

1. Route the nets, using one of these methods:
  - Use an on-demand routing command like *Expand* or *Show Critical Path*. For more netlist commands, see [Analyzing Netlists in Physical Analyst](#), on page 721.

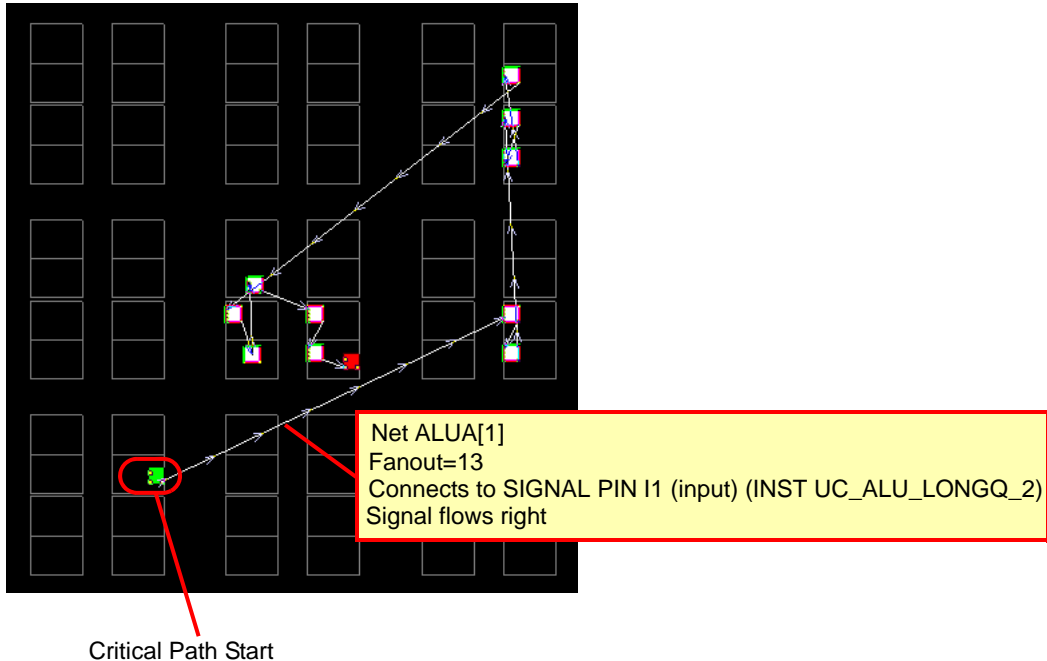
- Right-click and select Route Selected Instances. You can choose to display nets for connected instances only, as well as, from input pins, to output pins, or to all pins for selected instances.
2. Set the visibility options on the control panel. Select the Vis box for Nets. You must select this to make the other options available.


The following example shows point-to-point net routing:



3. To reduce clutter in the display, try the following techniques:
  - Isolate a critical path, or filter the design to show just a few paths you want to analyze.
  - In the control panel, select the Vis box for Prune Signals. When enabled, this option displays net segment that connect to an instance that is invisible because of filtering, for example, in a diminished color.
4. To view the direction of a signal, do either of the following:
  - Place the cursor over a net to view the predominant direction of its signal flow (right, left, up, or down).

- Select the Vis box for Signal Flow. The net is displayed with arrows showing the direction of its signal flow. The Signal Flow option is useful when you display the critical path. You can follow the arrows and lines along the critical path from the start point to the end point.



5. To hide all nets in the display, do either of the following:
  - Right-click in the Physical Analyst and select Unfilter->Show All Instances, Hide All Nets from the popup menu.
  - Click the Reset filter icon (  ).

## Selecting Objects in Physical Analyst

Selected objects are highlighted in the Physical Analyst view. If you have other windows open and crossprobing enabled, you can highlight the selected object in the other windows too.

The following procedure shows you ways to select objects.

1. To select an object, do the following:

- In the control panel, select the Sel box for the object to make it selectable.
  - Click on the object in the Device view.
2. To select multiple objects, use one of these methods:
- Draw a rectangle around the objects.
  - Select an object, press Ctrl, and click other objects you want to select. You can also deselect from the list of currently selected objects while holding the Ctrl key.
  - Position the cursor over an object and click the right mouse button; the object is automatically selected in the view. To preserve a prior selection, hold the Ctrl key and press the right mouse button.
  - Right-click and choose one of the Select commands from the popup menu.
3. To limit the selection range, use these techniques:
- Use the Physical Analyst control panel to enable or disable a class of objects to be selected. For example you can disable selection for nets, and only select instances in an area.
  - Use the Filter and Unfilter commands to restrict the scope of selection.
  - Use Find to select the objects you want. You can also use the Find command to select a subset of objects of a particular type. See [Using Find to Locate Physical Analyst Objects](#), on page 704.
  - Use the Go to Location command and specify an object location in microns to go directly to a coordinate pair location. See [Finding Physical Analyst Objects by Their Locations](#), on page 708.
4. To select a net or instance that overlaps another, do the following.
- Move your cursor over the overlapping object you want to select. The cursor changes shape to indicate that you need to resolve the selection.
  - Click the cursor, and the Resolve Selection dialog box opens.



- Select the object you want from the list and click Close.

Once you have selected an object, the software highlights the selected object in the Physical Analyst window. If you have other windows open and crossprobing enabled, the object is highlighted in the other windows too.

# Querying Physical Analyst Objects

The following procedures describe how to view object properties in the Physical Analyst:

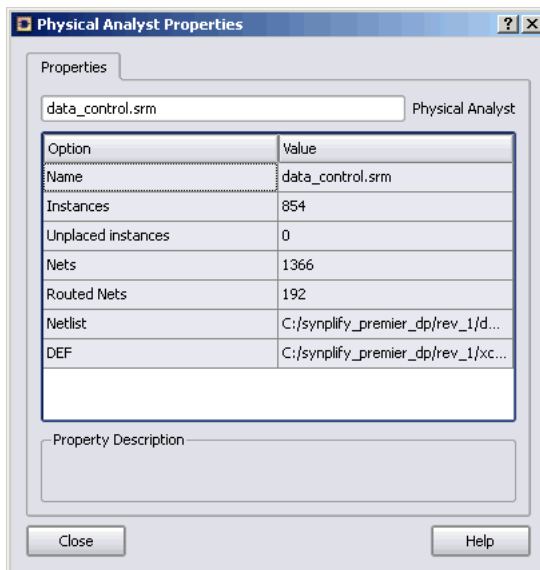
- [Viewing Properties in Physical Analyst](#), on page 699
- [Using Tool Tips to View Properties in Physical Analyst](#), on page 702

## Viewing Properties in Physical Analyst

You can view properties for the device design and for selected objects displayed in the view using the popup menu commands described here. For site properties, use tooltips, as described in [Using Tool Tips to View Properties in Physical Analyst](#), on page 702.

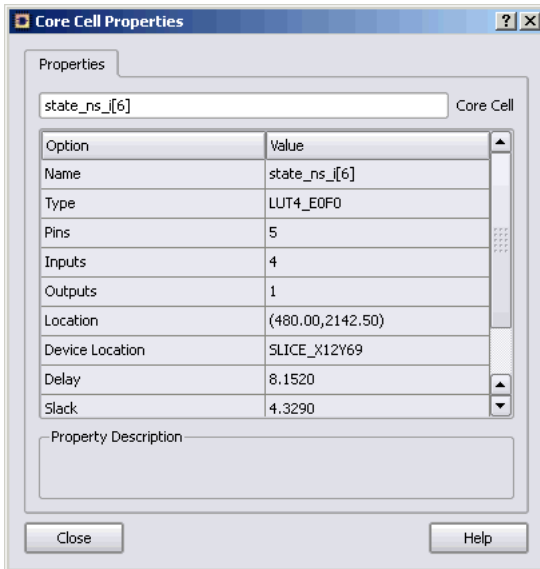
1. To view general information for the design, right-click anywhere in the Physical Analyst view, and select Physical Analyst Properties from the popup menu.

The dialog box shows information like the design name, the number of instances, unplaced instances, routed nets in the design, and the location of the netlist and floorplan (.def) files.



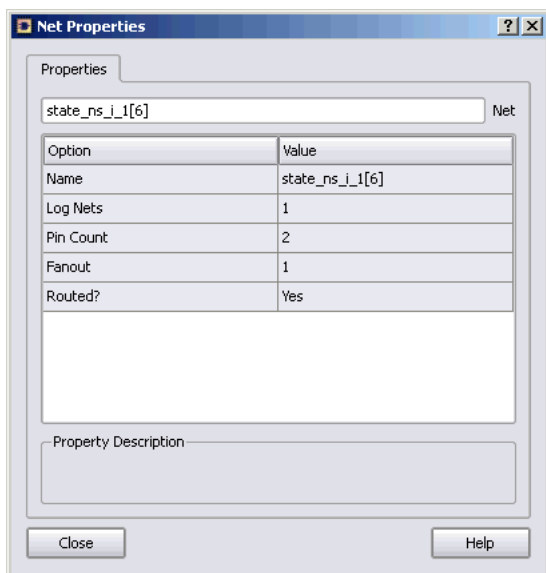
- To view properties for an instance, right-click the instance and select Properties (Core Cell).

The dialog box lists information like the instance name, type, and pins; placement information like its placement location and device-specific location; and its delay, slack, and clock signal. It also indicates whether the instance is included in the critical path.



- To view properties for a net, right-click the net and select Properties (Net).





The dialog box lists information like the net name, logical nets, pin count, and fanout. It also indicates if the net is a clock and if it has been globally routed.

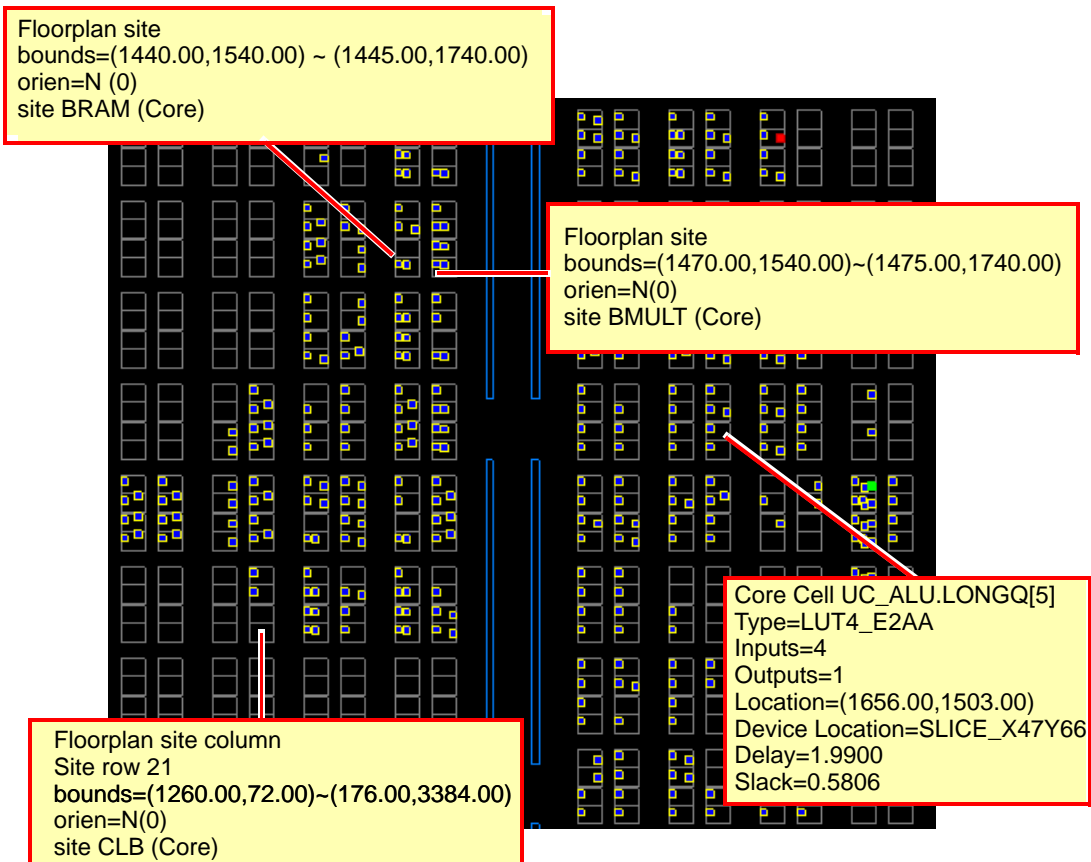
4. Click on an item in the list to view a definition of that term below.

## Using Tool Tips to View Properties in Physical Analyst

You can use this method to view properties for any objects in the design, as well as for various UI features. For instance, net, and design properties, you can also use the popup commands described in [Viewing Properties in Physical Analyst](#), on page 699.

1. Enable View->Tool Tip.
2. Move your mouse over an object.

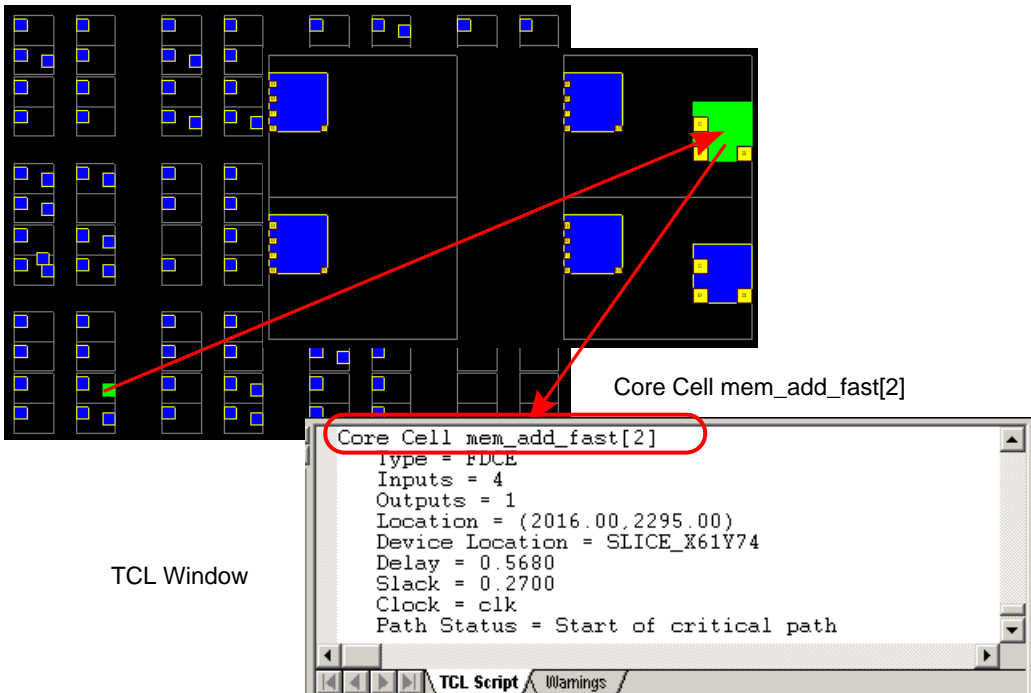
As you move the mouse over an object, you see information about that object. Coordinates for the objects are in microns. The following is an example of information for different objects.



3. To display the tooltip information in the Tcl window, do the following:
  - Select an object directly. Do not use this method with area selections or when objects are selected using other commands such as Expand or Find.
  - Either select View->Selection Transcription or right-click in the Physical Analyst view and select Selection Transcription from the popup menu.

The tooltip information is displayed in the Tcl window. You can copy and paste the information from the TCL window into other windows or files like the SCOPE window, the Find Object dialog box, or a text file.

The following figure shows an example of an object selected on the device and its tooltip information displayed in the TCL window.



## Finding Objects

To find and display objects in the Physical Analyst view, use the following options:

- [Using Find to Locate Physical Analyst Objects](#)), on page 704
- [Finding Physical Analyst Objects by Their Locations](#), on page 708
- [Using Markers to Find Physical Analyst Objects](#), on page 709
- [Identifying Encrypted IP Objects in Physical Analyst](#), on page 711

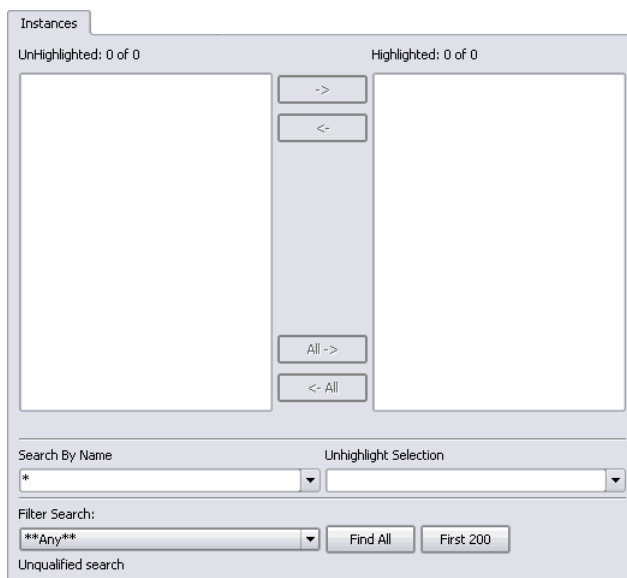
### Using Find to Locate Physical Analyst Objects)

This procedure shows you how to use the Find command to do a search on the entire design. The view displayed is flat, although the hierarchy of instance names is retained. The Find command does not include physical instances in its search.

1. To find nets, first make sure their display is enabled by selecting View->Unfilter->Show All or the corresponding command from the popup menu.
2. Right-click and select Find from the popup menu or press Ctrl-f. Move the dialog box so you can see both the view and the dialog box.

You can also open the dialog box by selecting Edit->Find from the menu or by clicking the Find icon in the tool bar.

3. Select the tab (at the top of the dialog box) for the type of object. The Unhighlighted box on the left will list objects of the selected type (instances, symbols, nets, or ports).



4. You can optionally restrict the scope of your search for the design in the following ways:
  - To filter the search using wildcards, see [Using Wildcards with the Find Command, on page 706](#).
  - To further filter the object type, see [Using Object Filters with the Find Command, on page 707](#).

The Unhighlighted box shows available objects within the scope you set when you click Find 200 or Find All. Objects are listed in alphabetical order.

5. Do the following to select objects from the list.
  - Click First 200 or Find All. The Unhighlighted box shows available objects (in alphabetical order) within the scope you set when you 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.
  - Move the objects you want to the Highlighted box by double-clicking them, or by clicking them to select them, and then clicking the right arrow. Right-click on the objects you want from the list.

- If the object name exceeds the width of the Unhighlighted box, check the object name by clicking the entry in the list, and viewing the entire name in the field below the Unhighlighted box.

Objects transferred to the Highlighted box are automatically highlighted in the view.

You can leave the dialog box open to do successive Find operations. Close the dialog box when you are done.

## Using Wildcards with the Find Command

The following procedure shows you how to use patterns or wildcards to restrict the search range with the Find command when you are locating objects in the Physical Analyst view. You can use wildcards to avoid typing long path names. Start with a general pattern, and then make it more specific.

1. Follow the first three steps described in [Using Find to Locate Physical Analyst Objects](#), on page 704.
2. Type a pattern in the Search By Name field.

\* The asterisk matches any sequence of characters.

---

? The question mark matches any single character.

---

. The dot (period) explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not as a hierarchy separator, type a backslash (\) before the dot.

---

When you use wildcards between hierarchies, all pattern matching is displayed from the top level to the lowest level hierarchy, inclusively.

3. Click First 200 or Find All.

The Unhighlighted box lists the objects that match the wildcard pattern criteria. If you selected First 200, it lists the first 200 matches, and then you can click the button again to find the next 200.

4. Select and move the objects you want to the Highlighted box by doing one of the following:
  - Select the objects in the Unhighlighted box and click the right arrow.
  - Double-click individual items in the Unhighlighted box.

The objects are automatically highlighted in the view.

## Using Object Filters with the Find Command

Use the Filter Search option on the Find Object dialog box to limit the objects you are searching for to a particular subcategory. This can be very useful in large designs.

1. Follow the first three steps described in [Using Find to Locate Physical Analyst Objects](#), on page 704.

At this point, you have already restricted your search to a certain type of object by selecting one of the tabs at the top.

2. Select a subcategory from the pull-down list in Filter Search.

The listed subcategories are for the kind of object you have already selected. For descriptions of the various subcategories, see [Object Filter Search for Find Command](#), on page 532 of the Reference Manual.

3. Click First 200 or Find All.

The Unhighlighted box lists the objects that match the filtered object criteria. If you selected First 200, it lists the first 200 matches, and then you can click the button again to find the next 200.

4. Select and move the objects you want to the Highlighted box by doing one of the following:
  - Select the objects in the Unhighlighted box and click the right arrow.
  - Double-click individual items in the Unhighlighted box.

The objects are automatically highlighted in the view.

## Finding Physical Analyst Objects by Their Locations

The Go to Location command allows you to specify a coordinate pair location or location of an object, and then zoom in on this location if requested.

This procedure shows you how to use the Go to Location command to search for objects, for example instances.

1. In the Physical Analyst view, type Ctrl-g or right-click and select Go to Location from the popup menu.

The command displays the Goto Location dialog box.



2. Do one of the following:
  - Enter the coordinates of the location (see step 3 for details).
  - Select a marker from the Marker pull-down. For information about creating markers at object locations, see [Using Markers to Find Physical Analyst Objects, on page 709](#).
3. Enter a coordinate pair (X and Y) location value in microns. You can do this in any of these ways:
  - Type a coordinate pair in the field.



The syntax is very flexible, providing various ways to separate coordinates. You can use a space, or one of the following punctuation marks: a comma (,), semi-colon (;), or colon (:). Optionally, enclose the coordinate pair location in parentheses.

- Copy and paste a coordinate pair location from a log file (.srr) or timing analyst file (.ta).
- Copy a location from a .def file. Note that the unit of measurement in the .def file is database units. Use the UNITS DISTANCE MICRONS factor from the .def file to convert database units to microns, before using it here.
- If you have used the command before and have a history of locations, select a location from the pull-down list in the History field.

A description of the object shows in the dialog box window, if applicable.

4. Select a zoom mode.
  - To center the location, without zooming, select Scroll.
  - To zoom into the selected area, select Zoom to Object.
  - To zoom at the 100% level, select Zoom Normal.
5. Click OK.

The Physical Analyst view shows the location you specified, at the zoom level you specified. The command keeps a running history of the locations you specified, and they appear in the History pull-down the next time you use the command.

## Using Markers to Find Physical Analyst Objects

Markers are bookmarks for physical coordinates in the Physical Analyst view. Markers are useful for analyzing floorplan placement in the Physical Analyst view. You can find an object, such as an instance, then create a marker on this instance. When multiple markers are defined, you can move from marker to marker, as well as, measure the distance from a marker or between any two markers.

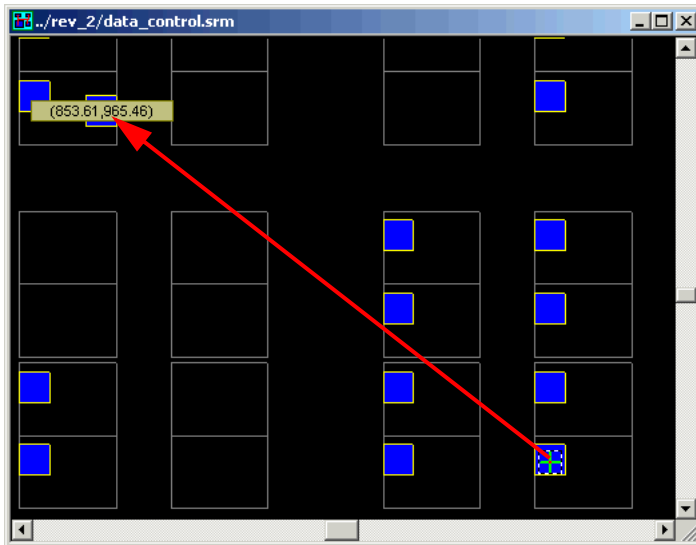
1. You can create a marker in either of these ways:
  - Select an object or click on the spot where you want to place the marker. Type Ctrl-m, or right-click and then select Markers->Add Marker from the popup menu. If a marker is created at an instance or net

location, the marker's name is Marker\_<object\_name>. All other markers are named Marker1, Marker2, etc.

- Click on the spot where you want to create the marker. Either type Ctrl-g or right-click and select Go to Location to open the Go to location dialog box. Your coordinates appear in the coordinates field. Check the Create Marker box in the dialog box. Either specify a name for the marker, or use the default marker name, which is GotoMarker1, GotoMarker2, etc. Click OK.

A marker symbol (⊕) appears in the Physical Analyst view at the location specified. As you move the cursor over the marker, a tool tip shows the marker name and its X and Y coordinates. You can also view this information by selecting the marker, right-clicking, and selecting Properties. The marker is automatically added to the list in the Go to Location dialog box, and you can use it to locate objects, as described in [Finding Physical Analyst Objects by Their Locations, on page 708](#).

2. To move a marker, select the symbol (⊕). Press the mouse button, drag the marker to its new location, and then release the mouse button.



3. To delete a marker, select the marker and press the Del key. Alternatively, right-click and select Markers->Remove Selected from the popup menu. To delete all markers, right-click and select Markers->Remove All from the popup menu.

4. To use markers to locate an object, see the procedure described in [Finding Physical Analyst Objects by Their Locations, on page 708](#).
5. Do the following to use markers for measuring distances:
  - To measure the distance from a marker to the cursor location, select the marker and position the cursor. The status bar at the bottom of the Physical Analyst view displays the manhattan distance (X+Y) between the two points, calculated in microns. It also displays the XY coordinates for the cursor.
  - To measure the distance between two markers, select two markers. The distance is displayed in the status bar. If you have more than two markers selected, the distance is not calculated.
6. To navigate from one marker to another, do the following:
  - To advance to the next marker, right-click and select Markers->Go to Next from the popup menu, or use the F2 key.
  - To go to the previous marker, right-click and select Markers->Go to Previous from the popup menu or use the Shift+F2 keys.

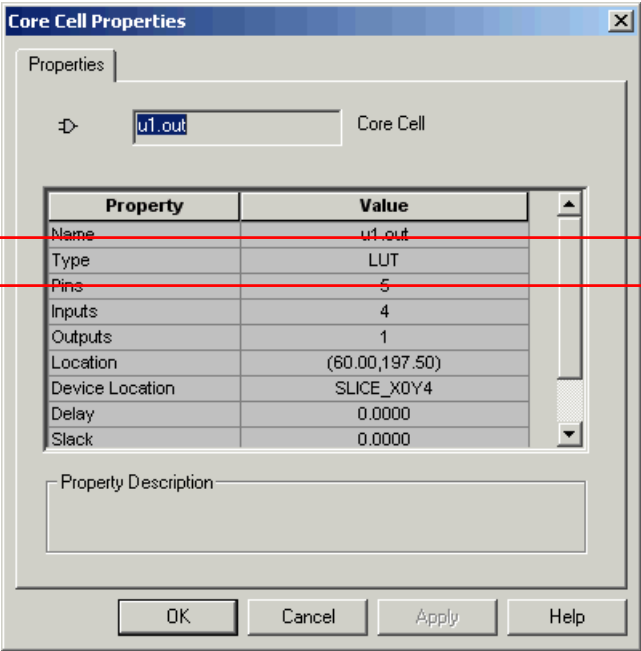
The sequence for the markers is the order in which they were created. If the view is zoomed, the selected marker is centered in the view.

## Identifying Encrypted IP Objects in Physical Analyst

You can use the Physical Analyst to identify cells that belong to an encrypted IP. When a cell belongs to an encrypted IP, it is implemented as the type LUT.

To view objects that might belong to encrypted IPs in the Physical Analyst view, perform the following tasks:

1. Select the cell.
2. Right-click and select Properties (Core Cell) from the popup menu.
3. Notice that the Type property is set to LUT in the dialog box.



# Crossprobing in Physical Analyst

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. The Physical Analyst responds to incoming cross probes as well as sending out cross probes in response to selections.

For details, see

- [Crossprobing from the Physical Analyst View](#), on page 713
- [Crossprobing from a Text File to Physical Analyst](#), on page 716
- [Crossprobing from the RTL View to Physical Analyst](#), on page 717
- [Crossprobing from the Technology View to Physical Analyst](#), on page 719

## Crossprobing from the Physical Analyst View

1. Set crossprobing options.
  - To crossprobe automatically from the Physical Analyst view, check that View->Send Crossprobes when selecting is enabled.
  - To crossprobe only on demand, disable View->Send Crossprobes when selecting. This can be more efficient with large designs.
2. To crossprobe to different files and views, follow the appropriate steps:

| <b>To Crossprobe to..</b> | <b>Procedure</b> |
|---------------------------|------------------|
|---------------------------|------------------|

---

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Source code | <ul style="list-style-type: none"><li>• Make sure that View-&gt;Crossprobing-&gt;Crossprobing to HDL Source is enabled.</li><li>• In the Physical Analyst view, double-click the object you want to crossprobe.<br/>If the source code file is not open, a Text Editor window opens to the appropriate section of code (for example, modules or instances). If the source file is already open, the software scrolls to the correct section of the code and highlights it.</li></ul> |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

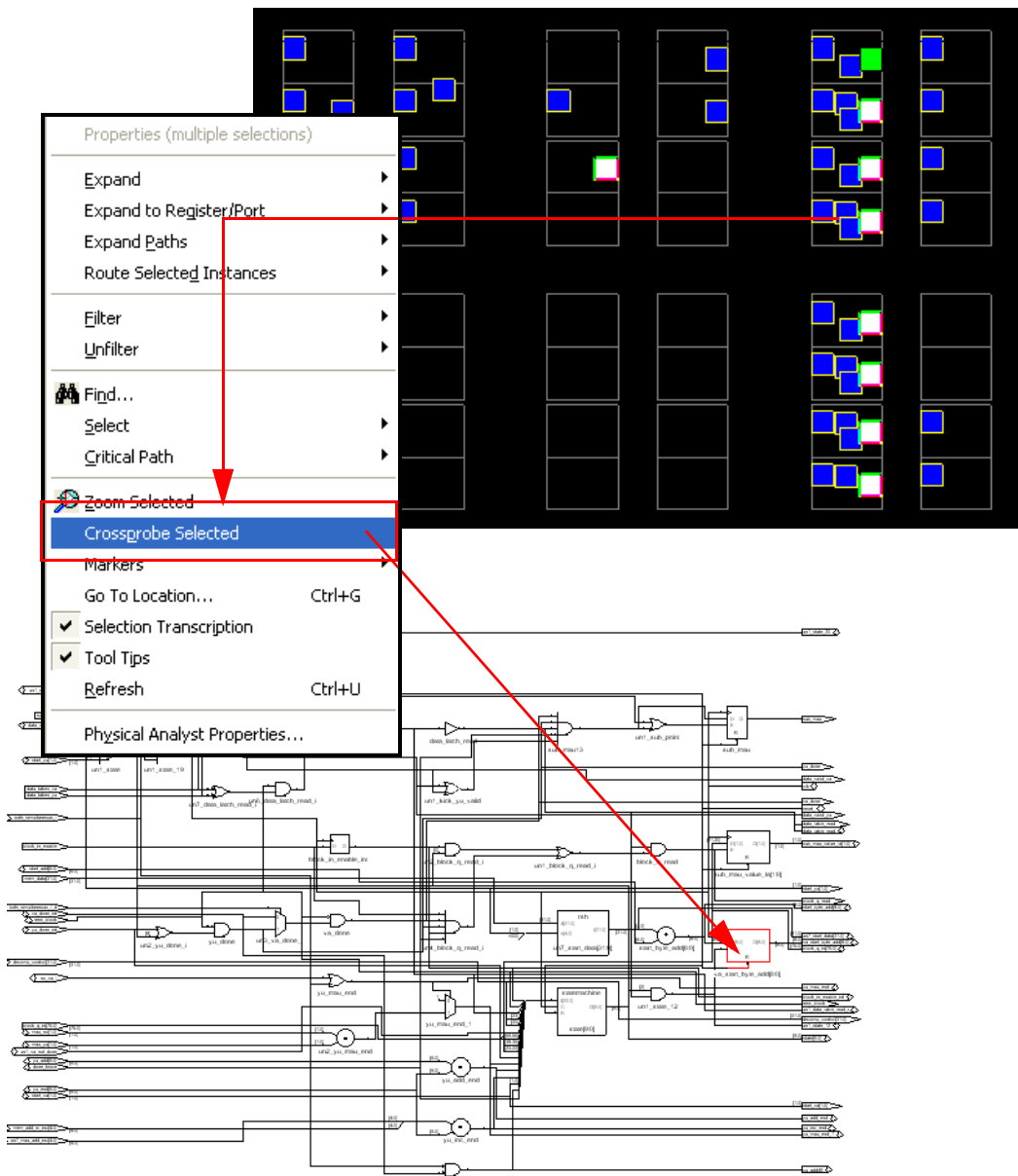
**To Crossprobe to.. Procedure**

---

|                 |                                                                                                                                                                                                                                                                                                                          |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RTL view        | <ul style="list-style-type: none"><li>• Open the RTL view.</li><li>• Make sure the crossprobing options (see step 1) are set for your needs.</li><li>• In the Physical Analyst view, select the object you want to crossprobe. The software highlights the corresponding object in the RTL view.</li></ul>               |
| Technology view | <ul style="list-style-type: none"><li>• Open the Technology view.</li><li>• Make sure the crossprobing options (see step 1) are set for your needs.</li><li>• In the Physical Analyst view, select the object you want to crossprobe. The software highlights the corresponding object in the Technology view.</li></ul> |

---

The following shows on-demand crossprobing from the Physical Analyst view:



## Crossprobing from a Text File to Physical Analyst

Instances from a text file, such as the HDL source code (Verilog/VHDL) or log file (.srr) can be highlighted in the Physical Analyst. Make sure the Physical Analyst view is already open.

1. Open the Physical Analyst view.
2. In the text file, highlight the appropriate portion of the text, like the hierarchical instance name.

For some objects in source code files, you might have to select an entire block of text.

3. Crossprobe the object.
  - To show only the object selected, click the Filter Schematics icon in the toolbar.
  - Right-click in the text file and select **Select in Analyst** from the popup menu.
4. Check the Physical Analyst view.

The selected instances are highlighted in this view. If the selected object does not have visibility enabled for it in the Control panel, the visibility will be automatically enabled.



Log File

Ending Points with Worst Slack  
\*\*\*\*\*

| Instance            | Starting Reference Clock | Type | Pin | Net                      | Required Time | Slack  |
|---------------------|--------------------------|------|-----|--------------------------|---------------|--------|
| UC_ALU.alu          | Clk                      | FDC  | D   | N_123_i                  | 6.667         | -0.660 |
| PrgmCtr.pc[5]       | Clk                      | FDPE | D   | pc_8[5]                  | 6.404         | 0.046  |
| Dmux.alu_a[1]       | Clk                      | FDC  | D   | ALU_d[1]                 | 6.441         | 0.132  |
| Dmux.alu_a_fast[0]  | Clk                      | FDC  | D   | ALU_d[0]                 | 6.404         | 0.177  |
| Dmux.alub_fast[5]   | Clk                      | FDC  | D   | ALU_d[5]                 | 6.404         | 0.288  |
| Dmux.alub[5]        | Clk                      | FDC  | D   | ALU_d[5]                 | 6.441         | 0.336  |
| Dmux.alub[3]        | Clk                      | FDC  | D   | ALU_d[3]                 | 6.441         | 0.406  |
| SPECIAL_REGS.portbr | CE                       |      |     | portbregister_1_scmuxa_1 | 6.206         | 0.432  |
| SPECIAL_REGS.portbr | CE                       |      |     | portbregister_1_scmuxa_1 | 6.206         | 0.432  |
| SPECIAL_REGS.trisb  | CE                       |      |     | trisb_reg_1_scmuxa       | 6.206         | 0.462  |

Physical Analyst View

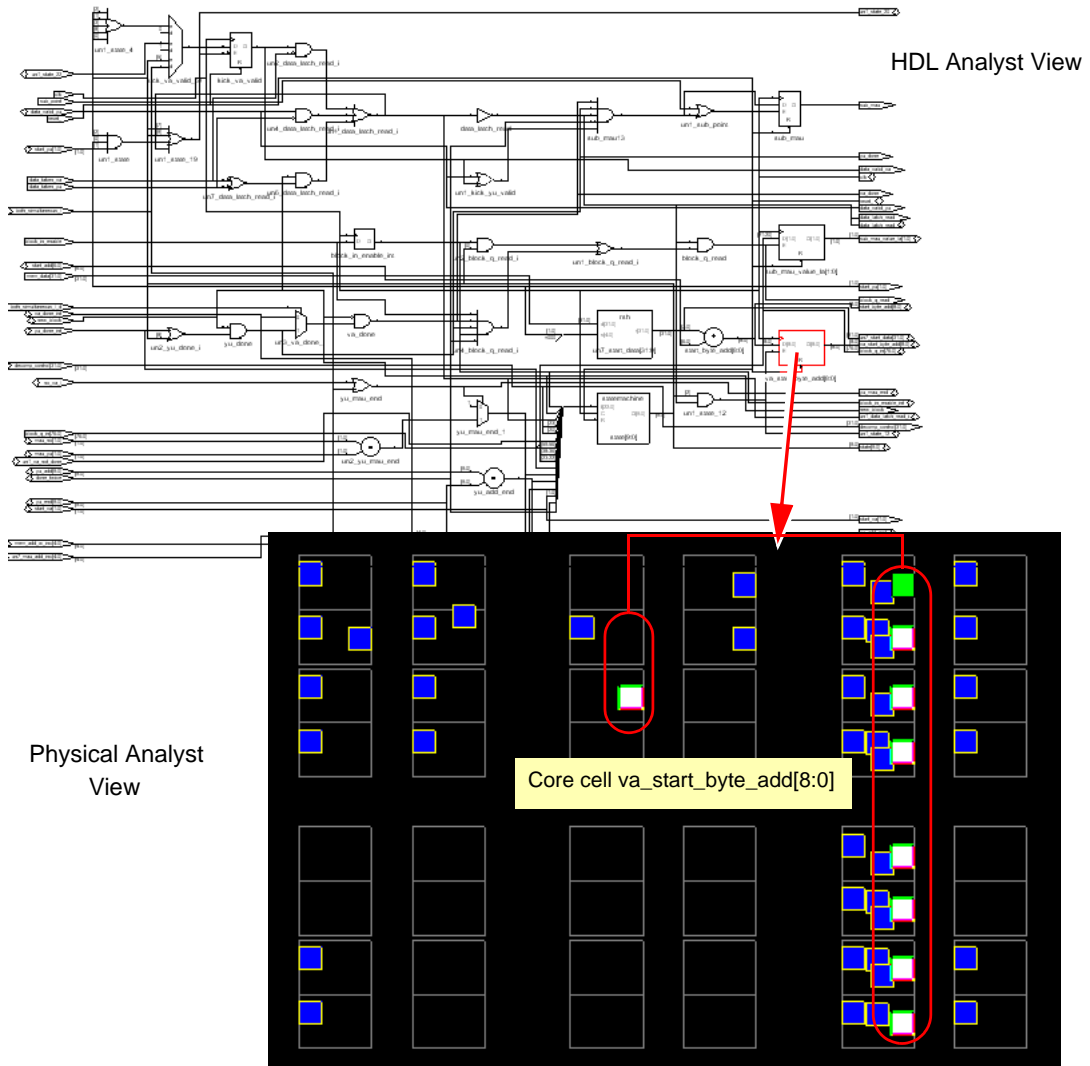
5. If needed, use the Reset filter icon to redisplay the unfiltered objects, if you filtered the view in step 3.

## Crossprobing from the RTL View to Physical Analyst

Follow this procedure to crossprobe from the RTL view to the Physical Analyst view.

1. Open the Physical Analyst view.
2. Enable the View->Cross Probing->Cross Probing from RTL Analyst option.
3. Click the object (instance or macro) in the RTL view to highlight and crossprobe it. You can use the schematic view, hierarchy browser, or the Object Query dialog box to select the object.

You can cross probe hierarchical objects in the RTL view to the set of objects for which the hierarchy is synthesized in the Physical Analyst view. You cannot cross probe primitives in the RTL view which do not have a counterpart in the mapped netlist. The tool highlights all objects relating to the RTL object. For example, if you selected a module, all mapped objects with physical information that implement the module in the Physical Analyst view are highlighted.



## Crossprobing from the Technology View to Physical Analyst

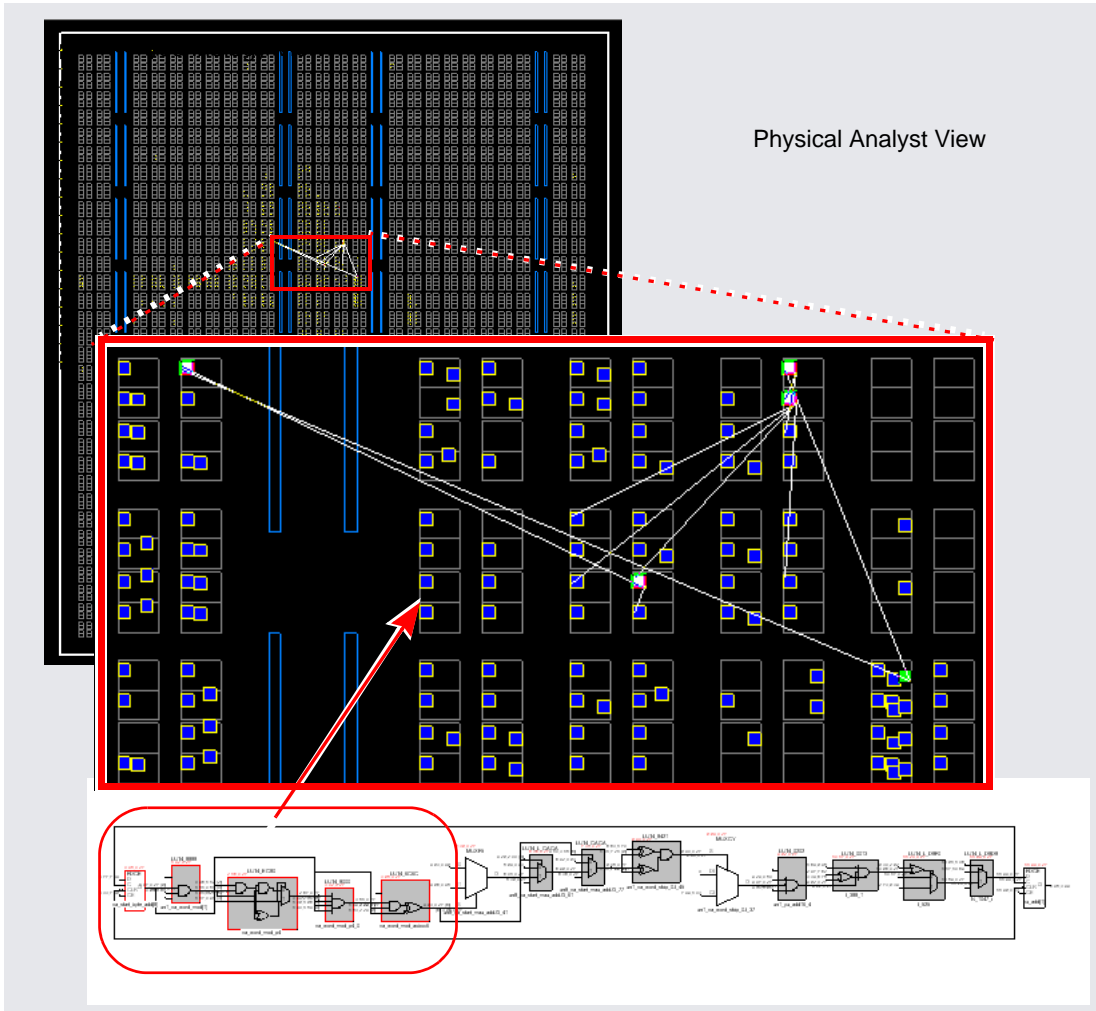
Follow this procedure to crossprobe from the RTL view to the Physical Analyst view.

1. Open the Physical Analyst view.
2. Enable the View->Cross Probing->Cross Probing from Tech Analyst option.
3. Click the object in the Technology view to highlight and crossprobe. You can use the schematic view, hierarchy browser, or the Object Query dialog box to select the object.

When you select an instance in the Technology view, the placed primitive that corresponds to that instance in the Physical Analyst is highlighted.

4. To automatically route cross-probed instances, enable the View->Cross Probing->Auto route cross probe insts option (the option is enabled by default).

The following shows cross probing from the Technology view to the Physical Analyst view when this option is enabled.



# Analyzing Netlists in Physical Analyst


In the Physical Analyst view, there are a number of commands for tracing logic and analyzing the netlist, which can be accessed from the right-click popup menus. These commands are context-sensitive, depending on the selected object and where you click. See [Chapter 3, \*User Interface Commands\*](#) of the *Reference Manual* for a complete list of View menu and Physical Analyst popup menu commands.

See the following for details:


- [Filtering the Physical Analyst View](#), on page 721
- [Expanding Pin and Net Logic in Physical Analyst](#), on page 722
- [Expanding and Viewing Connections in Physical Analyst](#), on page 727

## Filtering the Physical Analyst View

Filtering is a useful first step in analysis, because you can focus on the relevant parts of the design. Some commands, like the Expand Paths commands, automatically generate filtered views. This procedure only discusses manual filtering, where you use the Filter command to isolate selected objects.

1. Select the objects that you want to isolate.
2. Select the filter command, using one of these methods:
  - Select Filter->Show Selected from the Physical Analyst View menu or from the right-click popup menu.
  - Click the Filter Schematics icon ().
  - Press Alt and draw a narrow V-shaped mouse stroke in the schematic window. See Help->Mouse Stroke Tutor for an illustration.

The software filters the design and displays the selected objects in a filtered view. You can now analyze the objects and perform operations like tracing paths, building up logic, filtering further, finding objects, hiding objects, or crossprobing.

3. To return to the previous view, click the Back () icon.

## Expanding Pin and Net Logic in Physical Analyst

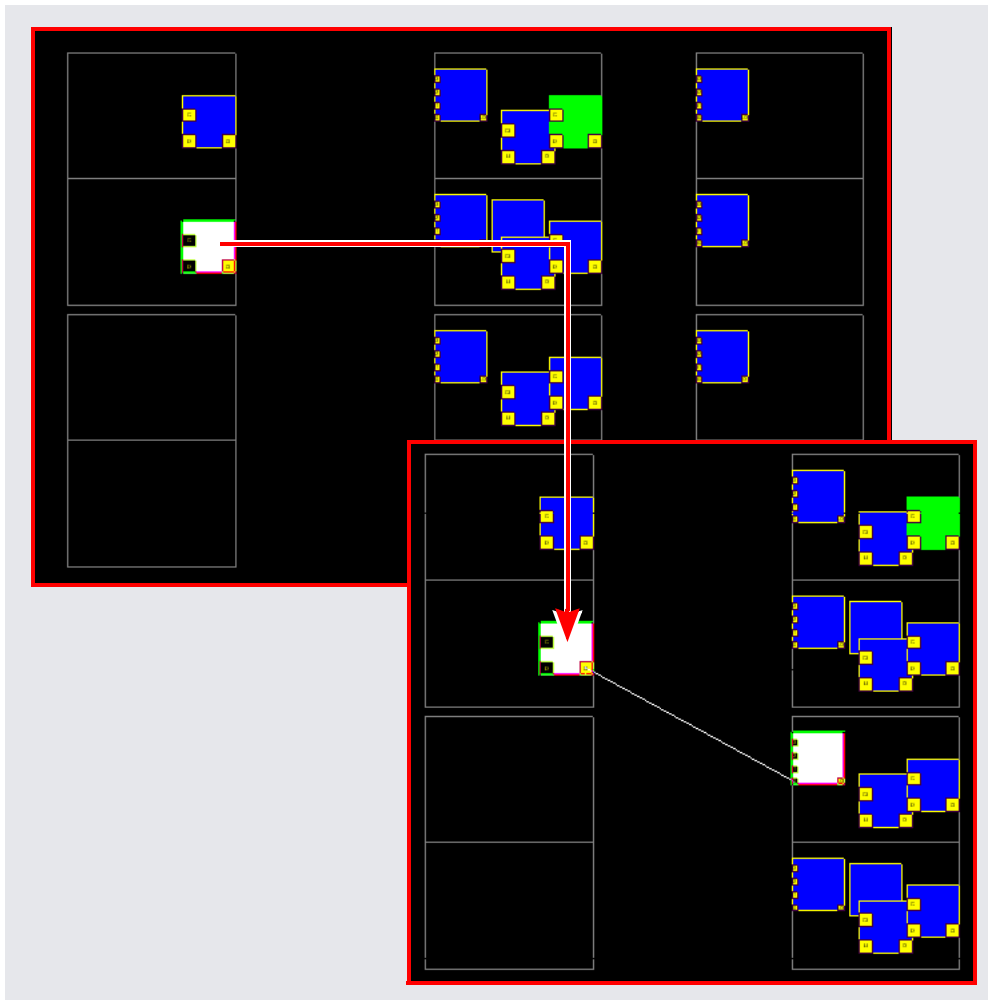
When you are working in a filtered view, you might need to include more logic in your selected set to analyze your design. This section describes commands that expand logic fanning out from pins or nets; to expand paths, see [Expanding and Viewing Connections in Physical Analyst, on page 727](#).

Use the Expand commands with the Filter and Nets->Visible commands to isolate and connect the logic that you want to examine.

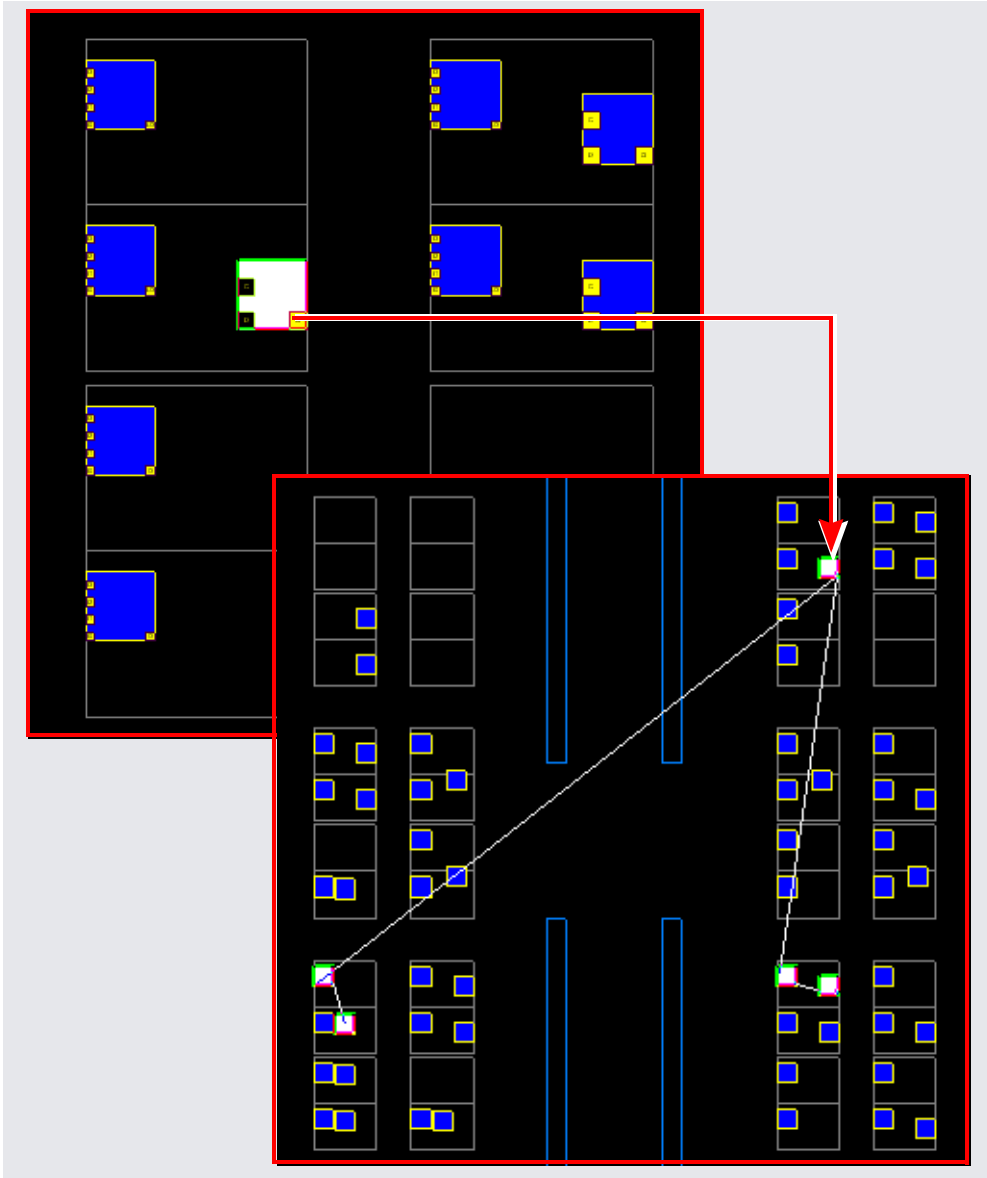
1. To expand logic from a pin, do the following in the Physical Analyst view.

| To view..                                                           | Do this..                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| All cells connected to a pin                                        | <ul style="list-style-type: none"><li>• Select a pin on the cell instance.</li><li>• Right-click and select Expand-&gt;Selected Pins. See <a href="#">Expanding Logic Example, on page 723</a>.</li></ul> <p>If you change your selection to all output pins, all input pins, or all pins, you can use the same command to expand from these points.</p>                                                          |
| All cells that are connected to a pin, up to the next register/port | <ul style="list-style-type: none"><li>• Select a pin on the cell instance.</li><li>• Right-click and select Expand to Register/Port-&gt;Selected Pins. See <a href="#">Expanding Logic to Register/Port Example, on page 724</a>.</li></ul> <p>If you change your selection to all output pins, all input pins, or all pins, you can use the same command to expand to registers and ports from these points.</p> |

## Expanding Logic Example



### Expanding Logic to Register/Port Example

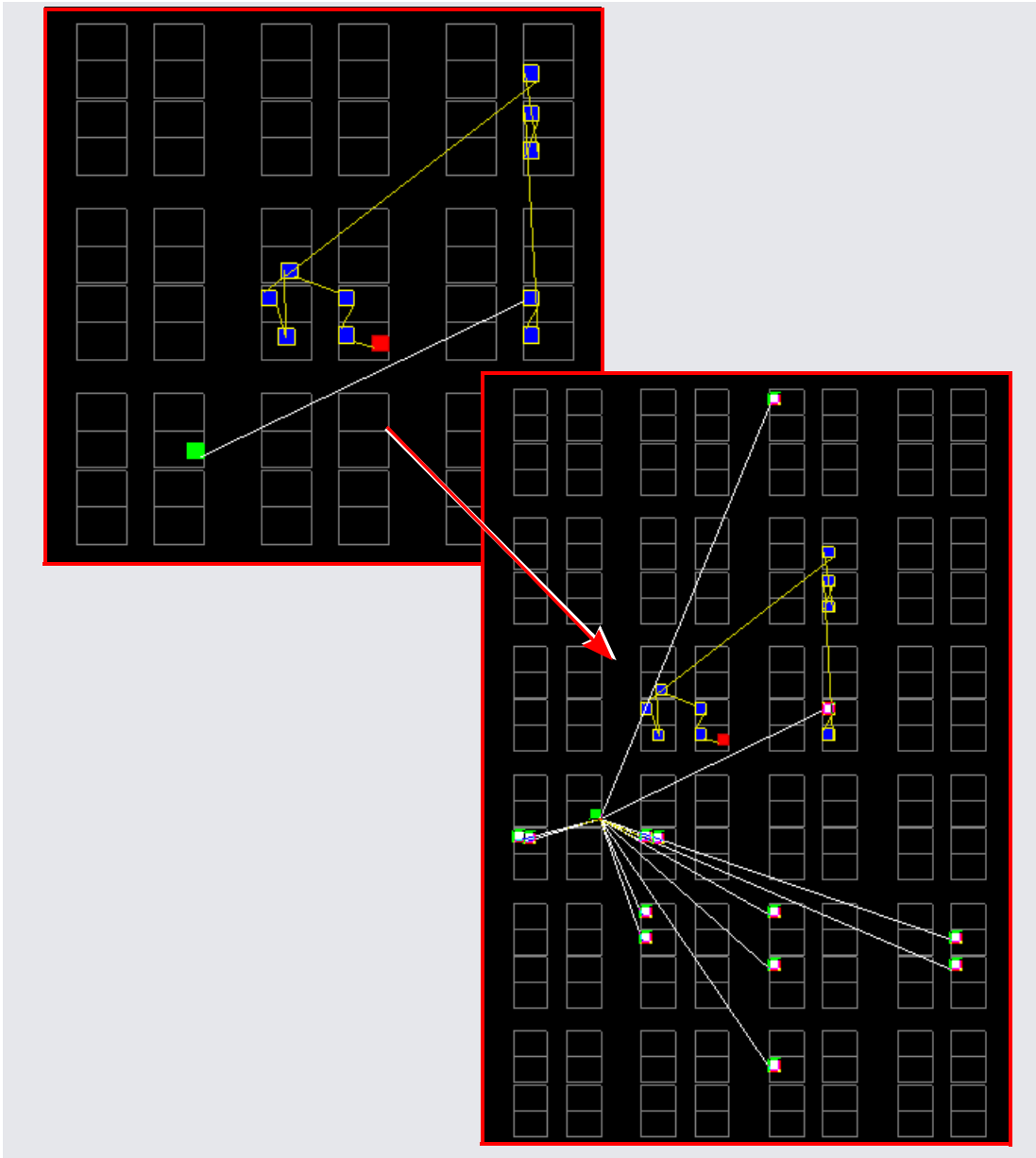




2. To expand logic from a net, use the commands shown in the following table.

| <b>To...</b>                             | <b>Do this...</b>                                                                                                                                                                                                                                                                              |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Select all instances on a net            | Select a net and select <b>Select Net Instances-&gt;All Pins</b> . The software shows an unfiltered view that includes all the instances connected to the net along the signal path.<br>You can also choose to show output pins or input pins with this command.<br>See the following example. |
| Highlight all visible instances on a net | Select a net and select <b>Highlight Visible Net Instances-&gt;All Pins</b> . You see a filtered view of all instances connected to the selected net along the signal path.<br>You can also select to show output pins or input pins.                                                          |
| Select the net driver                    | Select a net and select <b>Select Net Driver</b> . The software shows an unfiltered view that includes the driver of the net.                                                                                                                                                                  |
| Go to the net driver                     | Select a net and select <b>Go to Net Driver</b> . The software shows and scrolls to the driver of the net.                                                                                                                                                                                     |

This example shows instances on a critical path. First, the critical path was filtered. Then one of the nets on the critical path selected and the Select Net Instances->All Pins selected. The figure shows the results.



## Expanding and Viewing Connections in Physical Analyst

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 in Physical Analyst, on page 722](#). 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 with Physical Analyst, on page 750](#).

Use the following path commands with the Filter and Nets->Visible commands to isolate and connect the logic that you want to examine.

To expand and view connections between selected objects, do the following:

1. Select two or more objects.
2. To expand the logic, select Expand Paths->All Pins from the popup menu. Alternatively, you can select to expand from selected pins.



**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 17

# 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 730
- [Using the Stand-alone Timing Analyst](#), on page 736
- [Using the Island Timing Analyst](#), on page 743
- [Analyzing Timing with Physical Analyst](#), on page 750
- [Handling Negative Slack](#), on page 756

## Analyzing Timing in Schematic Views

You can use the Timing Analyst and HDL Analyst functionality to analyze timing. This section describes the following:

- [Viewing Timing Information](#), on page 730
- [Annotating Timing Information in the Schematic Views](#), on page 731
- [Analyzing Clock Trees in the RTL View](#), on page 733
- [Viewing Critical Paths](#), on page 733
- [Using the Stand-alone Timing Analyst](#), on page 736
- [Handling Negative Slack](#), on page 756
- [Using the Island Timing Analyst](#), on page 743 (Synplify Premier)

### 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.

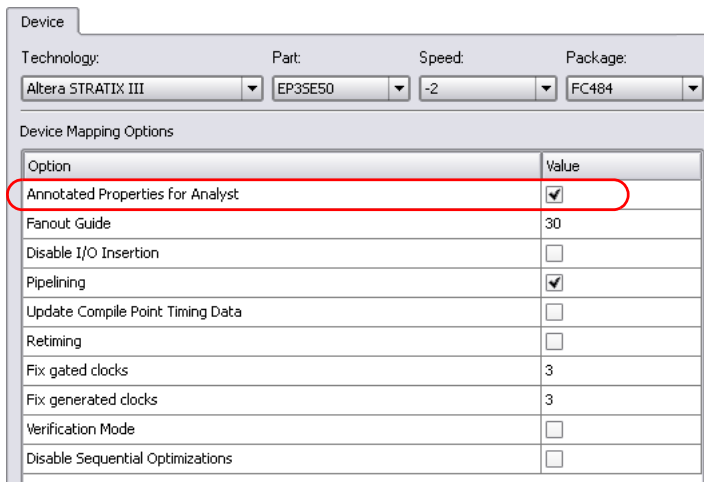
This displays the timing numbers for all instances in a Technology view. It shows the following:

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Delay</b>      | This is the first number displayed. <ul style="list-style-type: none"><li>• Combinational logic<br/>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<br/>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.

1. In the Device panel 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.

---

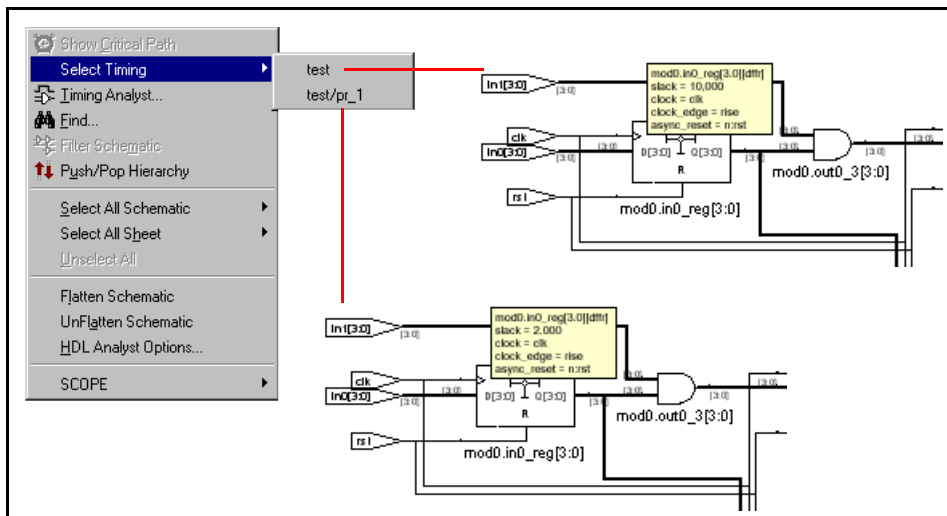
- .tap Timing Annotated Properties  
Contains the annotated timing properties generated after compilation.

---

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 {@<prop\_name>=<prop\_value>} command for the searches. See [Find Filter Properties, on page 1266](#) in the *Reference Manual* for a list of properties. For information about the find command, see [Tcl find Command, on page 1260](#) in the *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 658](#), [Expanding Pin and Net Logic, on page 660](#) and [Expanding and Viewing Connections, on page 664](#).


3. Check that your defined clock constraints cover the objects in the design.

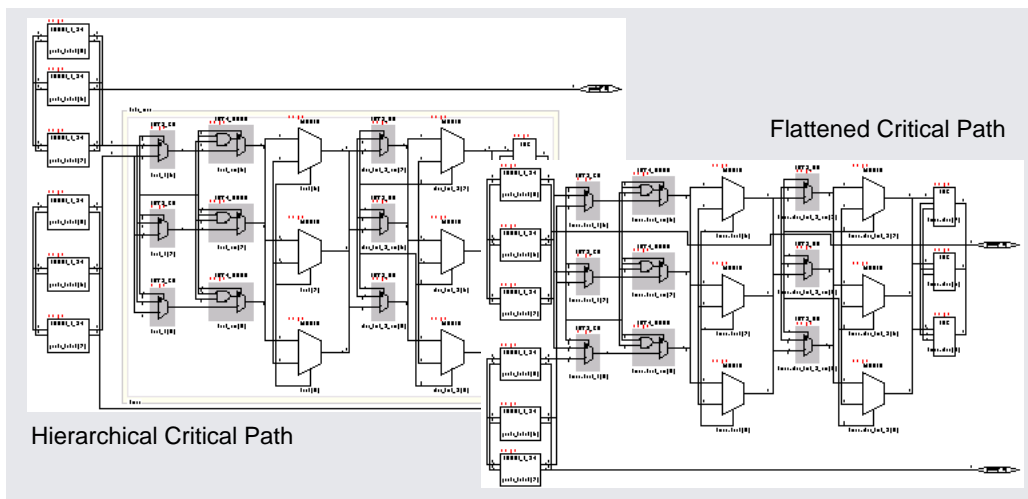
If you do not define your clock constraints accurately, you might not get the best possible synthesis optimizations.

## 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 [Using the Stand-alone Timing Analyst, on page 736](#) 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 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.



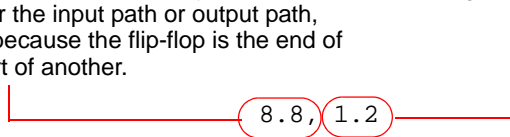
3. Use the timing numbers displayed above each instance to analyze the path. If no numbers are displayed, enable HDL Analyst->Show Timing Information. Interpret the numbers as follows:

**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.



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 756](#).

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.

## Using the Stand-alone Timing Analyst

The timing report in the log file provides default timing information. Use the stand-alone timing analyzer for your more specific requirements. You can run the stand-alone timing analyzer to customize a timing report (.ta) for the following types of information:

- More details on a specific path
- Results for paths other than the top five timing paths (default)
- Modifications to constraints for stand-alone timing analysis, without rerunning synthesis

1. Select Analysis->Timing Analyst or click on the Timing Analyst icon (🔍📄).

**Timing Report Generation**

Filters

From:

Through:

To:

Generate Asynchronous Clock Report

Limit Number of Critical Start/End Points To:

Limit Number of Paths To:

Enable Slack Margin (ns):

Open Report  Open Schematic

OK Cancel Generate

Output Files

Async Clock Report File:

TA File:  ...

SRM File:  ...

Constraint Files

| Sel | File |
|-----|------|
|     |      |

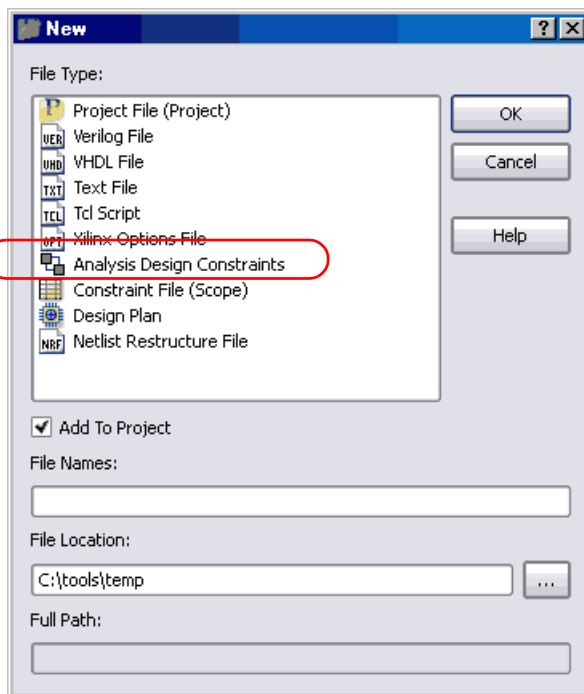
2. Fill in the parameters as appropriate.

You can type in the from/to or through points on the Timing Report Generation dialog box, or you can cut-and-paste or drag-and-drop valid objects from the Technology view into the fields. However, it is recommended that you open the Technology view, whenever you cut-and-paste or drag-and-drop objects.

See *Timing Report Generation Parameters*, on page 218 in the *Reference Manual* for details on timing analysis parameters and how they can be filtered.

3. You can modify constraints to determine the effect on timing. To do this you need a constraint file to use with the stand-alone Timing Analyzer.

- Select File->New and click on Analysis Constraint File.
- Fill in the constraint file name.



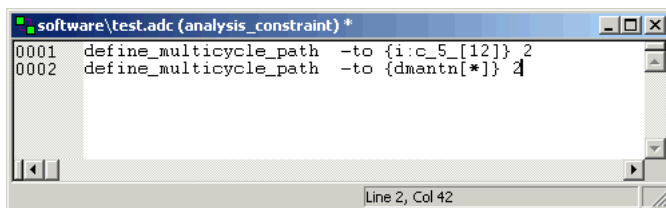
The tool automatically assigns the .adc extension to the filename.

- Make sure Add to Project is enabled and click OK.

This opens the text editor for you to create the new constraints.

- Enter any additional constraints that you want to apply for timing. Keep in mind that the original .sdc 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. See [Conflict Resolution for Timing Exceptions, on page 451](#) for information on how the tool prioritizes timing exceptions.

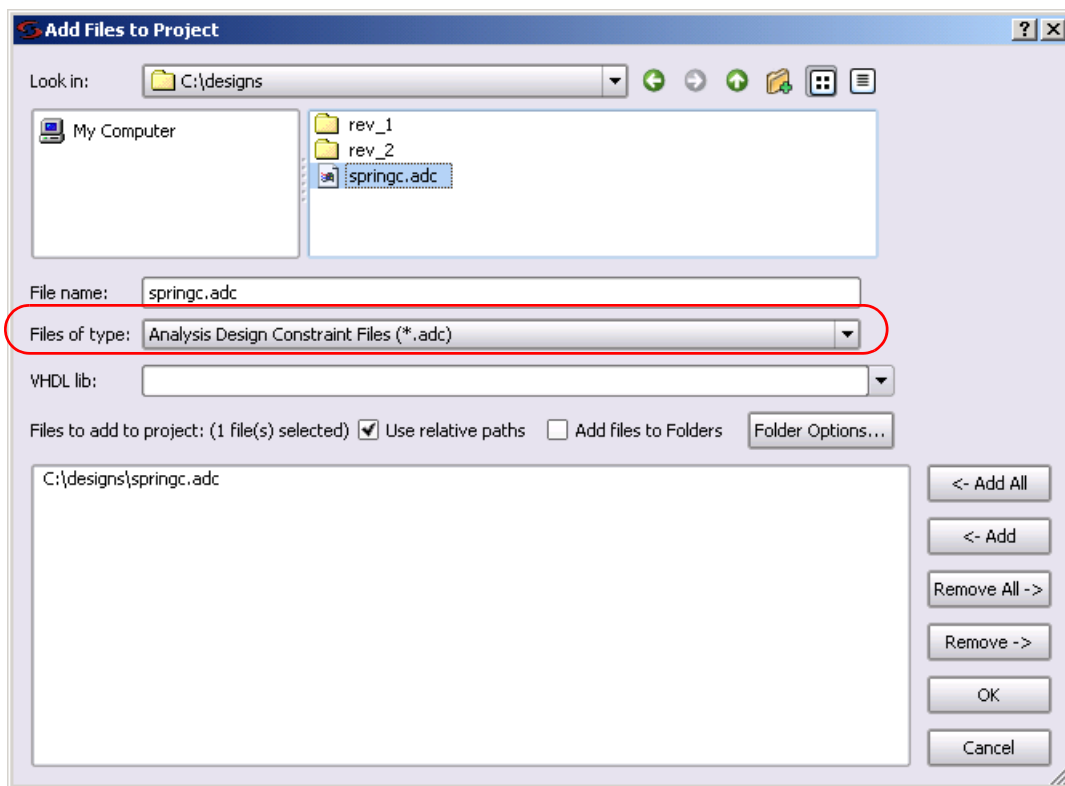
For more information about applying .adc constraints, see [Entering Constraints into the .adc File, on page 741](#).



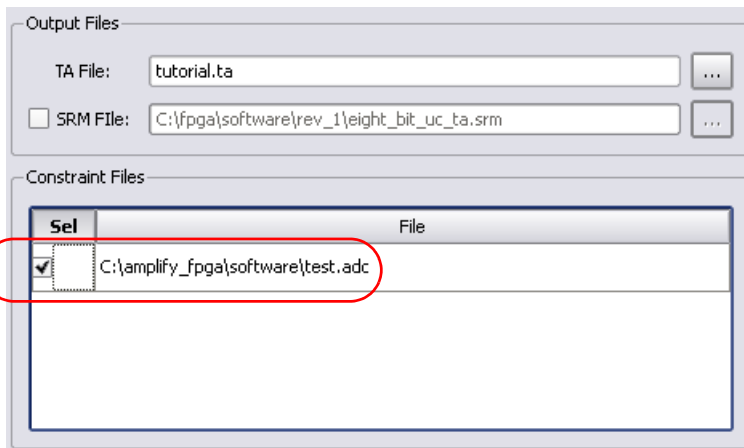
The screenshot shows a text editor window titled "software\test.adc (analysis\_constraint) \*". The window contains two lines of code: "0001 define\_multicycle\_path -to {i:c\_5\_[12]} 2" and "0002 define\_multicycle\_path -to {d:mantn[\*]} 4". The status bar at the bottom indicates "Line 2, Col 42".

- When you are done, save and close the file. This adds the file to your project.

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.



- Enable the appropriate .adc constraint file on the Timing Report Generation dialog box.



4. Click OK to set the selected options.
5. Click Generate to run the report (or select Analysis->Timing).

If the design was run in Physical Synthesis mode, make sure that the Physical Synthesis switch is still enabled when you run stand alone timing analysis to ensure proper results.

The software generates a custom report file called *project\_name.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.

6. Analyze results.
  - View the report (Open Report) in the Text Editor. The following figure shows 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: 1:swmult2o[11]
Constraint File(s):

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: smult2o[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 Type Pin Pin Delay Arrival No. of
Name Type Name Dir Delay Time Fan Out(s)

smult2o[11] stratixii_lcell_ff regout Out 0.094 0.094 -
smult2o[11] Net - - 0.311 - 2
smultl_carry_9 stratixii_lcell_comb dataf 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.

## Entering Constraints into the .adc File

Constraints and collections applied in the .sdc file reference the RTL-level database. Synthesis optimizations such as retiming and replication can change object names during mapping. This is when RTL-level constraints are properly translated and applied to the mapped objects.

However, the standalone timing analyst does not map objects. It simply reads the gate-level object names from post mapping; 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.

### Example

Suppose register en\_reg is replicated during mapping to reduce fanout. Also, let's assume that both registers en\_reg and en\_reg\_rep2 connect to register dataout[31:0].

If you define the following false path constraint in the .adc file:

```
define_false_path -from {{i:en_reg}} -to {{i:dataout[31:0]}}
```

the standalone timing analyzer does not automatically treat paths from the replicated register `en_reg_rep2` as false paths. Unlike constraints in the .sdc 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:

```
define_false_path -from {{i:en_reg}} -to {{i:dataout[31:0]}}
define_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)}
define_false_path -from {{$en_regs}} -to {{i:dataout[31:0]}}
```

## Using the Island Timing Analyst

In the Synplify Premier tool, after you synthesize a design, you can generate a timing report that contains a hierarchical display for groups of connected critical paths called islands. The island timing report is useful for creating a design plan and analyzing critical paths (routing vs. logic delay), because it identifies the instances or pins belonging to multiple paths and how the critical paths in an island group are connected. Critical paths with a large percentage of total route delay typically have better improvements with design planning.

See the following for more information:


- [Working in the Island Timing Analyst Interface](#), on page 743
- [Generating the Island Timing Report Automatically](#), on page 745
- [Generating the Island Timing Report Interactively](#), on page 747
- [Defining Group Range and Global Range for Island Timing](#), on page 748
- [Viewing the Island Timing Report](#), on page 749

For details on how to interpret information in this report, see [Synplify Premier Island Timing Report](#), on page 324.

The island timing report can be generated for the following technologies:

- Xilinx – Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, and Spartan-3
- Altera – Stratix, Stratix GX, Stratix II, Stratix II GX, Stratix III, Stratix IV, Cyclone, and Cyclone II

### Working in the Island Timing Analyst Interface

1. Open the Island Timing interface by clicking on the Control Panel icon ().

The Island Timing interface opens.

Islands/Paths Summary Report

Islands/Paths Control Panel                      Islands/Paths Details Report

- Set parameters and generate an island timing report, as described in [Generating the Island Timing Report Automatically, on page 745](#) and [Generating the Island Timing Report Interactively, on page 747](#).

The summary report opens.

- The following table shows you how to organize and work with the summary data:

#### Sort Columns

Click on the column header to toggle between ascending and descending order. When sorting several columns, the most recent column clicked will be the most significant column and the first column clicked becomes the least significant column in the summary display. Islands are sorted separately, unless you choose to display all paths at a flat level.

#### Reorder the columns

Drag and drop from the column header to the new location.

---


|                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Select Islands or Critical Paths | <p>To select islands or critical paths, use the left-mouse button.</p> <p>To select multiple islands or paths use either the Shift or Ctrl key with the left-mouse button.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Crossprobe to the HDL Analyst    | <p>To crossprobe to the HDL Analyst view, do the following:</p> <ul style="list-style-type: none"><li>• Make sure to open the HDL Analyst flattened Technology view first. If you are going use the RTL view, make sure open the Technology view also.</li><li>• Then select islands or critical paths from the Islands/Paths Summary display.</li><li>• Click on the Cross Probe button.</li><li>• You can then filter critical paths in the HDL Analyst view.</li><li>• Use crossprobing from the HDL Analyst view, to see timing data in the Technology view.</li></ul> <p>Note that when you choose to group by islands, simply select the island to crossprobe the entire island which includes all its paths in the HDL Analyst view.</p> |

---

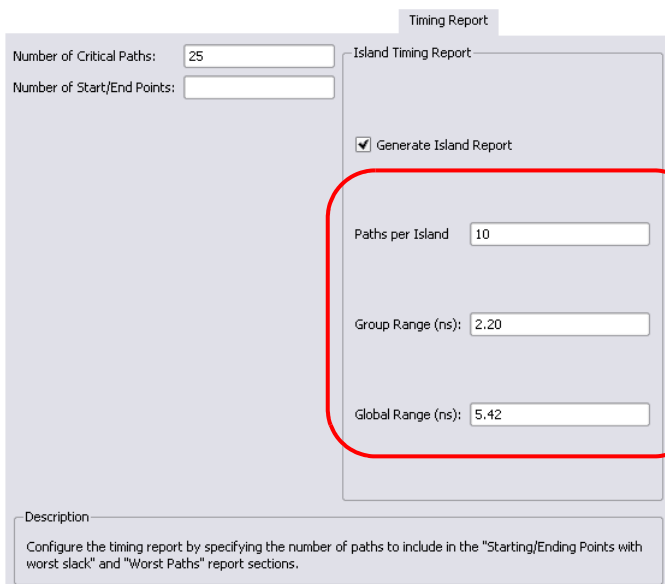
## Generating the Island Timing Report Automatically

Use the following process to automatically generate the hierarchical-bas island timing report during the mapper phase.

Use the Islands/Paths control panel to generate an island timing report.

1. Open the Island Timing interface by clicking on the Control Panel icon ().
2. Set the following parameters either manually in their appropriate parameter fields, or by using the slider controls on the control panel.
  - Paths per Island  
Specify the number of paths to report for each island.
  - Global Range (ns)  
Specify a global range from the worst case slack of the design to determine the number of islands displayed in the timing report. Only the islands above this global slack range (water level) are displayed in the island report. See [Defining Group Range and Global Range for Island Timing](#), on page 748.

- **Group Range (ns)**  
Specify a group range from the worst case slack of the island to determine the critical paths for each island. Only the paths within this group slack range for an island are displayed in the island report. See [Defining Group Range and Global Range for Island Timing, on page 748](#).
- **Max Paths/Island**  
Specify the number of paths to report for each island. Type the number of paths in the parameter field or use the up/down scroll option located on the right side of the parameter box. If the number of paths which do not meet the specified slack for an island exceeds the maximum number of paths specified, then the most critical paths within this limit are displayed in the island timing report.



Timing Report

Number of Critical Paths: 25

Number of Start/End Points:

Island Timing Report

Generate Island Report

Paths per Island: 10

Group Range (ns): 2.20

Global Range (ns): 5.42

Description

Configure the timing report by specifying the number of paths to include in the "Starting/Ending Points with worst slack" and "Worst Paths" report sections.

### 3. Click the Generate Report button.

You can interactively change the values used to generate the timing report. These changes are reflected in the Island Timing Analyst tool immediately after you click on the Generate Report button.

For more details about the Island Timing Report, see [Using the Island Timing Analyst, on page 743](#).

4. After you have set all the implementation option settings, click OK and close the dialog box.
5. When you are ready to synthesize your design, select Run->Synthesize in the Project view or simply click on the Run button.

After synthesis completes, the log file (.srr) displays the following message:


```
@N|Hierarchical island-based critical path report is located in
C:\path_directory\design_name.tah
```

The timing report file (.tah) is listed in the Implementation Results view of your project.

See [Viewing the Island Timing Report, on page 749](#) for further information.

## Generating the Island Timing Report Interactively

Use the following process to interactively generate the island report from the Island Timing Analyst.

1. Select the Timing Report tab of the Implementation Options panel and in the Island Timing Report section of this pane, check that the Generate Island Report switch is disabled. Otherwise, the Island Timing Analyst will display the report generated with the values used from this pane.
2. Then synthesize your design by either selecting Run->Synthesize in the Project view or simply clicking on the Run button.
3. Invoke the Island Timing Analyst by either clicking on the Island Timing Analyst icon () or selecting HDL Analyst->Island Timing Analyst from the menu.
4. Then you must specify values for group range, global range, and maximum paths per islands from the Islands/Paths Control panel. See [Defining Group Range and Global Range for Island Timing, on page 748](#).
5. Click on the Generate Report button from the Islands/Paths Control panel.

See [Viewing the Island Timing Report, on page 749](#) for more detailed information.

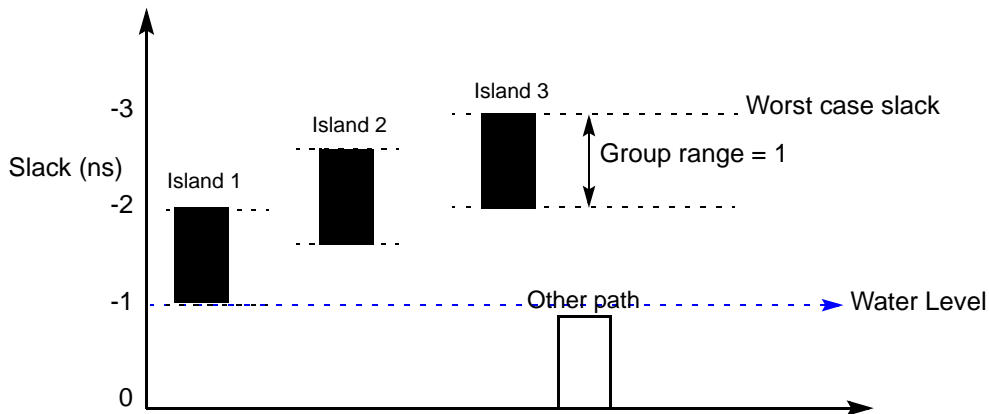
## Defining Group Range and Global Range for Island Timing

Global Range specifies the lower bound (or water level) for the timing report. The Group Range value specifies the range from the worst case slack, and thus determines the instances that are reported for each individual island.

The following table shows how different settings affect what is reported:

|                  |       |                                                                                                                                                                                                                                                                                                 |
|------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Worst Case Slack | -3 ns |                                                                                                                                                                                                                                                                                                 |
| Global Range     | 2 ns  | As the worst case slack is -3 ns, setting the global range to 2 causes the water level to be -1 ns ( $-3 + 2$ ). The island report will not contain instances with a slack that exceeds (is more positive) than -1.                                                                             |
| Group Range      | 1 ns  | This specifies a range from the worst case slack for an island; the software reports all island instances that fall within this range. If the worst-case slack for an island is -3 ns, the report for that island will contain instances with slack in the range of -3ns to -2 ns ( $-3 + 1$ ). |

The following graphically shows how the island report lists all islands in the design that fall within the range from -3 to -1 (global range). For each island, it reports instances whose slack is within 1 ns of the worst-case slack for that island (group range).





## Viewing the Island Timing Report

The timing report lists the islands with their worst paths displayed based on the number of paths you requested for the island. For each path, the logic elements and nets and net delays are displayed in an ordered from-to list.

1. To view the hierarchical-based island timing report file, you must first select Options->Project View Options and enable the Show all files in results directory check box.
2. To view the timing report, either
  - Double-click the .tah file.
  - Select the .tah file, right-click and select Open as Text from the popup menu.

```

1 #####
2 #
3 # Island is a group of instances connected by critical paths.
4 # Global range is a slack range for the instances that appear in the islands from the worst slack
5 # Group range is a slack range for the instances in an island from the worst slack of that island
6 #
7 #####
8
9 Number of islands = 2
10 Island global range = 5.42 ns
11 Island group range = 2.20 ns
12 Island max number of paths = 10
13
14 ##### START OF ISLAND 1 #####
15
16 Worst slack = 0.234ns Number of mapped instances = 411
17
18 RTL INSTANCES:
19
20 Start points: dmux.alubtmp[7:0] alua_tmp2[7:0] decode.decodes[13:0] special_regs.inst[11:0] rom.Data_1[11:0] decode.opcode_call decode.opcode_retlw prgmcntr.c_stacklevel[6:
21
22 End points: uc_alu.aluz dmux.alubtmp[7:0] uc_alu.aluout[7:0] special_regs.inst[11:0] prgmcntr
23 rom.Data_1[11:0] dmux.alua[7:0] special_regs.port_int_b[7:0]
24
25 MAPPED INSTANCES:
26
27 Start points Slack
28 -----
29 dmux.alubtmp_fast[0] 0.234
30 alua_tmp2[1] 0.339
31 alua_tmp2[2] 0.342
32 dmux.alubtmp[1] 0.439
33 alua_tmp2[0] 0.471
34 dmux.alubtmp[2]

```

3. To find information in the timing file, select Edit -> Find or press Ctrl-f. Fill out the criteria in the form and click OK.

To view the island timing report interactively from the Island Timing Analyst tool, see the Summary and Detail views in the Island Timing. See [Assigning Critical Paths from Island Timing to a Region](#), on page 516 for details about how to use the critical path timing information in this file or the tool for QoR improvements with physical synthesis.

# Analyzing Timing with Physical Analyst

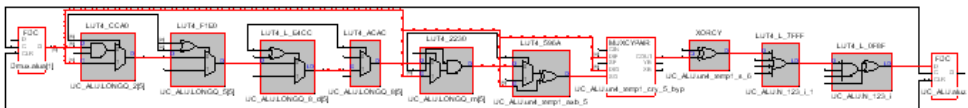
You can use the Physical Analyst functionality to analyze timing. This section describes how to view and use the critical path for further physical synthesis.

- [Viewing Critical Paths in Physical Analyst](#), on page 750
- [Tracing Critical Paths Forward in Physical Analyst](#), on page 753
- [Tracing Critical Paths Backward in Physical Analyst](#), on page 755

## Viewing Critical Paths in Physical Analyst

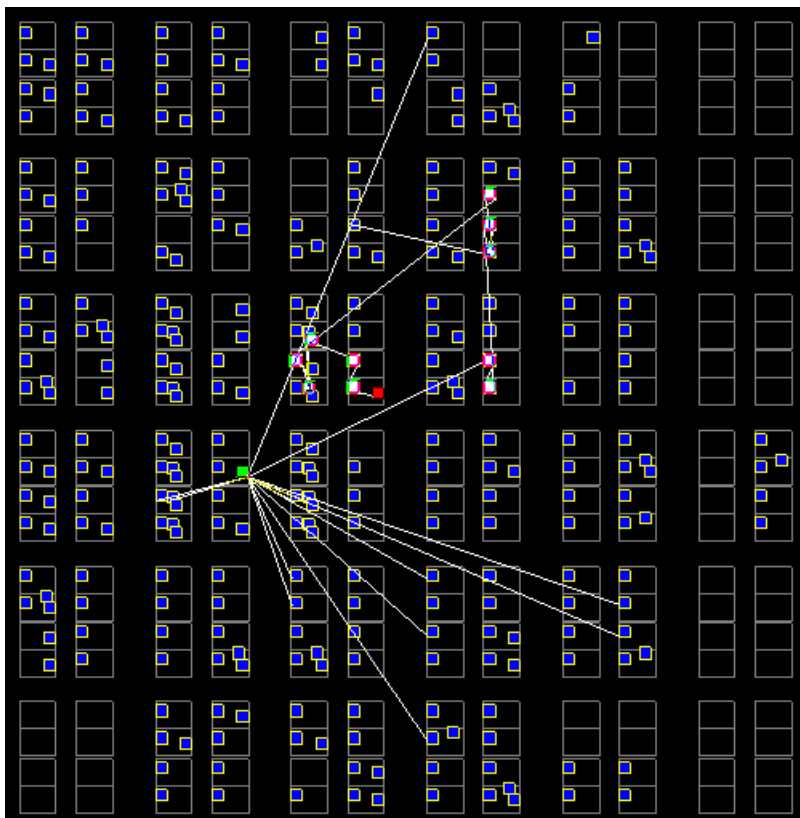
The Physical Analyst tool makes it easy to find and examine critical paths and the relevant logic in the HDL Analyst schematic view. Make sure the HDL Analyst view is open, for example, by selecting HDL Analyst->Technology->Flattened View or HDL Analyst->RTL->Flattened View. The following procedure shows you how to filter and analyze a critical path.


1. To generate a view of the critical path with the Physical Analyst tool, click the Show Critical Path icon (stopwatch icon (🕒)) or select the command from the popup menu. To zoom in on the critical path, right-click and select Zoom Selected from the popup menu.
2. Check the Technology view.

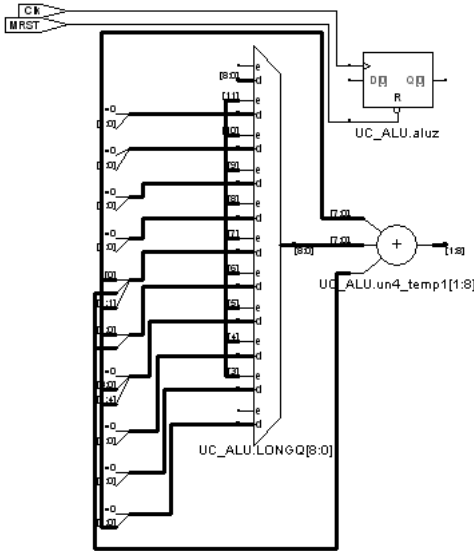


You can also cross probe the critical path from the flattened Technology view to the Physical Analyst view by clicking on the Show Critical Path icon (🕒). Then, right-click and select Select All Schematic->Instances. Make sure the Physical Analyst view is open.

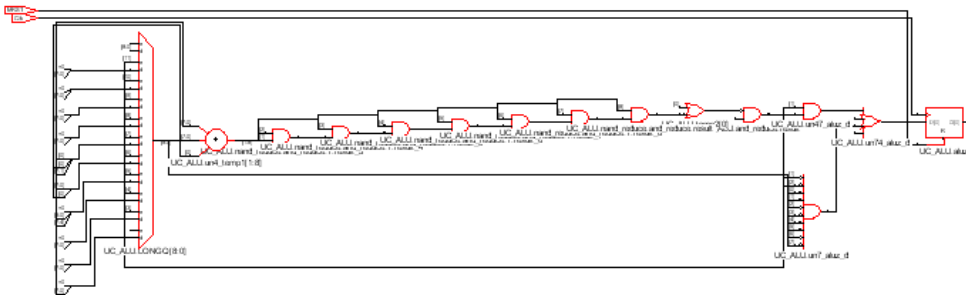
3. Check the Physical Analyst view. Critical path instances and nets should be highlighted in this view.



4. In the HDL Analyst view that is already open, click on the Filter Schematics icon (). Only the instances and nets belonging to the critical timing path are displayed, as shown below.



- In the HDL Analyst view, right-click and select Expand Paths from the popup menu. Then, you can drag-and-drop this logic into a region on the device design plan (.sfp) file for further physical synthesis.



## Tracing Critical Paths Forward in Physical Analyst

The following procedure shows you how to trace a critical path forwards starting from the instance containing the critical start point.

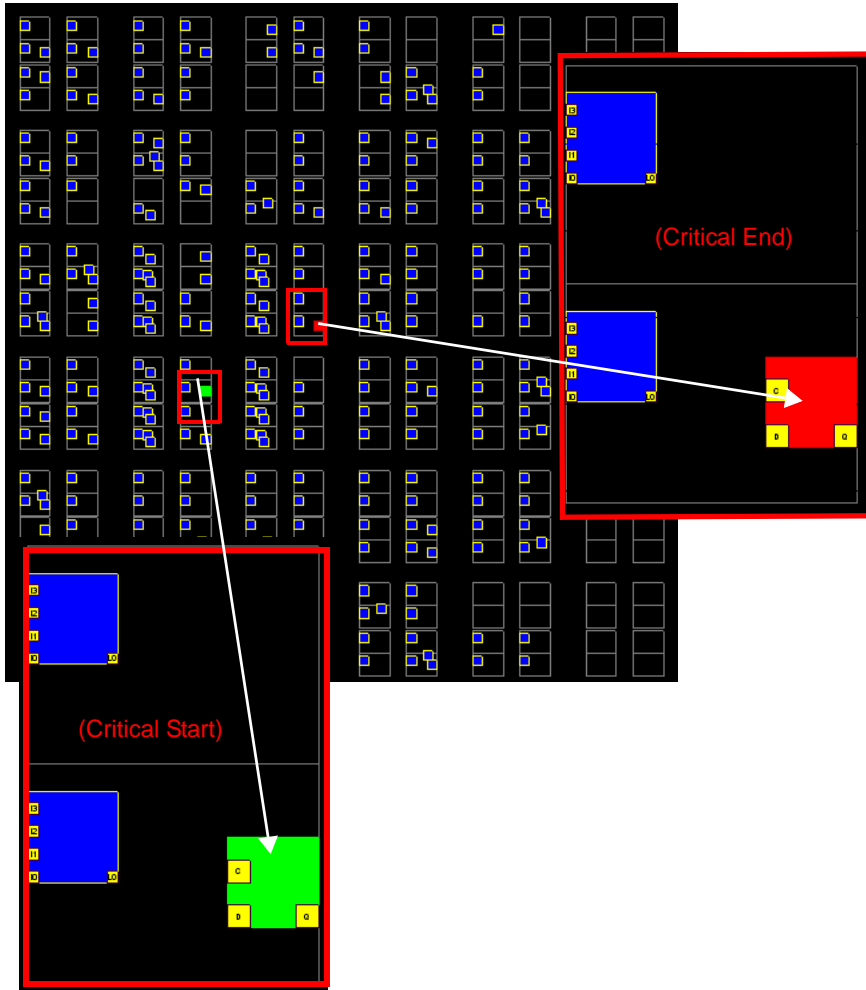
1. Do one of the following in the Physical Analyst view:
  - Right-click and select Critical Path->Expand Path Forward from the popup menu
  - Press F3.

The instance containing the critical path start point is displayed in green and highlighted. Move the cursor over the instance to display a tool tip that specifies its name and identifies this as the critical start point.

You can also use the Filter Search option of the Find command to locate the Critical path start point.

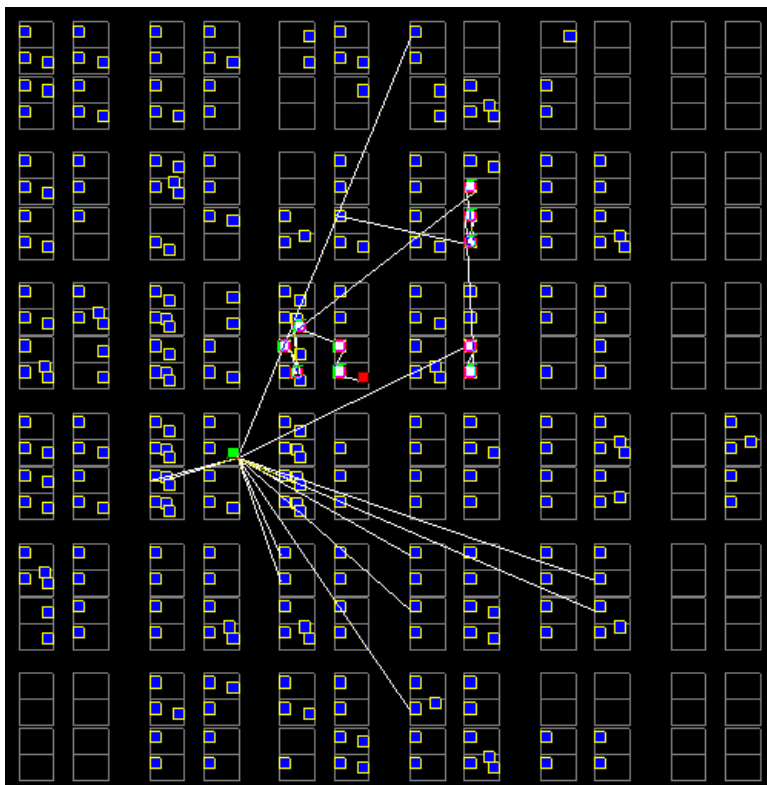
2. Select Critical Path->Expand Path Forward or press F3 again.

The next instances on the critical path and input ports that feed into the path are displayed and highlighted and shown connected to the critical path start point.



3. Repeat the previous step to continue to trace the path to the next instance in the path. Continue until you reach the end point.

The following figure shows you how the critical path is finally displayed.



## Tracing Critical Paths Backward in Physical Analyst

The following procedure shows you how to trace a critical path backward. See the figures in [Tracing Critical Paths Forward in Physical Analyst, on page 753](#), which also apply to this procedure.

1. Do one of the following in the Physical Analyst view:
  - Right-click and select Critical Path->Expand Path Backward from the popup menu.
  - Press Shift+F3.

The instance containing the critical path end point is displayed and highlighted. Move the cursor over the instance to display a tool tip that specifies its name and identifies this as the critical end point.

You can also use the Filter Search option of the Find command to locate the Critical path end point. The cell location of the critical path end point is displayed in red in the Physical Analyst view.

2. Use one of the methods described in the previous step to continue to trace the net to the next instance in its path.

The next instance containing the critical path and output ports that feed into the path are displayed and highlighted and shown connected to the critical path end point.

3. Repeat the previous step until you reach the start point. See the figure in step 3 of [Tracing Critical Paths Forward in Physical Analyst, on page 753](#) for an example of how the critical path is displayed.

## 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%.

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 660](#) and [Expanding and Viewing Connections, on page 664](#). For more information on filtering schematics, see [Filtering Schematics, on page 658](#).
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% 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 429](#)), retiming ([Retiming, on page 433](#)), FSM exploration ([Running the FSM Explorer, on page 458](#)), or resource sharing ([Sharing Resources, on page 450](#)). The Synplify product does not support pipelining, retiming, and FSM exploration.
    - Modifying the source code.
  5. Rerun synthesis and check your results.

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 18

# Optimizing for Specific Targets

---

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 8, \*Inferring High-Level Objects\*](#).

This chapter describes the following:

- [Optimizing Actel Designs](#), on page 760
- [Optimizing Altera Designs](#), on page 764
- [Optimizing Lattice Designs](#), on page 785
- [Optimizing Xilinx Designs](#), on page 797

## Optimizing Actel Designs

The Synplify and Synplify Pro synthesis tools support Actel designs. The following procedures Actel-specific design tips.

- [Using Predefined Actel Black Boxes](#), on page 760
- [Using ACTGen Macros](#), on page 761
- [Working with Radhard Designs](#), on page 762
- [Improving Performance in Actel Physical Synthesis Designs](#), on page 763

For additional Actel-specific information, see [Passing Information to the P&R Tools](#), on page 832 and [Generating Vendor-Specific Output](#), on page 836.

### Using Predefined Actel Black Boxes

The Actel macro libraries contain predefined black boxes for Actel macros so that you can manually instantiate them in your design. For information about using ACTGen macros, see [Using ACTGen Macros, on page 761](#). For general information about working with black boxes, see [Defining Black Boxes for Synthesis, on page 358](#).

To instantiate an Actel macro, use the following procedure.

1. Locate the Actel macro library file appropriate to your technology in one of these subdirectories under `synplify_install_dir/lib`.

|                      |                                                                                |
|----------------------|--------------------------------------------------------------------------------|
| <code>proasic</code> | ProASIC (500K), ProASICPLUS (PA), ProASIC3/3E, Fusion, and IGLOO/IGLOOe macros |
| <code>actel</code>   | Macros for all other Actel technologies.                                       |

Use the macro file that corresponds to your target architecture. If you are targeting the 1200XL architecture, use the `act2.v` or `act2.vhd` macro library.

2. Add the Actel macro library *at the top* of the source file list for your synthesis project. Make sure that the library file is first in the list.

3. For VHDL, also add the appropriate library and use clauses to the top of the files that instantiate the macros:

```
library family;
use family.components.all ;
```

Specify the appropriate technology in *family*; for example, act3.

## Using ACTGen Macros

The following procedure shows you how to include ACTgen macros in your design. For information about using Actel macro libraries, see [Using Predefined Actel Black Boxes, on page 760](#). For general information about working with black boxes, see [Defining Black Boxes for Synthesis, on page 358](#).

1. In ACTgen, generate the function you want to include.
2. Use the Actel netlist translation utility to convert the resulting EDIF netlist to VHDL or Verilog.
3. For VHDL macros, do the following:
  - Edit the ACTgen VHDL file, and add the appropriate library clause at the top of the file:

```
library family ;
use family.components.all
```
  - Include the VHDL version of the ACTgen result in your synthesis source file list.
4. For Verilog macros, do the following:
  - Include the appropriate Actel macro library file for your target architecture in your the source files list for your project.
  - Include the Verilog version of the ACTgen result in your source file list. Make sure that the Actel macro library is first in the source files list, followed by the ACTgen Verilog files, followed by the other source files.
5. Synthesize your design as usual.

## Working with Radhard Designs

The following procedure outlines how to specify radhard values for a design with the `syn_radhardlevel` attribute. Remember that the attribute is not recursive. It only applies to all registers at the level where it is set and does not affect lower-level registers.

You specify radhard values in modules and architecture in both the Attributes panel in SCOPE and in the source code. However, for registers, it must be specified in the source code only.

1. Add to your project the Actel macro files appropriate to the radhard values you plan to set in the design. The macro files are in `install_dir/lib/actel`:

| Radhard Value | Verilog Macro File | VHDL Macro File |
|---------------|--------------------|-----------------|
| cc            | cc.v               | cc.vhd          |
| tmr           | tmr.v              | tmr.vhd         |
| tmr_cc        | tmr_cc.v           | tmr_cc.vhd      |

For ProASIC3/3E devices only, you do not need to add the Actel macro file to your project.

2. To set a global or default `syn_radhardlevel` attribute, do the following:
  - Set the value in the source file for the module. The following sets all registers of `module_b` to `tmr`:

### VHDL

```
library synplify;
use synplify.attributes.all;
attribute syn_radhardlevel of
 behav: architecture is "tmr";
```

### Verilog

```
module module_b (a, b, sub,
 clk, rst) /*synthesis
 syn_radhardlevel="tmr"*/;
```

- Make sure that the corresponding Actel macro file (see step 1) is the first file listed in the project, if required.
3. To set a `syn_radhardlevel` value on a per register basis, set it in the source file. You can use a register-level attribute to override a default value with another value, or set it to a value of `none`, so that the global default value is not applied to the register. To set the value in the source file, add the

attribute to the register. For example, to set the value of register `bl_int` to `tmr`, enter the following in the module source file:

**VHDL**

```
library synplify;
use synplify.attributes.all;
attribute syn_radhardlevel of
 bl_int: signal is "tmr"
```

**Verilog**

```
reg [15:0] a1_int, bl_int
 /*synthesis syn_radhardlevel
 = "tmr"*/;
```

## Improving Performance in Actel Physical Synthesis Designs

The Synplify Premier tool is timing-driven, so optimizations depend on timing constraints and are applied until all constraints are met. Therefore, it is very important that you adequately apply timing constraints for physical synthesis and not over-constrain the tool. This section includes guidelines for applying constraints.

1. Verify constraints consistency between synthesis and P&R:
  - Clock constraints
  - Clock-to-clock constraints
  - I/O delays
  - I/O standard, drive, slew and pull-up/pull-down
  - Multi-cycle and false paths
  - Max-delay paths
  - DCM parameters
  - Register packing into IOB
2. Ensure the final physical synthesis slack is negative, but no more than 10-15% of the clock constraint.
3. Check the log file for Pre-placement timing snapshot.

If it indicates that a clock has positive slack at this point, but in the final results the clock has negative slack, use the `-route` constraint for the clock. This option allows you to control the amount of early timing optimizations for the clock domain. However, large `-route` values can degrade performance. Therefore, to determine the correct `-route` value to use, start with smaller values and increase iteratively. For example, start with half the difference between the estimate and actual slack, or 5% of the clock estimate, whichever is the smallest.

## Optimizing Altera Designs

This section includes some Altera technology-specific tips for optimizing your design. These tips are in addition to the general guidelines described in [Tips for Optimization](#), on page 426. This section discusses the following topics that are specific to Altera technologies:

- [Design Tips for APEX and FLEX Designs](#), next
- [FLEX Design Tips](#), on page 765
- [Determining ROM Implementation](#), on page 765
- [Working with Altera EABs and ESBs](#), on page 767
- [Working with Altera PLLs](#), on page 769
- [Specifying Altera I/O Locations](#), on page 772
- [Packing I/O Cell Registers in Altera Designs](#), on page 774
- [Specifying HardCopy and Stratix Companion Parts](#), on page 775
- [Specifying Core Voltage in Stratix III Designs](#), on page 776
- [Using LPMs in Simulation Flows](#), on page 777
- [Improving Altera Physical Synthesis Performance](#), on page 779
- [Working with Quartus II](#), on page 779

In addition, you can use the techniques described in these other topics, which apply to other vendors as well as Altera:

- [Defining Black Boxes for Synthesis](#), on page 358
- [Inferring RAMs](#), on page 372
- [Inferring Shift Registers](#), on page 410
- [Working with LPMs](#), on page 416
- [Passing Information to the P&R Tools](#), on page 832
- [Generating Vendor-Specific Output](#), on page 836



## Design Tips for APEX and FLEX Designs

Use these techniques when working with APEX and FLEX designs.

### APEX Design Tips

- Set the option to map to ATOM primitives. When the software maps elements to ATOM primitives, the Quartus tool can skip synthesis, thus reducing run time. The pin assignments, part information, and cliquing information are forward-annotated to Quartus.
- If you have a large design and need to conserve flip-flops, pack the registers into Apex I/O cells. See [Packing I/O Cell Registers in Altera Designs, on page 774](#) for more information.

### FLEX Design Tips

The software automatically maps logic to Altera cells like LCELL, carry and cascade primitives. However, the default setting for the Max+ Plus II place-and-route tool is to do technology mapping. You must reconfigure Max+ Plus II to take full advantage of the Synopsys synthesis results.

If you are not going to use the synthesis technology mapping, turn off the Map logic to Lcells option before you run synthesis.

## Determining ROM Implementation

The software automatically infers ROMs from CASE statements in the RTL code. This procedure shows you how to control the implementation of ROM in APEX and FLEX designs with the `syn_romstyle` attribute.

1. To implement the ROM structure as a block, do the following:
  - Apply the `syn_romstyle` attribute to the signal output value.
  - Set the value of the attribute to `block_ROM`. You can set the attribute in the source code, the SCOPE interface, or directly in the constraint file. See [Entering Attributes and Directives, on page 304](#) for information.

| <b>Format</b>          | <b>Example</b>                                                                                                                            |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog                | <pre>reg [3:0] z /* synthesis syn_romstyle="block_rom" */;</pre>                                                                          |
| VHDL                   | <pre>signal z : std_logic_vector(3 downto 0); attribute syn_romstyle : string; attribute syn_romstyle of z : signal is "block_rom";</pre> |
| Constraint file syntax | <pre>define_attribute {z_20[3:0]} syn_romstyle {block_rom}</pre>                                                                          |

- Run synthesis.

The software implements all small ROMs (less than seven address bits) as logic. It implements the larger ROM structures as extended system blocks (ESBs) in APEX designs and extended array blocks (EABs) in FLEX designs.

If you have to conserve ROM resources, you can turn off ROM implementation globally with the `altera_auto_use_esb` and `altera_auto_use_eab` attributes, and then specify the ROMs you want implemented as block ROMs with the `syn_romstyle` attribute.

- To implement the ROM structure as discrete logic, do the following:
  - Apply the `syn_romstyle` attribute to the signal output value.
  - Set the value of the attribute to `logic`.

| <b>Format</b>          | <b>Example</b>                                                                                                                        |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Verilog                | <pre>reg [3:0] z /* synthesis syn_romstyle="logic" */;</pre>                                                                          |
| VHDL                   | <pre>signal z : std_logic_vector(3 downto 0); attribute syn_romstyle : string; attribute syn_romstyle of z : signal is "logic";</pre> |
| Constraint file syntax | <pre>define_attribute {z_20[3:0]} syn_romstyle {logic}</pre>                                                                          |

- Run synthesis.

The software implements all small ROMs (less than seven address bits) and all other ROMs with this attribute as discrete logic primitives instead of blocks.

3. To view the ROM in your design, do the following:
  - Open the RTL view of the design.
  - Find the ROM block and push into it. A text window opens and displays the ROM table view of the data in the block.

## Working with Altera EABs and ESBs

An Altera EAB is an extended array block, in FLEX10K designs. An ESB is an extended system block in Apex 20K and 20KE designs.

1. Attach the `altera_implement_in_eab` attribute to the component you want to implement as an EAB, and set the value to 1.

See [altera\\_auto\\_use\\_eab Attribute](#), on page 915 in the *Reference Manual* for syntax details.

2. To implement an ESB, do the following:
  - Make your design hierarchical, and instantiate the module/entity in the ESB at the top level.
  - Attach the `altera_implement_in_esb` attribute to the component.
  - Set the value to true.

When this attribute is set, the software implements the logic as a PTERM in an extended system block.

3. Run synthesis.

For FLEX10KE, APEX20K, and 20KE designs, the software generates Altera-specific single or dual-port RAMs with asynchronous READs. When source code is written as a single port RAM, the software implements it as a dual-port RAM with single port RAM functionality, using the `LPM_RAM_DQ:ALTDPRAM` primitive.

### Verilog FLEX EAB Example

The following example uses the `altera_implement_in_eab` attribute to map logic into EABs for Altera FLEX devices. An integer square root of an input bus of width  $2n$  with a result bus size of  $n$  was computed. This function is mapped to a 256x4 block of RAM.

```

module sqrtb(z, a);
parameter asize = 8;
output [(asize/2)-1:0] z;
input [asize-1:0] a;
reg [(asize/2)-1:0] z;

always @(a) begin :lbl
integer i;
// r is remainder, tt is delta for adding one bit
// v is current sqrt value
reg [asize-1:0] v, r, tt;

v = 0;
r = a;
for (i = asize/2 - 1; i >= 0; i = i - 1) begin
tt = (v << (i + 1)) | (1 << (i + i));
if (tt <= r) begin
v = v | (1 << i);
r = r - tt;
end
end
z = v;
end
endmodule

```

## VHDL FLEX EAB Example

This example uses the `altera_implement_in_eab` attribute to map logic into EABs for Altera FLEX devices.

```

library ieee, synplify;
use ieee.std_logic_1164.all;

entity mymux is
port (in1: in std_logic_vector(9 downto 0);
sel : in std_logic;
dout : out std_logic_vector(9 downto 0));
end mymux;

architecture behave of mymux is
begin
dout <= in1 when sel = '1' else
(NOT in1) when sel = '0' else
(others => 'X');
end behave;

```

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

entity eab_test is
 port (a: in std_logic_vector(9 downto 0);
 s: in std_logic;
 o: out std_logic_vector(9 downto 0));
end eab_test;

architecture arch1 of eab_test is
 component mymux is
 port (in1: in std_logic_vector (9 downto 0);
 in2: in std_logic_vector (9 downto 0);
 sel : in std_logic;
 dout: out std_logic_vector (9 downto 0));
 end component mymux;
 attribute altera_implement_in_eab : boolean;
 attribute altera_implement_in_eab of U1: label is true;

begin
 U1: mymux port map (
 in1 => a,
 sel => s,
 dout => o);
end arch1;
```

## Working with Altera PLLs

The synthesis software recognizes the Altera PLL component, `altpll`, from the Stratix, Cyclone, and Arria GX device families. The following procedure shows you how to use this component in your designs. The procedure uses the Altera Megafunction wizard to generate structural VHDL or Verilog files for the Altera PLLs.

1. If you are using VHDL, the `altpll` component normally will be declared in the MegaWizard file, and you can comment out the `LIBRARY` and `USE` clauses in the file. The following shows an example of the lines to be commented out:

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

If the component declaration in the MegaWizard file is not compatible with a particular Quartus software version, use the appropriate `vhd` file packaged with the Synopsys software in the corresponding

lib/altera/quartus\_11nn directory. For example, the altera\_mf.vhd file for use with Quartus 8.1 is in the quartus\_1181 subdirectory.

2. If you are using Verilog, no action is necessary as the mapper understands the altpll component.

For compatibility with different Quartus versions, altera\_mf.v files are packaged with the software in the lib/altera/quartus\_11nn directory. Use the file from the directory that corresponds to the Quartus version that you are using.

3. Instantiate the altpll component in your design.
4. Add the MegaWizard Verilog or VHDL files to your project.
5. Open SCOPE and define the PLL input frequency in the SCOPE window. The synthesis software does not use the input frequency from the Altera MegaWizard software. Based on the input value you supply, the software generates the PLL outputs. All PLL outputs are assigned to the same clock group.
6. Set the target technology and the Quartus version (Implementation Options - > Implementation Results), and synthesize as usual. The software uses the altpll component information and the constraints when synthesizing your design. The synthesis software forward-annotates the PLL input constraints to Quartus.

## Instantiating Special Buffers as Black Boxes in Altera Designs

You can instantiate special buffers, like global buffers for clocks, sets/resets, and other heavily loaded signals, as black boxes in your design. See the Altera documentation for details about special buffers and the number of resources available for the part you are using.

1. Define a black-box module for your special buffer with the syn\_black\_box directive.

See the examples below and [syn\\_black\\_box Directive, on page 964](#) in the *Reference Manual* for syntax details.

2. Use this black-box module to buffer the signals you want assigned to special buffers.
3. Synthesize the design and place-and-route as usual.

The Altera tools accept the black box.

### Verilog Example of Instantiating Special Buffers as Black Boxes

```
module global(a_out, a_in) /* synthesis syn_black_box */ ;
output a_out;
input a_in;

/* This continuous assignment is used for simulation,
 but is ignored by synthesis. */
assign a_out = a_in;
endmodule

module top(clk, pad_clk) ;
output pad_clk;
input clk;
// pad_clk is the primary input
global clk_buf(pad_clk, clk);
endmodule
```

### VHDL Example of Instantiating Special Buffers as Black Boxes

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

entity top is
port (clk : out std_logic;
pad_clk : in std_logic);
end top;

architecture structural of top is
-- In this example, "global" is an Altera vendor macro directly
-- instantiated in the Altera VHDL design as a black box.

component global
port(a_out : out std_logic; a_in : in std_logic) ;
end component;

-- Set the syn_black_box attribute on global to true.
attribute syn_black_box of global: component is true;
```

```

-- Declare clk, the internal global clock signal
begin
-- pad_clk is the primary input
clk_buf: global port map (clk, pad_clk);
end structural;

```

## Specifying Altera I/O Locations

You can specify I/O locations in Altera designs using the `syn_loc` attribute. If you do not specify I/O locations, the P&R tool automatically assigns them locations.

1. If you used the QSF2SDC utility, to translate Altera QSF `set_location_assignment` and `set_instance_assignment` constraints, do nothing.

The utility automatically assigns the `syn_loc` attribute to the pins with constraints.

2. To define an I/O location manually, use the following syntax:

|                     |                                                                |
|---------------------|----------------------------------------------------------------|
| Top-level .sdc file | <b>define_attribute {portName} syn_loc {pinNumbers}</b>        |
| Verilog             | <b>object/* synthesis syn_loc = "pinNumbers" */</b>            |
| VHDL                | <b>attribute syn_loc of object: objectType is "pinNumbers"</b> |

### Verilog FLEX `syn_loc` Example

```

module adder_8(cout, sum, a, b, cin);

/* Put the cout output on pin 159. */
output cout /* synthesis syn_loc="@159" */;
output [7: 0] sum /* synthesis syn_loc=
 "@17,@166,@191,@152,@15,@148,@147,@149" */;

/* Put the "a" input bus from bits 7 through 0 on pins
 194, 177, 70, 97, 109, 6, 174, and 204, respectively. */
input [7: 0] a /* synthesis syn_loc=
 "@194,@177,@70,@97,@109,@6,@174,@204" */;

/* Let Altera place the "b" and "cin" inputs. */
input [7:0] b;
input cin;
assign {cout, sum} = a + b + cin;
endmodule

```



### Verilog APEX20K/E/C syn\_loc Example

```

module alu(out1, opcode, a, b, sel, clk);
output [7:0] out1 /*synthesis syn_loc =
 "14,12,11,5,21,18,16,15" */;
input [2:0] opcode;
input [7:0] a,b;
input sel, clk;
reg [7:0] alu_tmp;
reg [7:0] out1;

```

### VHDL FLEX10K / ACEX1K / MAX syn\_loc Example

```

library ieee, synplify;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adder_8 is
 port (a, b: in std_logic_vector (7 downto 0);
 result: out std_logic_vector(7 downto 0));

 -- Assign the "result" output bus from bits 7 down
 -- through 0 on pins 17, 166, 191, 152, 15, 148, 147,
 -- and 149, respectively
 attribute syn_loc : string;
 attribute syn_loc of result : signal is
 "@17, @166, @191, @152, @15, @148, @147, @149";

 -- Assign the "a" input bus from bits 7 through 0 on
 -- pins 194, 177, 70, 97, 109, 6, 174, and 204.
 attribute syn_loc of a : signal is
 "@194, @177, @70, @97, @109, @6, @174, @204";

 -- Let the "b" input be placed by Altera
end adder_8;

architecture behave of adder_8 is
begin
 result <= a + b;
end;

```

### VHDL APEX20K/E/C, APEX II syn\_loc Example

```

attribute syn_loc : string;
attribute syn_loc of result : signal is "14, 12,
 11, 5, 21, 18, 16, 15";

```

## Packing I/O Cell Registers in Altera Designs

You can improve input or output path timing in designs by packing registers into I/O cells with the `syn_useioff` attribute.

1. To pack the registers globally, set `syn_useioff=1` on the top-level module or architecture. Specify the attribute in the source code, the SCOPE interface, or directly in the constraint file.

| Format                 | Example                                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| Verilog                | <pre>module test(d, clk, q) /* synthesis syn_useioff=1 */;</pre>                                                                 |
| VHDL                   | <pre>architecture rtl of test is   attribute syn_useioff : boolean;   attribute syn_useioff of rtl : architecture is true;</pre> |
| Constraint file syntax | <pre>define_global_attribute syn_useioff 1</pre>                                                                                 |

2. To set the attribute locally, set `syn_useioff=1` on a port.

| Format                 | Example                                                                                                                                                                                                                                                      |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog                | <pre>module test(d, clk, q);   input [3:0] d;   input clk;   output [3:0] q /* synthesis syn_useioff=1 */;   reg q;</pre>                                                                                                                                    |
| VHDL                   | <pre>entity test is   port (d : in std_logic_vector (3 downto 0);         clk : in std_logic;         q : out std_logic_vector (3 downto 0);         attribute syn_useioff : boolean;         attribute syn_useioff of q : signal is true;   end test;</pre> |
| Constraint file syntax | <pre>define_attribute {p:q[3:0]} syn_useioff 1</pre>                                                                                                                                                                                                         |

3. Synthesize the design.

The order of precedence used when there are conflicts for packing is registers, followed by ports, and finally global.

The software infers I/O cells with presets or clears, and an embedded flip-flop in the I/O cell. The software packs registers with asynchronous clear pins and asynchronous preset pins for APEX20KE I/O cells.

If `syn_useioff` is enabled for Arria GX and Stratix families, registers are not packed into Multiply/Accumulate (MAC) blocks.

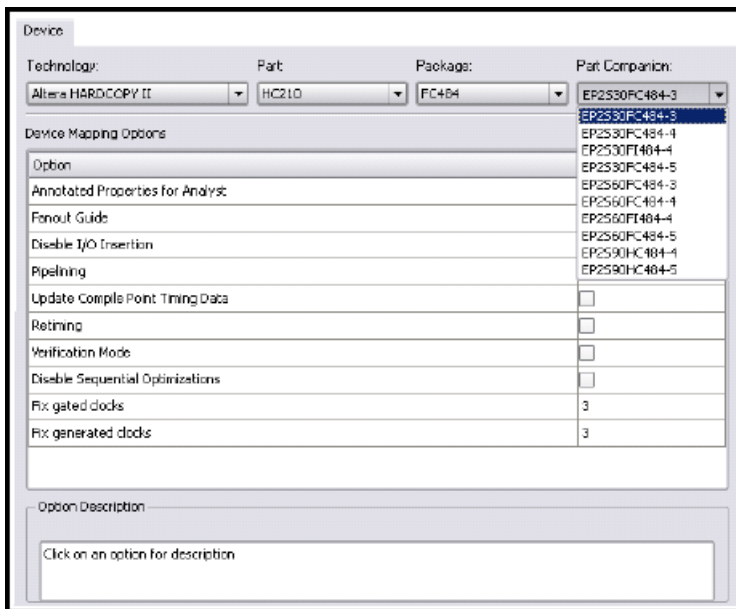
## Specifying HardCopy and Stratix Companion Parts

For Stratix II, Stratix III, and Stratix IV devices, you can specify an associated HardCopy II, HardCopy III, or HardCopy IV companion part to allow you to migrate from Stratix to HardCopy in Quartus. By default, no companion part is specified for a Stratix device family. However, for a HardCopy device family, a Stratix companion part must be specified.

You select the companion device in the Device tab of the Implementation Options dialog box as in the following example:

You can use any of the following methods to specify companion parts:

- Select the companion device in the Device tab of the Implementation Options dialog box as shown here:



- Use this Tcl command, where *partName* is the part name and number:

```
set_option -part_companion partName
```

When you specify a companion part, the mapper targets the device with the least resources. For example, if your Stratix device has five memories and the companion HardCopy device has four memories, the mapper only uses four memory resources and maps the rest to logic.

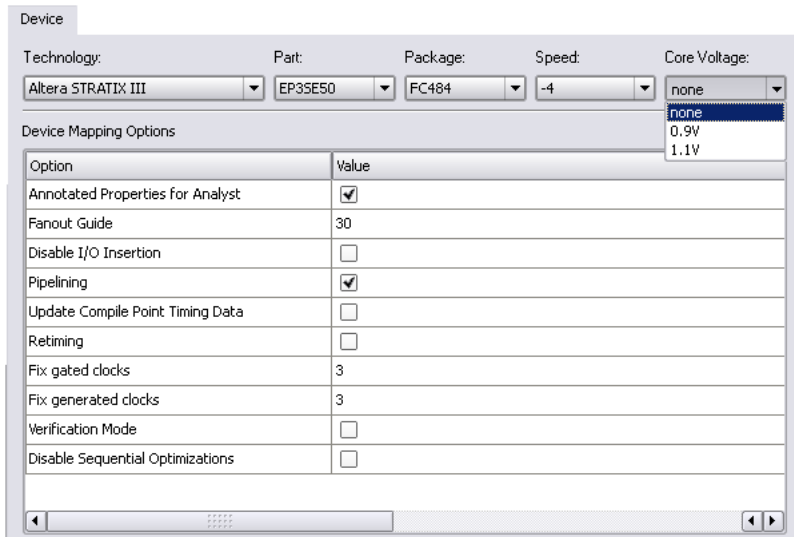
## Specifying Core Voltage in Stratix III Designs

For some Stratix III devices, you can specify core voltage. Do the following:

1. Click Implementation Options and do the following:
  - On the Device tab, set Technology to a Stratix III device.
  - Set Speed to -4.

This makes the Core Voltage option available.

2. Set Core Voltage to the value you want, and click OK.



Alternatively, you can use the corresponding Tcl command: `set_option -voltage <voltage_value>`.

For example:

```
set_option -voltage 1.1V
set_option -voltage none
```

## Using LPMs in Simulation Flows

This section describes how to use instantiated LPMs in simulation flows. For information about instantiating LPMs, see [Working with LPMs, on page 416](#).

### Simulation Flows

The simulation flows vary, depending on the method used to instantiate the LPMs. For information about instantiating LPMs, see [Instantiating Altera LPMs Using VHDL Prepared Components, on page 421](#), [Instantiating Altera LPMs as Black Boxes, on page 417](#), and [Instantiating Altera LPMs Using a Verilog Library, on page 423](#). The following table summarizes the differences between the flows:

|                                 | <b>Black Box Flow</b>                                  | <b>Verilog Library/VHDL Prepared Component Flows</b> |
|---------------------------------|--------------------------------------------------------|------------------------------------------------------|
| Applies to any LPM              | Yes                                                    | No                                                   |
| Synthesis LPM timing support    | No                                                     | Yes                                                  |
| Synthesis procedure             | Many steps                                             | Simple                                               |
| RTL simulation                  | Complicated steps                                      | Easy                                                 |
| Post-synthesis (.vm) simulation | Yes                                                    | No                                                   |
| Post-P&R (.vo) simulation       | Yes                                                    | Yes                                                  |
| Software version                | Any version<br>Max+PlusII<br>Quartus II 1.0 or earlier | Quartus II 1.1 or later                              |

### Black Box Method Simulation Flow

Use the following flow when you instantiate LPMs as Verilog or VHDL black boxes. You can use this procedure for any LPM supported by Altera.

1. Use the Altera MegaWizard Plug-In Manager to create an LPM megafunction with the same module and port names as the black-box module in your synthesis design.
2. Compile the following:
  - Test bench
  - The design (RTL, post-synthesis `.vm` file, or the post-P&R `.vo` file)
  - The `.v` file you generated in the previous step
3. Compile the LPM megafunction simulation model: `220model.v` or `altera_mf.v`.
4. For `.vm` or `.vo` simulation, compile the primitive simulation model. For example `apex20Ke_atoms.v`.
5. Simulate the design.

## Library/Prepared Component Simulation Flow

Use this simulation procedure when you use a Verilog library or VHDL prepared components to instantiate the LPMs. You can use this flow for `.vo` simulation if your design contains the supported LPMs.

1. Instantiate the LPMs.
  - For VHDL designs, use the prepared components methods described in [Instantiating Altera LPMs Using VHDL Prepared Components, on page 421](#) or [Instantiating Altera LPMs as Black Boxes, on page 417](#).
  - For Verilog designs, use the library methods described in [Instantiating Altera LPMs Using a Verilog Library, on page 423](#) or [Instantiating Altera LPMs as Black Boxes, on page 417](#).
2. Compile the test bench and design. The design can be either RTL or the post-P&R `.vo` file.
3. Compile the LPM megafunction simulation model: `220model.v` or `altera_mf.v`.
4. For `.vo` simulation, compile the primitive simulation model. For example `apex20Ke_atoms.v`.
5. Simulate the design.

## Improving Altera Physical Synthesis Performance

The Synplify Premier tool is timing-driven; optimizations depend on timing constraints and are applied until all constraints are met. Therefore, it is very important that you adequately apply timing constraints and not over-constrain the tool. This section includes guidelines for applying constraints.

- Verify the consistency of constraints between synthesis and P&R:
  - Clock constraints
  - Clock-to-clock constraints
  - IO delays
  - IO standard, drive, slew and pull-up/pull-down
  - Multi-cycle and false paths
  - Max-delay paths
  - DCM parameters
  - Register packing into IOB
  - SYN\_LOC on IO pins and pad types
  - Placement constraints on instances
- Ensure that the final physical synthesis slack is negative, but no more than 10-15% of the clock constraint.

## Working with Quartus II

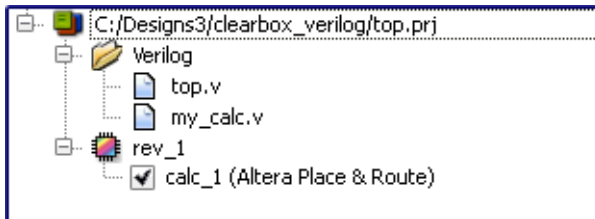
The following procedures show you how to use the synthesis information to run Quartus II in an integrated mode with the Synopsys FPGA synthesis tool, directly from the synthesis interface, or in a standalone batch mode. Each procedure assumes that you have set the `QUARTUS_ROOTDIR` environment variable to point to your Quartus II installation directory. After synthesis, the Verilog netlist (`.vqm`), forward annotated timing constraints and pin assignments (`.tcl/.scf`) are placed in the named Quartus project.

### Integrated Mode

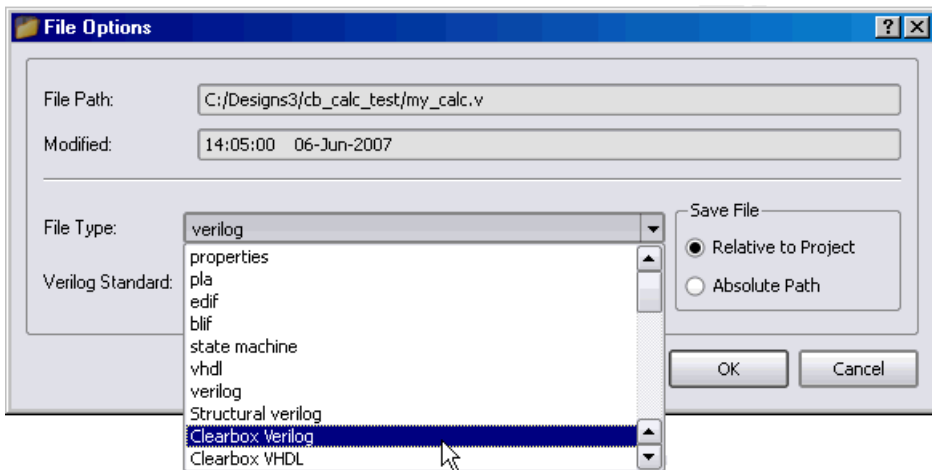
To run Quartus II in an integrated mode:

1. In the project view, click the Add P&R Implementation button to display the Add New Place & Route Job dialog box.

- Optionally assign a P&R job name and click OK. The job is displayed in the project view under the active implementation.



- Right click on the RTL source file and select File Options to display the File Properties dialog box.
- In the File type field drop-down menu, select either Clearbox Verilog or Clearbox VHDL according to the RTL file type and then click OK.



- Click the Run button; the clearbox netlist is copied to the PR\_1 directory, the design is synthesized, and then placed and routed.

## Synthesis Interface

To place and route interactively from the synthesis interface, select Quartus II-> Launch Quartus from the Options menu. This command opens the Quartus II GUI and automatically runs Quartus II with the project settings from the



synthesis run. You can monitor placement and routing as it progresses, see errors and warning messages, check what percentage of the job has completed, and execute other Quartus II commands.

## Batch Mode

To run Quartus II in batch mode, select Quartus II->Run Background Compile from the Options menu. This command runs place and route using the default Quartus settings and the information in the `project_name_cons.tcl` and `project_name.tcl` files to set up and compile the Quartus project and to read the forward-annotated information from the prior synthesis run. Quartus log files are updated with placement, routing, and timing data as the design compiles.

## Configuring Max+Plus II for FLEX and ACEX1K

Altera Max+Plus II software provides LAB arrays containing Logic Cells (LCELLs), global routing between LABs, and local routing within LABs. LCELLs implement many, simple logic gates. Taken together, these elements control mapping for the FLEX and ACEX1K architectures.

The Synopsys synthesis tools map technology directly to Altera LCELLs. It inserts LCELLS into its netlist to improve performance and logic use. You can configure Max+Plus II in two ways:

- Using mapped logic from the synthesis tools. See [Running Max+Plus II With Mapped LCELLs, on page 781](#) for details.
- Without mapped logic, as described in [Running Max+Plus II Without Mapped LCELLs, on page 783](#). You might need to use this method when registers are not properly packed, or if the design is too big or too slow.

## Running Max+Plus II With Mapped LCELLS

To map logic to LCELLs, follow these steps:

1. Do the following in the synthesis tool graphic user interface:
  - Turn on the Map Logic to LCELLs option (Device tab of the Implementation Options dialog box).
  - Synthesize the design.

- If necessary, set the `syn_edif_name_length` attribute to `restricted`. This attribute ensures that port names in the EDIF output file do not exceed 30 characters in length.
2. In Max+Plus II, select `Assign->Global Project Logic Synthesis` and change `Global Project Synthesis Style` to `WYSIWYG` (the default is `NORMAL`). You can also turn ON the following two options:
    - `Automatic I/O Cell Registers` – This option moves registers into the I/Os, reducing area and improving I/O performance, but selecting this option might worsen internal clock frequency.
    - `Automatic Register Packing` – This option (in `FLEX10K`) packs unrelated logic and flip-flops into the same cells, reducing area and sometimes delay, but selecting this option might worsen routability.
  3. Do the following:
    - Click the `Define Synthesis Style` button.
    - Click the `Advanced Options` button. The `Advanced Options` dialog box displays.
    - Turn OFF the `NOT Gate Push-Back` option. Click OK three times to exit the dialog boxes.
  4. Run the place-and-route software.
  5. Review the cell types in the Max+Plus II report file.

If cell types include `LCELL`, `carry`, `cascade`, `dff`, and `dffe`, then Max+Plus II honored the mapping instructions, and you have finished.

If the report file contains a high percentage of `and2`, `or2`, or similar cell types, or if many such cells exist along your critical paths, then Max+Plus II did not honor the instructions and may have worsened the results. If such unwanted cell types occur, use the method described in [Running Max+Plus II Without Mapped LCELLs](#), on page 783.

6. If you want faster timing, reset timing options in the synthesis tool interface, re-run the tool on your design, and re-run Max+Plus II.

## Running Max+Plus II Without Mapped LCELLS

To obtain fast performance or small area when mapping cannot be used, follow the steps below.

1. Use either of these methods to turn off the LCELLS option:
  - Select Project -> Implementation Options -> Device and click off Map Logic to LCELLS.
  - Use the `set_option Tcl` command.
2. If necessary, set the `syn_edif_name_length` attribute to `restricted`, and synthesize the design.

This attribute ensures that port names in the EDIF output file do not exceed 30 characters in length.

3. If you want fast performance, take these steps:

- In Max+Plus II, choose Assign -> Global Project Logic Synthesis. In the dialog box, change Style to FAST (the default is NORMAL).
- Set Automatic Fast I/O and Automatic Register Packing to ON.

Automatic Fast I/O moves registers into the I/Os, saving area and improving I/O performance, but perhaps worsening internal clock frequency.

Automatic Register Packing (in FLEX10K) packs unrelated logic and flip-flops into the same cells. This saves area and sometimes delay, but may worsen routability.

- Click OK to exit the form.
- Run placement and routing.

4. If you want small area, perform these steps:

- In Max+Plus II, choose Assign->Global Project Logic Synthesis. In the dialog box, change Style to SLOW (the default is NORMAL).
- Set Automatic Fast I/O and Automatic Register Packing to ON.

Automatic Fast I/O moves registers into the I/Os, saving area and improving I/O performance, but perhaps worsening internal clock frequency.

Automatic Register Packing (in FLEX10K) packs unrelated logic and flip-flops into the same cells. This saves area and sometimes delay, but may worsen routability.

- Click the Define Synthesis Style button.
- Click the Use Default button in the dialog box. Change the Carry Chain and Cascade Chain settings from Ignore to Manual.
- Click OK twice to exit the forms.
- Run placement and routing.

## Configuring Max+Plus II for MAX Designs

To prepare the design for the P&R tool, you must make sure the port names in your output file are the right length for Max+Plus II. Then select a synthesis style that matches your objectives and configure Max+Plus II to access the LMF.

1. Set the `syn_edif_name_length` attribute value to `restricted`.

This ensures that the port names in the EDIF output file do not exceed 30 characters. This matches the name length requirement for Max+Plus II.

2. Select a synthesis style:

- From Altera Max+Plus II open the Assign -> Global Project Logic Synthesis dialog box.
- Set the Global Project Synthesis Style option to either FAST or NORMAL.

If you select FAST for a congested part, Max+Plus II may be unable to place-and-route your design, because global routing connections to LABs are overloaded with LCELLs. If this congestion occurs, let synthesis automatically map the logic into LCELLs.

3. Configure the compiler to access the LMF and interpret primitives in EDIF netlists.
  - Select the compiler, and then select Interfaces->EDIF Netlist Reader Settings.
  - Click CUSTOMIZE. A text box will display.
  - Enter the full path to the LMF: `install_dir/lib/synplcty.lmf`

- Click the small box to the left of the path name.
- Make sure the power and ground signal names are set to default values of VCC and GND.

## Optimizing Lattice Designs

The Synplify and Synplify Pro synthesis tools include support for Lattice technologies. This section describes the following techniques for working with Lattice designs:

- [Instantiating Lattice Macros](#), on page 785
- [Using Lattice GSR Resources](#), on page 787
- [Inferring Carry Chains in Lattice XPLD Devices](#), on page 788
- [Inferring Lattice PIC Latches](#), on page 788
- [Controlling I/O Insertion in Lattice Designs](#), on page 794
- [Forward-Annotating Lattice Constraints](#), on page 795
- For additional information about working with Lattice designs, see [Passing Information to the P&R Tools](#), on page 832 and [Generating Vendor-Specific Output](#), on page 836.

### Instantiating Lattice Macros

You can instantiate Lattice macros that are predefined in the Lattice libraries that come with the tool, in the `synplify_install_dir/lib` directory.

1. To use a Verilog macro library, add the appropriate library to your project, making sure that it is the first file in the source files list.

The Verilog macro libraries are under the `synplify_install_dir/lib` directory: Add the library appropriate to the technology you are using

|              |                                                                                         |
|--------------|-----------------------------------------------------------------------------------------|
| ORCA devices | <code>synplify_install_dir/lib/lucent/orca*.v</code>                                    |
|              | Replace the asterisk with either 2, 3, or 4, according to the ORCA series you are using |

---

|                       |                                                                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| LatticeXP2/XP devices | <i>synplify_install_dir/lib/lucent/xp.v</i><br><i>synplify_install_dir/lib/lucent/xp2.v</i>                                                  |
| LatticeSC/SCM devices | <i>synplify_install_dir/lib/lucent/sc.v</i>                                                                                                  |
| MachXO devices        | <i>synplify_install_dir/lib/lucent/machxo.v</i>                                                                                              |
| ECP/ECP2/EC devices   | <i>synplify_install_dir/lib/lucent/ecp.v</i><br><i>synplify_install_dir/lib/lucent/ecp2.v</i><br><i>synplify_install_dir/lib/lucent/ec.v</i> |
| ispXPGA devices       | <i>synplify_install_dir/lib/lattice/laval.v</i>                                                                                              |
| CPLD devices          | <i>synplify_install_dir/lib/cpld/lattice.v</i>                                                                                               |

- To use a VHDL macro library, add the appropriate library and use clauses to your VHDL source code at the beginning of the design units that instantiate the macros.

You only need the VHDL macro libraries for simulation, but it is good practice to add them to the code. The library names may vary, depending on the map file name, which is often user-defined. The simulator uses the map file names to point to a library.

|                       |                                                                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CPLD Devices          | <code>library lattice;<br/>use lattice.components.all;</code>                                                                                                                                         |
| ORCA Devices          | Replace the asterisk with the series number (2, 3, or 4) for the Lattice ORCA Series 2, Series 3, or Series 4 macro library you are using.<br><code>library orca*;<br/>use orca*.orcacomp.all;</code> |
| LatticeXP2/XP devices | <code>library xp;<br/>use xp.components.all<br/>library xp2;<br/>use xp.components.all</code>                                                                                                         |
| LatticeSC/SCM devices | <code>library sc;<br/>use sc.components.all</code>                                                                                                                                                    |
| MachXO devices        | <code>library machxo;<br/>use machxo.components.all</code>                                                                                                                                            |

---

|                     |                                                                                                                            |
|---------------------|----------------------------------------------------------------------------------------------------------------------------|
| ECP/ECP2/EC devices | <pre>library ecp; use ecp.components.all;  library ecp2; use ecp2.components.all;  library ec; use ec.components.all</pre> |
| ispXPGA Devices     | <pre>library lava; use lava.components.all;</pre>                                                                          |

---

3. Instantiate the macros from the library as described in [Instantiating Black Boxes and I/Os in Verilog, on page 358](#) and [Instantiating Black Boxes and I/Os in VHDL, on page 360](#).

## Using Lattice GSR Resources

The following procedure describes how to use GSR (global set/reset) resources and check resource usage. The GSR resource is a prerouted signal that connects to the reset input of every flip-flop, regardless of any other defined reset signals.

1. For the LatticeECP/ECP2/EC, LatticeXP2/XP, LatticeSC/SCM, MachXO, and ORCA families, you can control the use of GSR resources as follows:
  - To improve routability and performance, use the dedicated GSR resource. Select Project ->Implementation Options and enable the Force GSR Usage option on the Device tab.

When you set this option, the synthesis software creates a GSR instance to access the resource. It uses the GSR resource for reset signals, instead of general routing. All registers are reset. when the GSR is activated, even if some flip-flops do not have a reset.
  - If a global set/reset does not correctly initialize the design, turn off the option. Select Project ->Implementation Options and disable the Force GSR Usage option on the Device tab. When this option is off, the software does not use the GSR resource unless all flip-flops have resets, and all resets use the same signal.
2. To optimize area, set the Resource Sharing option, as described in [Sharing Resources, on page 450](#).

3. To check resource usage, do the following:
  - Synthesize the design.
  - Select View Log and check the Resource Usage section. For ORCA families, you can compare the LUTs in the synthesis usage report to the occupied PFUs (function units) in the report generated after placement and routing. Each PFU consists of four 4-input LUTs and four registers. An occupied PFU means that least one LUT or register was used.

## Inferring Carry Chains in Lattice XPLD Devices

For XPLD devices, you can control the inference of carry chains with the `syn_use_carry_chain` attribute. By default, all counters are implemented as carry chains when they are over 4 bits wide. To override this, set the `syn_use_carry_chain` attribute with a value of 0 on the registers of the counter or adder.

## Inferring Lattice PIC Latches

The following procedure shows you how to control the inference of programmable I/O cells (PICs) in Lattice designs.

1. For the software to automatically infer PICs, make sure of the following:
  - The latch must be at the input port.
  - The latch must be directly driven by the input FPGA pad.
  - The design has one of the supported input control schemes: no clear or reset controls, and asynchronous clear or asynchronous resets.

After synthesis, the tool implements the following primitives for the PICs:

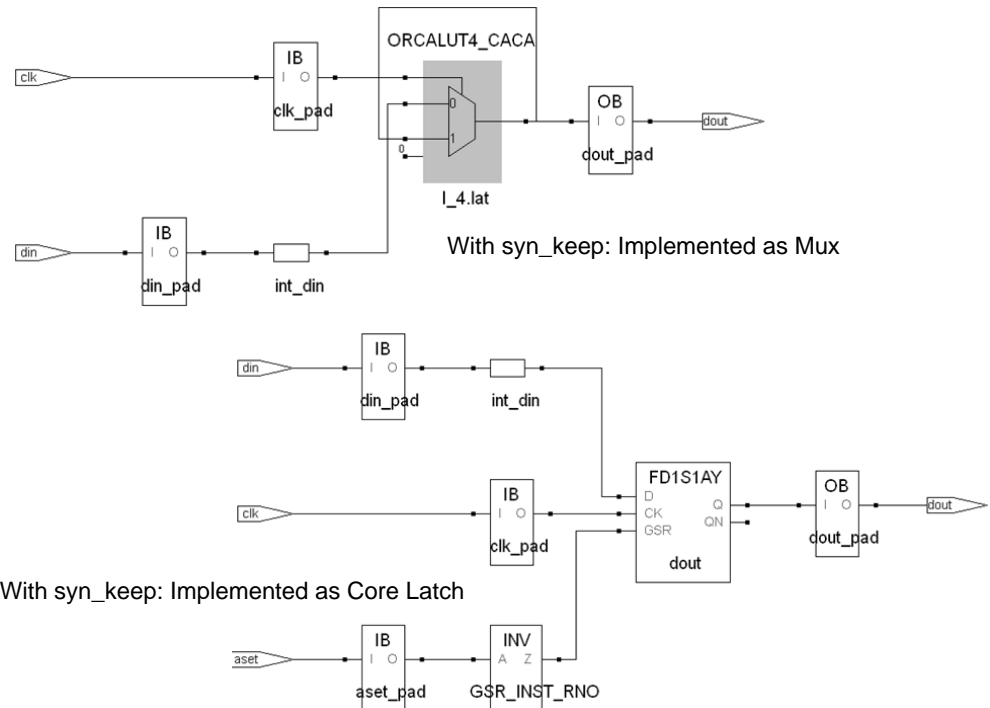
|        |                                                                     |
|--------|---------------------------------------------------------------------|
| IF1S1D | Latches with asynchronous clear<br>Latches with GSR used for clear  |
| IF1S1B | Latches with asynchronous reset<br>Latches with GSR used for reset. |

See [Examples of PIC Latches, on page 789](#) for examples of inferred PIC latches.



- If you do not want to infer a PIC, set `syn_keep` on the input data net for the latch.

After synthesis, the tool implements the latch as either a core latch with the LATCH primitive or as a mux, depending on the Lattice technology you selected. The following figure shows an input latch with no reset or clear implemented as a mux and a core latch in different technologies.



## Examples of PIC Latches

The following examples show how the tool infers PICs from the code:

- [Positive Level Data Latch with No Resets or Clears](#), on page 790
- [Negative Level Data Latch with No Resets or Clears](#), on page 791
- [Positive Level Data Latch with Asynchronous Reset](#), on page 792
- [Positive Level Data Latch with Asynchronous Clear](#), on page 793

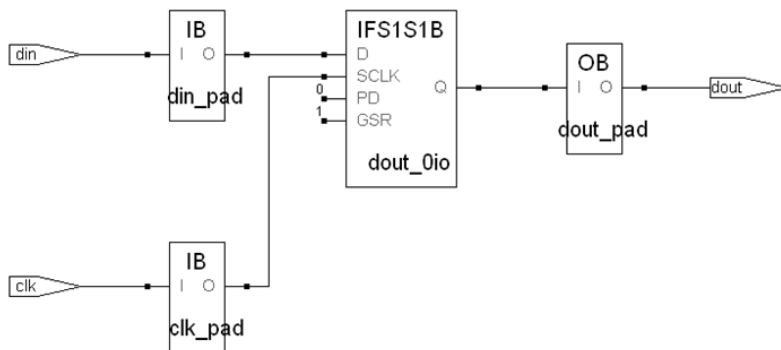
## Positive Level Data Latch with No Resets or Clears

```
VHDL library ieee; use ieee.std_logic_1164.all;
 entity inlatch is port
 (clk : in std_logic;
 din : in std_logic;
 dout: out std_logic);
 end entity inlatch;

 architecture bhve of inlatch is
 begin
 process(clk,din)
 begin
 if clk='1' then
 dout <= din;
 end if;
 end process;
 end bhve;
```

```
Verilog module inlatch (clk,din,dout);
 input clk; input din; output dout;
 reg dout;
 always @(clk)
 begin
 if(clk)
 dout <= din;
 end
end
endmodule
```

With this code, the tool implements the IFS1S1B latch primitive in the Technology view:

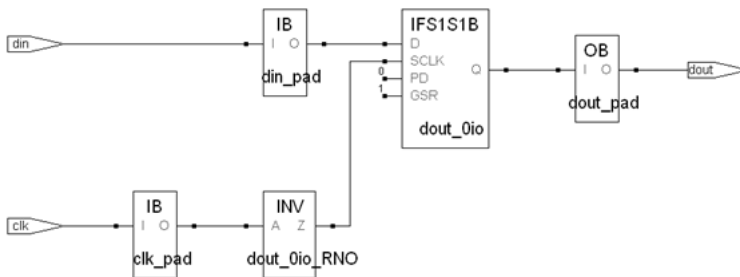


## Negative Level Data Latch with No Resets or Clears

```
VHDL library ieee; use ieee.std_logic_1164.all;
 entity inlatch is port
 (clk : in std_logic;
 din : in std_logic;
 dout: out std_logic);
 end entity inlatch;
 architecture bhve of inlatch is
 begin
 process(clk,din)
 begin
 if clk='0' then
 dout <= din;
 end if;
 end process;
 end bhve;
```

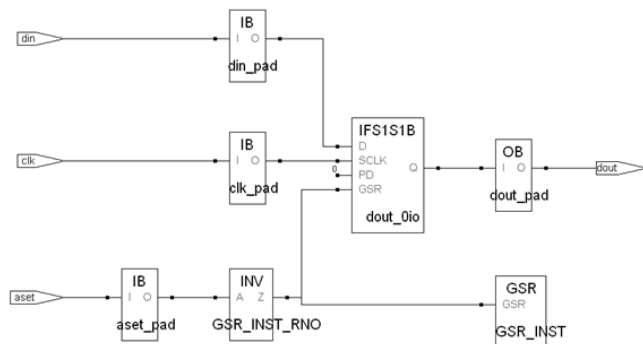
```
Verilog module inlatch (clk,din,dout);
 input clk; input din; output dout;
 reg dout;
 always @(clk)
 begin
 if(!clk)
 dout <= din;
 end
 endmodule
```

With this code, the tool implements the IFS1S1B latch primitive in the Technology view:



## Positive Level Data Latch with Asynchronous Reset

The tool infers the IFS1S1B latch primitive in the Technology view with the code shown below:



```
VHDL library ieee; use ieee.std_logic_1164.all;
 entity inlatch is port
 (clk : in std_logic; aset: in std_logic;
 din : in std_logic; dout: out std_logic);
 end entity inlatch;
 architecture bhve of inlatch is
 begin
 process(clk,din,aset)
 begin
 if aset ='1' then
 dout <='1';
 elsif clk='1' then
 dout <= din;
 end if;
 end process;
 end bhve;
```

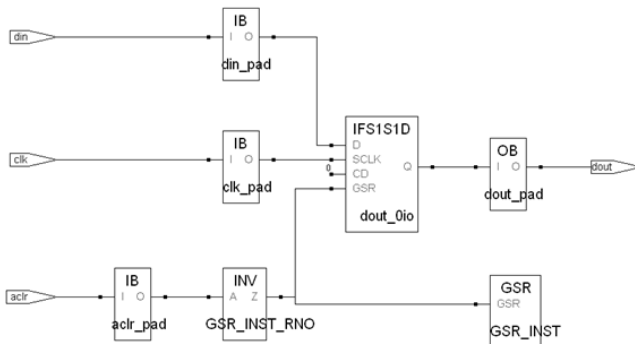
```
Verilog module inlatch(clk,din,aset,dout);
input clk; input din; input aset; output dout;
reg dout;
always @(clk or aset)
begin
 if(aset)
 dout <= 1'b1;
 else if (clk)
 dout <= din;
end
endmodule
```

## Positive Level Data Latch with Asynchronous Clear

```
VHDL library ieee; use ieee.std_logic_1164.all;
 entity inlatch is port
 (clk : in std_logic; aclr: in std_logic;
 din : in std_logic; dout: out std_logic); }
 end entity inlatch;
 architecture bhve of inlatch is
 begin
 process(clk,din,aclr)
 begin
 if aclr ='1' then
 dout <='0';
 elsif clk='1' then
 dout <= din;
 end if;
 end process;
 end bhve;
```

```
Verilog module inlatch(clk,din,aclr,dout);
 input clk; input din; input aclr; output dout;
 reg dout;
 always @(clk or aclr)
 begin
 if(aclr)
 dout <= 1'b0;
 else if (clk)
 dout <= din;
 end
end
endmodule
```

With this code, the tool infers the IFS1S1D primitive in the Technology view:



## Controlling I/O Insertion in Lattice Designs

You can control I/O insertion globally, or on a port-by-port basis.

1. 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.
- If you do not want to insert any I/O pads in the design, enable Disable I/O Insertion

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.

- If you want to insert I/O pads, disable the Disable I/O Insertion option.

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.

2. To force I/O pads to be inserted for input ports that do not drive logic, follow the steps below.

- To force I/O pad insertion at the module level, set the `syn_force_pads` attribute on the module. Set the attribute value to 1. To disable I/O pad insertion at the module level, set the `syn_force_pads` attribute for the module to 0.
- To force I/O pad insertion on an individual port, set the `syn_force_pads` attribute on the port with a value to 1. To disable I/O insertion for a port, set the attribute on the port with a value of 0.

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.

## Forward-Annotating Lattice Constraints

For LatticeECP2/ECP/EC, LatticeSCM, LatticeXP2/XP, MachXO, and ORCA technology families, you can forward-annotate multicycle and false path constraints to ispLEVER by following the procedure below. For additional information about forward-annotation, see [Generating Constraint Files for Forward Annotation](#), on page 105.

1. To forward-annotate a from, to, or through multicycle constraint, open the SCOPE spreadsheet and do either of the following:

- Click the Multi-Cycle Paths tab. Depending on the type of constraint you want to set, select or type the instance name under the To, From or Through column. Next, set the number of clock cycles under the Cycles column.

When you set this constraint, the software runs timing-driven synthesis and then forward-annotates the constraint.

- Click the Other tab. In the Command column, type `define_multicycle_path`. In the Arguments column, type `-from` and the source port or register name, and `-to` and the destination port or register name. For example: `-from in0_int -to output 2`.

When you set this constraint from the Other tab, the software forward-annotates the constraint, but does not run timing-driven synthesis using this constraint.

2. To forward-annotate a false path constraint, open the SCOPE spreadsheet and do either of the following:

- Click the False Paths panel. Depending on the type of constraint you want to set, select or type the instance name under the To, From or Through column. When you set this constraint, the software runs timing-driven synthesis and then forward-annotates the constraint.
- Click the Other tab. In the Command column, type `define_false_path`. In the Arguments column, type `-from` and the source port or register name, and `-to` and the destination port or register name. For example:

| Command                        | Arguments                             |
|--------------------------------|---------------------------------------|
| <code>define_false_path</code> | <code>-from in1_int -to output</code> |
| <code>define_false_path</code> | <code>-from in* -to out*</code>       |

When you set this constraint from the **Other** tab, the software forward-annotates the constraint, but does not run timing-driven synthesis using this constraint.

3. Select Project->Implementation Options and enable the Write Vendor Constraint File option on the Implementation Results tab.
4. Run your design. The synthesis tool creates the \$DESIGN\_synplify.lpf file in the same directory as your results files.
5. Start the Lattice ispLEVER place-and-route tool and run the Map stage (after importing the \$DESIGN\_synplify.lpf file).
6. Run the PAR and BIT stages in ispLEVER.



# Optimizing Xilinx Designs

This section contains tips for working with Xilinx designs:

- [Designing for Xilinx Architectures](#), on page 797
- [Specifying Xilinx Macros](#), on page 798
- [Specifying Global Sets/Resets and Startup Blocks](#), on page 800
- [Inferring Wide Adders](#), on page 801
- [Instantiating CoreGen Cores](#), on page 804
- [Packing Registers for Xilinx I/Os](#), on page 807
- [Specifying Xilinx Register INIT Values](#), on page 810
- [Specifying RLOCs](#), on page 819
- [Specifying RLOCs and RLOC\\_ORIGINS with the synthesis Attribute](#), on page 821
- [Using Clock Buffers in Virtex Designs](#), on page 822
- [Working with Clock Skews in Xilinx Virtex-5 Physical Designs](#), on page 824
- [Reoptimizing With EDIF Files](#), on page 826
- [Improving Xilinx Physical Synthesis Performance](#), on page 827
- [Running Post-Synthesis Simulation](#), on page 828
- [Instantiating Special I/O Standard Buffers for Virtex](#), on page 825

For additional Xilinx-specific techniques, see [Xilinx Compile-point Synthesis Flow](#), on page 583, [Working with Xilinx Incremental Flows](#), on page 862, [Working with Gated Clocks](#), on page 464, [Inferring RAMs](#), on page 372, and [Inferring Shift Registers](#), on page 410. Note that some of these features are not available in the Synplify product.

## Designing for Xilinx Architectures

The tips listed here are in addition to the technology-independent design tips described in [Tips for Optimization](#), on page 426.

- For critical paths, attach the `xc_fast` attribute to the I/Os.

- To ensure that frequency constraints from register to output pads are forward annotated to the P&R tools, add default `input_delay` and `output_delay` constraints of 0.0 in the synthesis tool. The synthesis tool forward-annotates the frequency constraints as PERIOD constraints (register-to-register) and OFFSET constraints (input-to-register and register-to-output). The place-and-route tools use these constraints.
- Run successive place-and-route iterations with progressively tighter timing constraints to get the best results possible.
- Specify a UNISIM library using the following syntax:

```
library unisim;
use unisim.vcomponents.all;
```

Remove any other package files with user-defined UNISIM primitives.

## Specifying Xilinx Macros

The synthesis tool provides Xilinx macro libraries that you can use to instantiate components like I/Os, I/O pads, gates, counters, and flip-flops. Using the macros from these libraries allows you to perform a subsequent simulation run without changing your code.

1. To use the Verilog macro library, do the following:
  - Review the library file for the available macros. The Verilog library is `install_dir/lib/xilinx/unisim.v`.
  - Add the `unisim.v` Xilinx macro library file to your project file.
  - Make sure the library is the first in the list of source files..
2. To use a VHDL library, do the following:
  - Review the `unisim.vhd` macro library in the `install_dir/lib/xilinx` directory to check the macros that are available.
  - Add the corresponding library and use clauses to the beginning of the design units that instantiate the macros, as in the following example:

```
library unisim;
use unisim.vcomponents.all;
```

You do not need to add the macro library files to your the source files for your project.

3. Instantiate the macro component in your design.

4. To instantiate an I/O pad with different I/O standards, do the following:
  - Specify the macro library as described in the first two steps.
  - Instantiate the I/O pad component in your design. You can instantiate IBUF, IBUFG, OBUF, OBUFT, and IOBUF components.
  - In the source files, define the generic or parameter values for the I/O standard. Use an IOSTANDARD generic/parameter to specify the I/O standard you want. Refer to the Xilinx documentation for a list of supported IOSTANDARDS. For certain pad types, you can also specify the output slew rate (SLEW) and output drive strength (DRIVE). See [OBUF Instantiation Example, on page 799](#) for an example.

## OBUF Instantiation Example

The following examples show the declaration of OBUF in macro library files:

```
VHDL component OBUF
 generic (
 IOSTANDARD : string := "default";
 SLEW : string := "SLOW";
 DRIVE : integer := 12
);
 port (
 O : out std_logic;
 I : in std_logic;
);
 end component;
 attribute syn_black_box of OBUF : component is true
```

```
Verilog module OBUF(O, I); /* synthesis syn_black_box */
 parameter IOSTANDARD="default";
 parameter SLEW="SLOW";
 parameter DRIVE=12;
 output O;
 input I;
 endmodule
```

To use the macro libraries to instantiate I/O pad types, define the generic/parameter values in the Verilog or VHDL source files. The following examples show how to instantiate OBUF pads with an I/O standard value of LVCMOS2, an output slew value of FAST, and an output drive strength of 24.

```
VHDL Data : OBUF
 generic map (
 IOSTANDARD => "LVCMOS2",
 SLEW => "FAST",
 DRIVE => 24
)
 port map (
 O => o1,
 I => i1
);
```

```
Verilog OBUF Data(.O(o1), .I(i1));
 defparam Data.IOSTANDARD = "LVCMOS2";
 defparam Data.SLEW = "FAST";
 defparam Data.DRIVE = 24;
```

The resulting EDIF file contains the following, which corresponds to the instantiations:

```
(instance (
 rename dataZ0 "data")
 (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
 (property iostandard (string "LVCMOS2"))
 (property slew (string "FAST"))
 (property drive (integer 24))
)
```

## Specifying Global Sets/Resets and Startup Blocks

The global set/reset (GSR) resource is a pre-routed signal that goes to the set or reset input of each flip-flop in your design. Using this resource instead of general routing for a set or reset signal can have a significant positive impact on the routability and performance of your design. The synthesis tools infer this resource automatically in most cases, but you can also specify access to the GSR resource with a Xilinx startup block.

1. Specify access to the GSR as follows:
  - To automatically use the GSR if needed, eslect Implementation Options -> Device, and set Force GSR Usage to auto. With this setting, the tool automatically determines if it needs to use the GSR.
  - To use the GSR, set Force GSR Usage to yes.
  - If you do not want to use the GSR, set Force GSR Usage to no.
2. For Xilinx XC designs, specify global sets/resets (GSR) as follows:
  - For a design with a single GSR, the synthesis tools automatically connect it to a startup block, even if the flip-flops have no sets/resets specified. If you need to change this setting, select Project-> Implementation Options ->Device, and set Force GSR Usage to no. With this setting, all flip-flops must have a set or reset and the set or reset must be the same before GSR is used.
  - For designs with multiple GSRs, the synthesis tool does not automatically create a startup block for GSR. If you still want to use one of the set or reset signals for GSR, you must instantiate a STARTUP\_GSR component manually, as described in the next step.

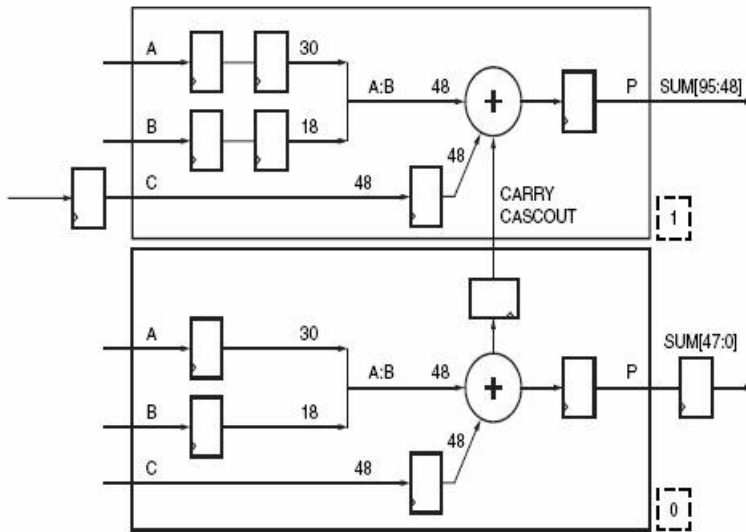
For XC4000 technologies, the synthesis tool forces the creation of a startup block to access the GSR resource, if it is appropriate for your design.

3. To instantiate a start-up block manually, do the following:
  - Go to the *install\_dir/lib/xilinx* directory and locate the appropriate Xilinx startup blocks in either the Verilog (*unisim.v*) or VHDL (*unisim.vhd*) format.
  - Instantiate the startup block component in your design. Where there is more than one component listed, you can use them independently, because the blocks are merged to form a single block in the EDIF file.

## Inferring Wide Adders

For Virtex-5 and Virtex-6 designs, you can map wide adder/subtractor structures to DSP48Es. Xilinx architectures let you use cascading DSP48Es and the CARRYCASCOUT pin to support a structure with up to three pipeline registers with different synchronous control signals. It supports two or three signed/unsigned inputs (with carry/borrow).

The following shows the implementation of wide adders with one pipelined register and no pipelined registers as DSP48Es:



To automatically map to DSP48Es in the synthesis tools, do the following:

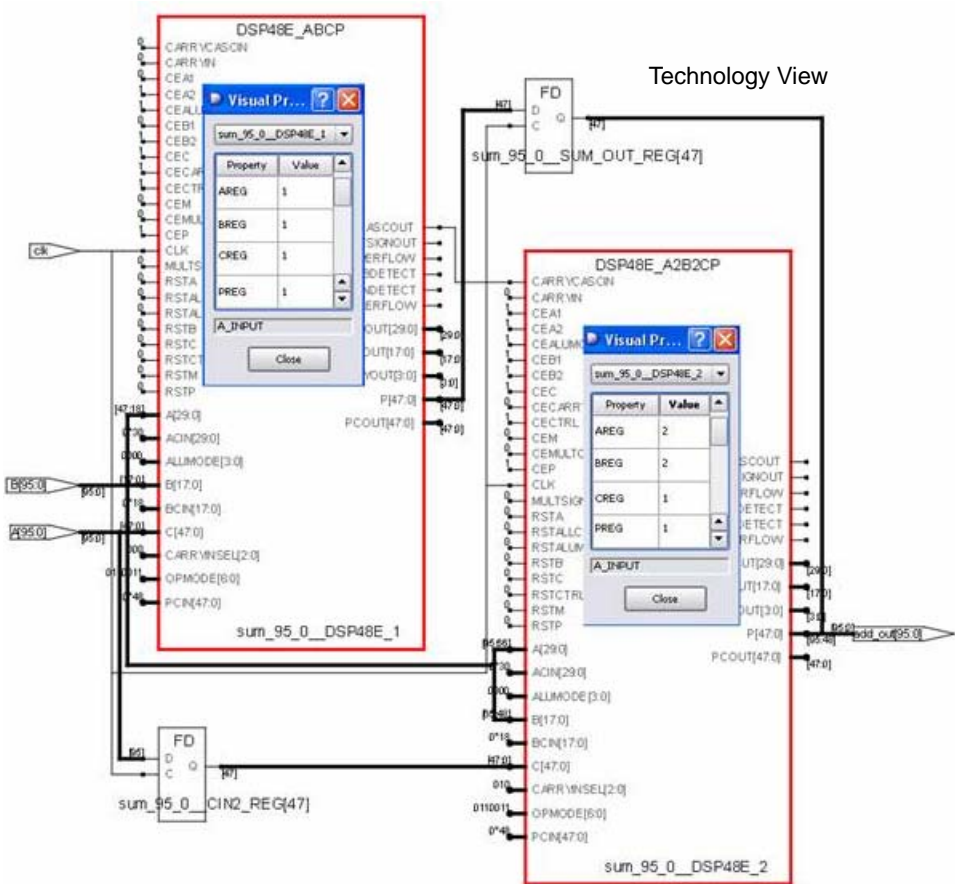
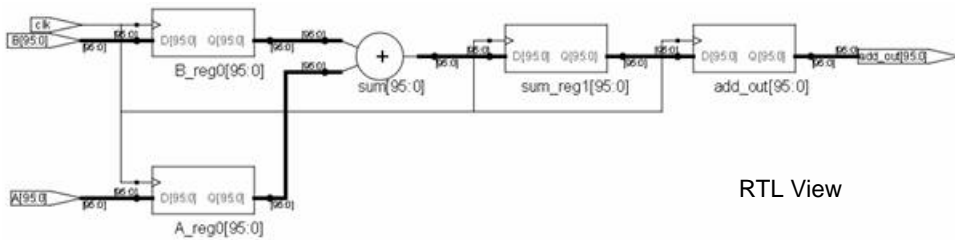
1. Make sure the structure you want to map conforms with these rules:
  - The adder/subtractor does not have more than 96 bits.
  - All registers share the same control signals (enables, clocks, reset). Registers with different control signals are mapped to the DSP48E, but they are kept outside the DSP48E.
  - The adder does not have a 48-bit input and a 49-bit output.
2. Set `syn_dspstyle` to `dsp48`.

You must set this attribute, or the tool does not infer a DSP48E. See [syn\\_dspstyle Attribute, on page 973](#) for the syntax for this attribute.

3. Synthesize the design.

If your structure has less than three pipelined registers, you see an advisory message in the log file, because three pipelined registers give the best performance.

The following is an example of how the synthesis tool maps an adder->register->register structure with 96-bit signed input and output to a DSP48E:



## Instantiating CoreGen Cores

Predesigned IP cores save on design effort and improve performance. The process for handling IP cores is slightly different for CoreGen and Virtex PCI cores. The following procedure describes how to instantiate a CoreGen module. For Virtex PCI cores, see [Instantiating Virtex PCI Cores, on page 805](#).

1. Use the Xilinx CORE generator to create structural EDIF netlists and generate timing and resource usage information for synthesis.
  - For legacy cores, generate a single flat .edf netlist file.
  - For newer cores, generate a top-level flat .edn or .edf netlist file that instantiates .ndf files for each hierarchical level in the design.
2. Open the synthesis software, and add the generated files (.edf only for legacy cores; .edn or .edf and .ndf for newer cores) to your project.
3. Define the core as a black box by adding the `syn_black_box` attribute to the module definition line, or by using the Coregen .v file. The following is an example of the attribute:

```
module ram64x8(din, addr, we, clk, dout)/* synthesis syn_black_box
*/;
 input[7:0] din;
 input [5:0] addr;
 input we, clk;
 output [7:0] dout;
endmodule;
```

4. Make sure the bus format matches the bus format in the core generator, using the `syn_edif_bit_format` and `syn_edif_scalar_format` directives if needed.

```
module ram64x8(din, addr, we, clk, dout)
 /* synthesis syn_black_box syn_edif_bit_format = "%u<%i>"
 syn_edif_scalar format = "%u" */;
```

5. Instantiate the black box in the module or architecture.

```
ram64x8 r1(din, addr, we, clk, dout);
```

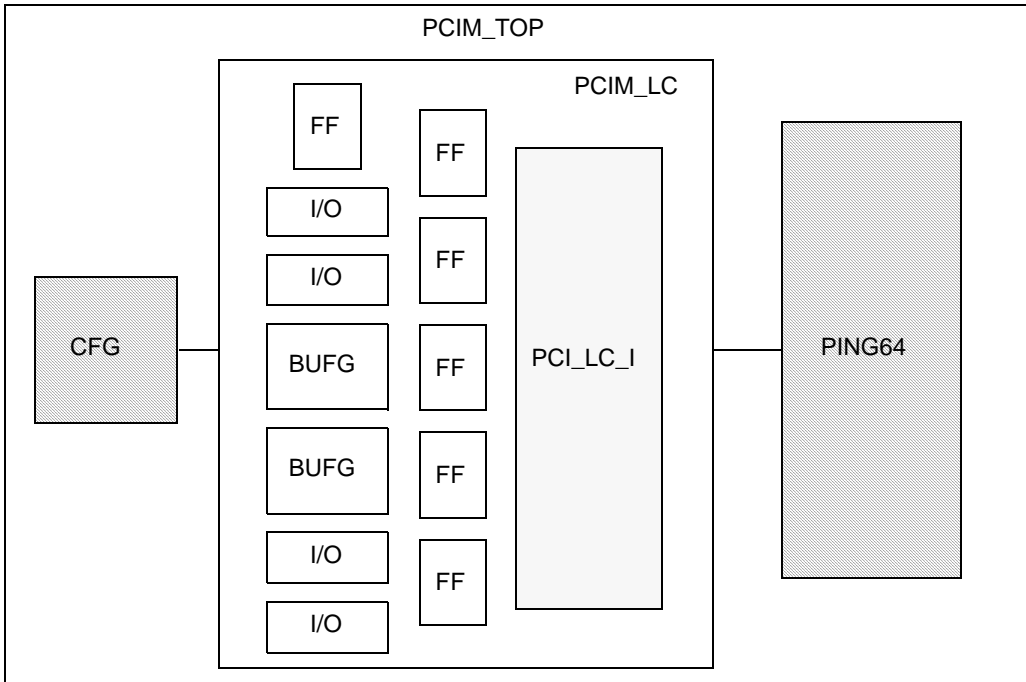
6. Synthesize the design.

If you supplied structural EDIF netlists, the software optimizes the design based on the information in the structural netlists. The generated reports contain the optimization information.



## Instantiating Virtex PCI Cores

For Virtex PCI cores, you can use either a top-down or bottom-up methodology. This figure shows a design that is used in the explanations of both methodologies, below.



### Bottom-Up Method

The bottom-up method synthesizes lower-level modules first. The synthesized modules are then treated as black boxes and synthesized at the next level. The following procedure refers to the figure shown above.

1. Synthesize the user-defined application (PING64) by itself.
  - Make sure that the Disable I/O Insertion option is on.
  - Specify the `syn_edif_bit_format = "%u<%i>"` and `syn_edif_scalar_format = "%u"` attributes. This ensure that the

EDIF bus names match the Xilinx upper-case, angle bracket style bus names and the Xilinx upper-case net names, respectively.

The software generates an EDIF file for this module.

2. Synthesize the top-level module that contains the PCI core, with the Disable I/O Insertion option enabled and the EDIF naming attributes described in the previous step. Use the following files to synthesize:
  - The top-level module (PCIM\_LC) file, with the PCI core (PCI\_LC\_I) declared as a black box with the `syn_black_box` attribute.
  - A black box file for the core (PCI\_LC\_I), that only contains information about the PCI core ports. This file is the source file that is generated for simulation, not the `.ngo` file.
  - The appropriate synthesis Virtex file (*install\_dir/lib/xilinx*) that contains module definitions of the I/O pads in the top-level module, PCIM\_LC.

The software generates an EDIF file for this module.

3. Synthesize the top level (PCIM\_TOP) with Disable I/O Insertion off. Use the following files:
  - The source file for CFG.
  - A black box file for PING64.
  - A black box file for PCIM\_LC.
  - A top-level file that contains black box declarations for PING64 and PCIM\_LC.

The software generates an EDIF file for the top level.

4. Place and route using the Xilinx `.ngo` file for the core, and the three EDIF files generated from synthesis: one for each of the modules PING64 and PCIM\_LC, and the top-level EDIF file. Select the top-level EDIF file when you run place-and-route.

## Top-down Methodology

The top-down method instantiates user application blocks and synthesizes all the source files in one synthesis run. This method can result in a smaller, faster design than with the bottom-up method, because the tool can do cross-boundary optimizations. The following procedure refers to the design shown in the previous figure.

1. Create your own configuration file for your application model (CFG).
2. Edit the top-level source file to do the following:
  - Instantiate your application block (PING64) in the top-level source file.
  - Add the ports from your application.
3. Add the appropriate synthesis Virtex file (*install\_dir/lib/xilinx*) to the project. This file contains module definitions of the I/O pads in the PCIM\_LC module.
4. Specify the top-level file in the project.
5. Synthesize your design with the following files:
  - Virtex module definition file (previous step)
  - Source files for top-level design, user application (PING64), PCIM\_LC, and CFG
  - Simulation wrapper file for PCI core

The software generates an EDIF file for the top level.

6. Place and route the design using the top-level EDIF file from synthesis and the Xilinx .ngo file for the PCI core.

## Packing Registers for Xilinx I/Os

When a register drives an input or output, you might want to pack it in an IOB instead of a CLB, as in these cases:

- The chip interfaces with another, and you have to minimize the register-to-output or input-to-register delay.
- You have limited CLB resources, and packing the registers in an IOB can free up some resources.

To pack registers in an IOB, you set the `syn_useioff` attribute.

1. To globally embed all the registers into IOBs, attach the `syn_useioff` attribute to the module in one of these ways:
  - Add the attribute in the SCOPE window, attaching it to the module, architecture, or the top level. Check the Enable box, set the Attribute column to `syn_useioff`, the Object column to `<global>`, and the attribute value to 1. The constraint file syntax looks like this:

```
define_global_attribute syn_useioff 1
```

- To add the attribute in the Verilog source code, add this syntax to the top level:

```
module global_test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

- To add the attribute in the VHDL source code, add this syntax to the top level architecture declaration:

```
architecture rtl of global_test is
 attribute syn_useioff : boolean;
 attribute syn_useioff of rtl : architecture is true;
```

For details about attaching attributes using the SCOPE interface and in the source code, see [Entering Attributes and Directives, on page 304](#).

When set globally, all boundary registers and (OE) registers associated with the data registers are marked with the Xilinx IOB property. This property is forward annotated in the EDIF netlist and used by the Xilinx place-and-route tools to determine how the registers are packed. All marked registers are packed in the corresponding IOBs.

2. To apply `syn_useioff` to individual registers or ports, use one of these methods:
  - Add the attribute in the SCOPE window, attaching it to the ports you want to pack, and set the attribute value to 1. The resulting constraint file syntax looks like this:

```
define_attribute {p:q[3:0]} syn_useioff 1
```

- To add the attribute in the Verilog source code, add this syntax:

```
module test is (d, clk, q);
 input [3:0] d;
 input clk;
 output [3:0] q /* synthesis syn_useioff = 1 */;
 reg q;
```

- To add the attribute in the VHDL source code, add syntax as shown inside the entity for the local port:

```
entity test is
 port (d : in std_logic_vector(3 downto 0);
 clk : in std_logic
 q : out std_logic_vector(3 downto 0);
 attribute syn_useioff : boolean;
 attribute syn_useioff of q : signal is true;
end test;
```

The software attaches the IOB property as described in the previous step, but only to the specified flip-flops. Packing for ports and registers without the attribute is determined by timing preferences. If a register is to be packed into an IOB, the IOB property is attached and forward annotated. If it is to be packed into a CLB, the IOB property is not forward annotated.

In Virtex designs where the synthesis software duplicates OE registers, setting the `syn_useioff` attribute on a boundary register only enables the associated OE register for packing. The duplicate is not packed, but placed in a CLB. The packed registers are used for data path, and the CLB registers are used for counter implementation.

In Virtex designs where a shift register is at a boundary edge and the `syn_useioff` attribute is enabled, the software extracts only the initial or final SRL16 shift register from the LUT for packing. The shift register that is implemented in the technology view is smaller because of the extraction.

3. If you set multiple `syn_useioff` attributes at different levels of the design, the tool uses the most specific setting (highest priority).

This table summarizes `syn_useioff` priority settings, from the highest priority (register) to the lowest (global):

| I/O Type | <code>syn_useioff</code> Value | Description                                                                           |
|----------|--------------------------------|---------------------------------------------------------------------------------------|
| Register | 1                              | Packs registers into the I/O pad cells, overriding port or global specifications.     |
|          | 0                              | Does not pack registers into I/O pad cells, overriding port or global specifications. |

| <b>I/O Type</b> | <b>syn_useioff Value</b> | <b>Description</b>                                                               |
|-----------------|--------------------------|----------------------------------------------------------------------------------|
| Port            | 1                        | Packs registers into the I/O pad cells, overriding any global specification.     |
|                 | 0                        | Does not pack registers into I/O pad cells, overriding any global specification. |
| Global          | 1                        | Packs registers into the I/O pad cells.                                          |
|                 | 0                        | Does not pack registers into I/O pad cells.                                      |

## Specifying Xilinx Register INIT Values

You can specify initial values for registers so that the RTL, gate-level simulation, and the final implementation results match. You can specify INIT values for registers either with the HDL initialization specification built into Verilog or VHDL, or by adding the synthesis attribute. You can then pass the values to the Xilinx P&R tools .

Both methods are described in the following procedure, but the HDL specification method is recommended.

1. To ensure that the register is not optimized away during synthesis, set the `syn_preserve` directive on the register in the source code.

Use this directive even if you define the INIT values with a constraint in the `.sdc` file. If you do not have this directive, the register can be removed during optimization.

2. To set a register value using the HDL initialization feature, use the following syntax:

```
HDL Initialization reg myreg=0;
 initial myreg=0; (Verilog only)
```

For example:

```
Verilog HDL Initialization reg error_reg = 1'b0;
 reg [7:0] address_reg = 8'hff;
```

```
VHDL HDL Initialization signal tmp: std_logic = '0';
```

This is the preferred method to pass INIT values to the Xilinx place-and-route tools.

- To set a register value using the synthesis attribute, add the attribute to the register in the source code or the constraint file, and specify the INIT value as a string:

---

```
Verilog reg [3:0] rst_cntr /* synthesis INIT="1" */;
```

---

```
VHDL attribute INIT: string;
attribute INIT of rst_cntr : signal is "1";
```

---

```
SDC define_attribute {i:rst_cntr} INIT {"1"}
```

---

Xilinx ISE 8.2sp3 and later versions require that the INIT value be a string rather than an integer. For code examples, see [INIT Values, on page 1178](#) in the *Reference Manual*.

- To specify different INIT values for each register bit on a bus, do the following:
  - Set `syn_preserve` on the register as described in step 1, so that it is not optimized away. You can now either use the HDL specification or set an attribute.
  - To specify the values using the HDL specification, use the syntax as shown in the following examples:

**Verilog HDL Bus Initialization** `reg [7:0] address_reg = 8'hff;`

---

**VHDL HDL Bus Initialization** `signal q: std_logic_vector  
(11 downto 0) := X"755";`

---

- To specify the value with the INIT attribute in the `.sdc` constraint file, set INIT values for the individual register bits on the bus. Specify the register using the `i:` prefix, with periods as hierarchy separators.

The following specifies INIT values for individual bits of `rst_cntr`, which is part of the `init_attrver` module, under the top-level module:

```
define_attribute {i:init_attrver.rst_cntr[0]} INIT {"0"}
define_attribute {i:init_attrver.rst_cntr[1]} INIT {"1"}
define_attribute {i:init_attrver.rst_cntr[2]} INIT {"0"}
define_attribute {i:init_attrver.rst_cntr[3]} INIT {"1"}
```

## 5. Synthesize the design.

The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist. If the register is an asynchronous output register with an initial value, the mapper preserves the initial value and packs the register into the Block RAM.

## Inserting Xilinx I/Os and Specifying Pin Locations

By default, the synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). You can change this by enabling **Disable IO Insertion** in the **Device** tab of the **Implementation Options** dialog box. You can also insert I/Os manually by instantiating them.

Whether you use the automatic or manual method, you can specify pin locations for the I/Os with the `xc_loc` attribute. By default, or if no location is specified, the Xilinx tool assigns pin locations automatically.

The following provide details:

- [Assigning Pin Locations for Automatically Inserted Xilinx I/Os](#), on page 812
- [Manually Inserting Xilinx I/Os in Verilog](#), on page 815
- [Manually Inserting Xilinx I/Os in VHDL](#), on page 817

### Assigning Pin Locations for Automatically Inserted Xilinx I/Os

The synthesis tool automatically inserts the I/Os (unless you have checked **Disable IO Insertion** in the **Device** tab of the **Implementation Options** dialog box). The following procedure shows you how to assign pin locations for automatically inserted I/Os in a Verilog or VHDL design.

1. Create a new top-level module or entity and instantiate it in your Verilog or VHDL design.

This module/entity holds I/O placement information. Creating this lets you keep your vendor-specific information separate from the rest of your design. Your original design remains technology-independent.

For example, this is a Verilog counter definition:



```

module cnt4 (cout, out, in, ce, load, clk, rst);
// Counter definition
endmodule

```

You create a top-level module that instantiates your design:

```

module cnt4_xilinx (cout, out, in, ce, load, clk, rst);

```

2. If you do not want to specify locations, specify the inputs or outputs as usual. The following is an example of Verilog inputs in the top-level module:

```

input ce, load, clk, rst;

```

The Xilinx place-and-route tool automatically places these inputs.

3. Optionally, specify I/O locations in the new top-level module, by setting the `xc_loc` attribute.

You can specify the `xc_loc` attribute in the Attribute panel of the SCOPE spreadsheet, as shown below.

|   | Enabled                             | Object Type | Object   | Attribute | Value   | Val Type | Description    |
|---|-------------------------------------|-------------|----------|-----------|---------|----------|----------------|
| 2 | <input checked="" type="checkbox"/> | port        | in1[2:0] | xc_loc    | P2,P3,P | string   | Port placement |
| 3 | <input checked="" type="checkbox"/> | port        | p:out1   | xc_loc    | R1      | string   | Port placement |
| 4 | <input checked="" type="checkbox"/> |             |          |           |         |          |                |

Attributes

Alternatively, you can specify it in the HDL files, as described in [Manually Inserting Xilinx I/Os in Verilog, on page 815](#) and [Manually Inserting Xilinx I/Os in VHDL, on page 817](#). See [xc\\_loc Attribute, on page 1165](#) in the *Reference Manual* for syntax details.

The following Verilog code includes `xc_loc` attributes that specify the following locations:

- cout at A1
- out in the top left (TL) of the chip
- in[3] at P20, in[2] at P19, in[1] at P18, and in[0] at P17

```

output cout /* synthesis xc_loc="A1" */;
output [3:0] out /* synthesis xc_loc="TL" */;
input [3:0] in /* synthesis xc_loc="P20,P19,P18,P17" */;

```

4. Instantiate the top-level module or entity with the placement information you specified in your design. For example:

```
cnt4 my_counter (.cout(cout), .out(out), .in(in),
 .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule
```

5. Synthesize the design.

The synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). The Xilinx place-and-route tool automatically selects locations for I/Os with no `xc_loc` attribute defined. If you specified `xc_loc` settings, they are honored.

## VHDL Automatic I/O Insertion Example

```
library synplify;
entity cnt4 is
 port (cout: out bit;
 output: out bit_vector (3 downto 0);
 input: in bit_vector (3 downto 0);
 ce, load, clk, rst: in bit);
end cnt4;

architecture behave of cnt4 is
begin
 -- Behavioral description of the counter.
end behave;

-- New top level entity, created specifically
-- to place I/Os for Xilinx. This entity is typically
-- in another file, so that your original
-- design stays untouched and technology independent.

entity cnt4_xilinx is
 port (cout: out bit;
 output: out bit_vector (3 downto 0);
 input: in bit_vector (3 downto 0);
 ce, load, clk, rst: in bit);

 -- Place a single I/O for cout at location A1.
 attribute xc_loc : string;
 attribute xc_loc of cout: signal is "A1";

 -- Place all bits of "output" in the
 -- top-left of the chip.
 attribute xc_loc of output: signal is "TL";
```

```
-- Place input(3) at P20, input(2) at P19,
-- input(1) at P18, and input(0) at P17
attribute xc_loc of input: signal is "P20, P19, P18, P17";

-- Let Xilinx place the rest of the inputs.
end cnt4_xilinx;

-- New top level architecture instantiates your design.
architecture structural of cnt4_xilinx is
-- Component declaration for your entity.

component cnt4
 port (cout: out bit;
 output: out bit_vector (3 downto 0);
 input: in bit_vector (3 downto 0);
 ce, load, clk, rst: in bit);
end component;
begin

-- Instantiate your VHDL design here:
my_counter: cnt4 port map (cout, output, input,
 ce, load, clk, rst);
end structural;
```

## Manually Inserting Xilinx I/Os in Verilog

To insert a Xilinx I/O manually, you must instantiate a black box macro for that I/O from the Xilinx library file. You can then choose to assign it a location or have the Xilinx tool automatically select one for it.

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the `install_dir/lib/xilinx/unisim.v` macro library file to the *top* of the source files list for your project.
2. Create instances of I/Os by instantiating a black box in your Verilog source code.

These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To insert an I/O manually and specify pin locations, do the following:

1. Create a new top-level module and instantiate your Verilog design.

2. Add the `install_dir/lib/xilinx/unisim.v` macro library file to the top of the source files list for your project.
3. Create instances of I/Os by instantiating a black box in your Verilog source code.
4. Specify I/O locations by adding the `xc_loc` attribute to the I/Os.

See [Verilog Manual I/O Insertion Example, on page 816](#) for an example of the code. The Xilinx tool honors any locations assigned with the `xc_loc` attribute, and automatically selects locations for any remaining I/Os without definitions.

## Verilog Manual I/O Insertion Example

```

module cnt4 (cout, out, in, ce, load, clk, rst);
 /* Your counter definition goes here, */
endmodule
/* Create a top level to place I/Os specifically
 for Xilinx. Any top level pins which do not have
 I/Os will be automatically inserted */
module cnt4_xilinx(cout, out, in, ce, load, clk, rst);
 output [3:0] out;
 output cout;
 input [3:0] in;
 input ce, load, clk, rst; wire [3:0] out_c, in_c;
 wire cout_c;

 /* The xc_loc attribute can be added right after the
 instance name like that shown below, or right before
 the semicolon. */

 IBUF i3 /* synthesis xc_loc="P20" */ (.O(in_c[3]), .I(in[3]));
 IBUF i2 /* synthesis xc_loc="P19" */ (.O(in_c[2]), .I(in[2]));
 IBUF i1 /* synthesis xc_loc="P18" */ (.O(in_c[1]), .I(in[1]));
 IBUF i0 /* synthesis xc_loc="P17" */ (.O(in_c[0]), .I(in[0]));

 OBUF o3 /* synthesis xc_loc="TL" */ (.O(out[3]), .I(out_c[3]));
 OBUF o2 /* synthesis xc_loc="TL" */ (.O(out[2]), .I(out_c[2]));
 OBUF o1 /* synthesis xc_loc="TL" */ (.O(out[1]), .I(out_c[1]));
 OBUF o0 /* synthesis xc_loc="TL" */ (.O(out[0]), .I(out_c[0]));

 OBUF cout_p /* synthesis xc_loc="BL" */ (.O(cout), .I(cout_c));

 cnt4 it(.cout(cout_c), .out(out_c), .in(in_c),
 .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule

```

## Manually Inserting Xilinx I/Os in VHDL

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the corresponding library and use clauses to the beginning of your design units that instantiate the macros.

```
library unisim;
use unisim.vcomponents.all;
```

The Xilinx `unisim.vhd` macro library is always visible in the synthesis tool, so do not add this library file to the source files list for your project. To see which design units are available, use a text editor to view the file located in the `install_dir/lib/xilinx` directory. Do not edit this file in any way.

2. Create instances of I/Os by instantiating a black box in your Verilog source code.

These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To insert an I/O manually and specify pin locations, do the following:

1. Create a new top-level module and instantiate your VHDL design.
2. Instantiate the Xilinx I/Os.
3. Add the appropriate library and use clauses to the beginning of design units that instantiate the I/Os.

```
library unisim;
use unisim.vcomponents.all;
```

See the source code in [VHDL Manual I/O Insertion Example, on page 818](#) for an example.

4. To specify I/O locations, add the `xc_loc` attribute to the I/O instances for which you want to specify the locations.
5. If you leave out the `xc_loc` attribute, the Xilinx place-and-route tool will choose the locations.

## VHDL Manual I/O Insertion Example

The following example is a behavioral D flip-flop with instantiated data input I/O. The other ports will have synthesized I/Os.

```
library ieee, synplify;
use synplify.attributes.all;
use ieee.std_logic_1164.all;
-- Library and use clauses for access to the Xilinx Macro Library.
library unisim;
use unisim.vcomponents.all;

entity place_example is
 port (q: out std_logic;
 d, clk: in std_logic);
end place_example;

architecture behave of place_example is
 signal dz: std_logic;

 attribute xc_loc of I1: label is "P3";
begin
 I1: IBUF port map (I=>d,O=>dz);

 process (clk) begin
 if rising_edge(clk) then
 q<=dz;
 end if;
 end process;

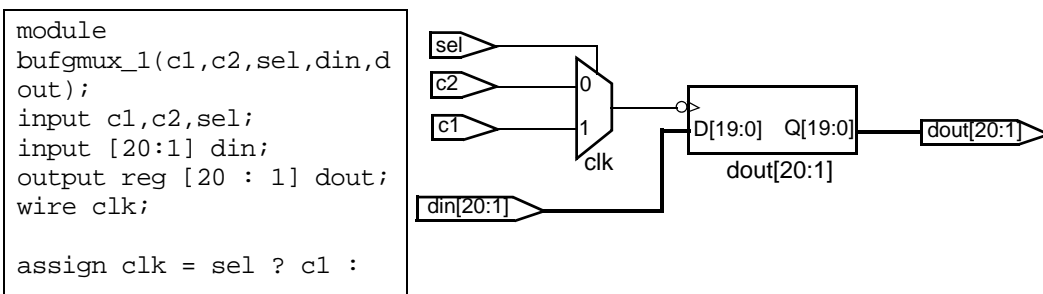
end behave;
```

## Working with Xilinx Buffers

By default, the synthesis tools do not automatically infer Xilinx buffers. If you want the tools to infer Xilinx buffers, you must use attributes, as described below.

1. To infer BUFGMUX components, do the following:
  - Attach the `syn_insert_buffer` attribute to the mux instance. If you need information on how to do this, see [Entering Attributes and Directives, on page 304](#).
  - Set the attribute value to `bufgmux`. When you set this value, the tool infers a BUFGMUX\_1 if the muxed clock operates on the negative edge;

otherwise it infers a BUFGMUX. If you do not specify this value, by default the tool infers the LUT that drives the BUFG.



For details about the `syn_insert_buffer` syntax, see [syn\\_insert\\_buffer Attribute](#), on page 1014 in the *Reference Manual*

2. To infer IBUFDS, IBUFGDS, OBUFDS, OBUFTDS, and IOBUFDS differential buffers, do the following:
  - Attach the `syn_diff_io` attribute to the inputs of the buffer.
  - Set the value to 1 or true.

For details about the `syn_diff_io` syntax, see [syn\\_diff\\_io Attribute](#), on page 968 in the *Reference Manual*.

## Specifying RLOCs

RLOCs are relative location constraints. They let you control placement in critical sections, thus improving performance. You specify RLOCs using three attributes, `xc_map`, `xc_rloc`, and `xc_uset`. As with other attributes, you can define them in the source code, or in the SCOPE window.

You can also specify RLOCs directly, as described in [Specifying RLOCs and RLOC\\_ORIGINs with the synthesis Attribute](#), on page 821.

1. Create the modules you want to constrain, and specify the kind of Xilinx primitive you want to map them to, using the `xc_map` attribute. The modules can have only one output.

| Family                        | xc_map Value | Max. Module Inputs |
|-------------------------------|--------------|--------------------|
| XC4000, Spartan families      | fmap         | 4                  |
|                               | hmap         | 3                  |
| Virtex and Spartan-3 families | lut          | 4                  |

This Verilog example shows a 4-input Spartan XOR module:

```
module fmap_xor4(z, a, b, c, d) /* synthesis xc_map=fmap*/ ;
output z;
input a, b, c, d;
assign z = a ^ b ^c ^d;
endmodule
```

This is the equivalent VHDL example:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fmap_xor4 is
 port (a: in std_logic;
 b: in std_logic;
 c: in std_logic;
 d: in std_logic
);
end fmap_xor4;

architecture rtl offmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl: architecture is "fmap";
begin
 z <= a xor b xor c xor d;
end rtl;
```

2. Instantiate the modules you created at a higher hierarchy level.
3. Group the instances together (xc\_uset attribute) and specify the relative locations of instances in the group with the xc\_rloc attribute.

This example shows the Verilog code for the top-level CLB that includes the 4-input module in the previous example:



```

module clb_xor9(z, a) ;
output z;
input [8:0] a;
wire x03, x47;
//Code for XC4000 or Spartan
fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
 (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
 (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
 (z, z03, z47, a[8]);
//Code for Virtex differs because it includes the slice
fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
 (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
 (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.S1" */
 (z, z03, z47, a[8]);endmodule

```

4. Create a top-level design and instantiate your design.

## Specifying RLOCs and RLOC\_ORIGINS with the synthesis Attribute

You can specify RLOCs and RLOC\_ORIGINS with the synthesis attribute, and then pass them to the Xilinx P&R tools. Alternatively, you can specify RLOCs using the three attributes described in [Specifying RLOCs, on page 819](#).

1. In the source code, use the synthesis attribute to specify the RLOC and RLOC\_ORIGIN values:

```

Verilog /* synthesis RLOC_ORIGIN="X0Y2" RLOC="X0Y0" */;

VHDL attribute RLOC_ORIGIN : string;
 attribute RLOC_ORIGIN of behave : architecture is "X0Y2";
 attribute RLOC : string;
 attribute RLOC of q : signal is "X0Y0";

```

For code examples, see [RLOC Constraints, on page 1179](#) in the *Reference Manual*.

2. To specify different RLOC and RLOC\_ORIGIN values for bits on a bus, do the following:

- Specify the RLOC\_ORIGIN for the Verilog module or VHDL architecture in the source file. See step 1 for the syntax.
- Define RLOCs for the individual register bits as constraints in the .sdc file. Do not define RLOCs for individual bits in the source code, or you will get a Xilinx ISE error.

```
define_attribute {i:tmp[0]} RLOC {"X3Y0"}
define_attribute {i:tmp[1]} RLOC {"X2Y0"}
define_attribute {i:tmp[2]} RLOC {"X1Y0"}
define_attribute {i:tmp[3]} RLOC {"X0Y0"}
```

### 3. Synthesize the design.

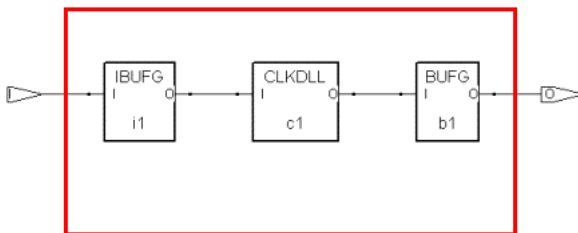
The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist.

## Using Clock Buffers in Virtex Designs

The software can infer a buffer called BUFGDLL that includes the CLKDLL primitive. BUFGDLL consists of an IBUFG followed by a CLKDLL (Clock Delay Locked Loop) followed by a BUFG. To use this CLKDLL primitive, you must specify the xc\_clockbuftype attribute. The following steps show you how to add the attribute in SCOPE or the HDL files.

1. To specify the attribute in the SCOPE window, use the procedure described in [Specifying Attributes Using the SCOPE Editor, on page 307](#) to add the xc\_clockbuftype attribute to a port.

The software infers a buffer as shown in the following figure.



The output EDIF netlist contains text like the following:

```
(instance clk_ibuf (viewRef PRIM (cellRef BUFGDLL (libraryRef VIRTEX)))
```

2. To specify the attribute in Verilog, add the attribute as shown in this example.

```
module test(d, clk, rst, q);
input [1:0] d;
input clk /* synthesis xc_clockbuftype = "BUFGDLL" */, rst;
output [1:0] q;
//other coding
```

3. To specify the attribute in VHDL, add the attribute as shown in this example.

```
entity test_clkbuftype is
 port (d: in std_logic_vector(3 downto 0);
 clk, rst : in std_logic;
 q : out std_logic_vector(3 downto 0)
);
attribute xc_clockbuftype of clk : signal is "BUFGDLL";
end test_clkbuftype
```

## Working with Clock Skews in Xilinx Virtex-5 Physical Designs

The Synplify Premier software version 9.0.1 and later supports clock skews for Virtex-5 devices, and the SRM clock insertion delay property (Clock input arrival\_time). Clock insertion delay models are included for the following components:

- DCM
- BUFG
- BUFR
- BUFIO
- Flip-flops generating clocks

This feature ensures that cross-clock paths are compared correctly. Also, it has a large impact on timing constraints for I/O paths, since any clock delay will be added to the output delay and subtracted from the setup delay. This results in improved timing correlation between the Synplify Premier software and Xilinx timing.

### Example1: Calculating Slack with Clock Skew

Clock skew is utilized to calculate the slack in the following Virtex-5 example:

- Source DCM (clock insertion delay = 0.000ns)
- Load IBUFG (clock insertion delay = 4.157ns)

```
Requested Period: 5.000
- (Setup Time): 0.004
+ (Clock Delay at Ending Point): 4.157
+ (Clock Latency at Ending Point): 0.000
= Required Time: 9.153
- (Propagation Time): 0.746
- (Clock Latency at Starting Point): 0.000
= Slack (non-critical): 8.407
```

## Example 2: Calculating Slack Using Clock Skew

Clock skew is utilized to calculate the slack in the following Virtex-5 example:

- Source IBUFG (clock insertion delay = 4.157ns)
- Load DCM (clock insertion delay = 0.000ns)

```

Requested Period: 5.000
 - (Setup Time): 0.004
+ (Clock Latency at Ending Point): 0.000
 = Required Time: 4.996
 - (Propagation Time): 0.745
- (Clock Delay at Starting Point): 4.157
- (Clock Latency at Starting Point): 0.000
 = Slack (critical): 0.094

```

The Synplify Premier software does not automatically forward annotate constraints for derived clocks. Therefore, a clock generated from a set of flip-flops and logic requires you to add a constraint in the UCF file. As a recommendation, derive the clock period the same as the original clock and add a 2-cycle multicycle path from the clock to itself. Better solutions will be provided in the future.

## Instantiating Special I/O Standard Buffers for Virtex

The software supports all the I/O Virtex standards, like HSTL\_\*, CTT, AGP, PC133\_\*, PC166\_\*, etc. You can either instantiate these primitives directly, or specify them with the `xc_padtype` attribute.

1. To instantiate I/O buffers, use code like the following to specify them.

```

module inst_padtype(a, b, clk, rst, en, bidir, q) ;
input [0:0] a, b;
input clk, rst, en;
inout bidir;
output [0:0] q;

reg [0:0] q_int;
wire a_in, q_in;
IBUF_AGP i1 (.O(a_in), .I(a)) ;
IOBUF_CTT i2 (.O(q_in), .IO(bidir) , .I(q_in), .T(en)) ;
OBUF_F_12 o1 (.O(q), .I(q_in)) ;

```

```
always @(posedge clk or posedge rst)
 if (rst)
 q_int = 1'b0;
 else
 q_int = a & b;

endmodule
```

2. To specify the I/O buffers with an attribute, add the attribute in the SCOPE window (refer to [Specifying Timing Constraints, on page 219](#) for details) or in the source code, as the following example illustrates.

```
module inst_padtype(a, b, clk, rst, en, bidir, q) ;
 input [0:0] a /* synthesis xc_padtype = "IBUF_AGP" */, b;
 input clk, rst, en;
 inout bidir /* synthesis xc_padtype = "IOBUF_CTT" */;
 output [0:0] q /* synthesis xc_padtype = "OBUF_F_12" */;

 reg [0:0] q_int;

 assign q = bidir;
 assign bidir = en ? q_int : 1'bz;
 always @(posedge clk or posedge rst)
 if (rst)
 q_int = 1'b0;
 else
 q_int = a_in & b;
endmodule
```

## Reoptimizing With EDIF Files

You can resynthesize an EDIF file to refine and optimize your design further.

1. Make sure your design conforms to these rules:
  - The design must not have mixed language files.
  - The name of the EDIF file matches the module name.
2. Create a project and add the EDIF file to the design.
3. Specify the EDIF as the top-level design.
  - Click Implementation Options and go to the Verilog or VHDL tab.

- Enter the module name in the Top Level Module/Entity field. If your module is not in the work library, specify the library first:

`<library>.<module>`

- Click OK.

4. Set any other options you want and resynthesize your design.

## Improving Xilinx Physical Synthesis Performance

The Synplify Premier tool is timing-driven; optimizations depend on timing constraints and are applied until all constraints are met. Therefore, it is very important that you adequately apply timing constraints and not over-constrain the tool. This section includes guidelines for applying constraints.

- Verify the consistency of constraints between synthesis and P&R:
  - Clock constraints
  - Clock-to-clock constraints
  - IO delays
  - IO standard, drive, slew and pull-up/pull-down
  - Multi-cycle and false paths
  - Max-delay paths
  - DCM parameters
  - Register packing into IOB
  - LOC/RLOC constraints on macros (BUFG, DCM, RAMB, DSP, MULT, etc.)
  - LOC/RLOC constraints on instances (Register, LUT, SRL, RAMS, RAMD, etc.)
  - AREA\_GROUP constraints
  - IDELAYCTRL and IDELAY constraints
- Ensure that the final physical synthesis slack is negative, but no more than 10-15% of the clock constraint.
- Check the log file for Pre-placement timing snapshot.  
If it indicates that a clock has positive slack at this point, but in the final results the clock has negative slack, use the `-route` constraint for the clock. This lets you to control the amount of early timing optimizations

for the clock domain. However, large `-route` values can degrade performance. Therefore, to determine the correct `-route` value to use, start with smaller values and increase iteratively. For example, start with half the difference between the estimate and actual slack, or 5% of the clock estimate, whichever is the smallest.

- Experiment with ignoring the relationally-placed macro (RPM) constraints.

RPMs (also known as RLOCs) can negatively affect results. You can compare placement results using the Synplify Premier tool by setting the global attribute `xc_use_rpms` to 0. For details on this attribute, see [xc\\_use\\_rpms Attribute, on page 1188](#) in the *Reference Manual*.

- Ensure placement for I/Os, Block Rams, and DSP48 devices.

This version of the tool uses the Xilinx placer to generate locations for I/Os and block components. To avoid block component placement problems, you need to lock placement. See [Generating a Xilinx Coreloc Placement File, on page 354](#) for information.

## Running Post-Synthesis Simulation

For post-synthesis simulation with a Xilinx design, do the following:

1. Run synthesis as usual.

The run generates a `.vhm` file, which references the synplify library:

```
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;
```

2. Set up the libraries.
  - Create a library called `synplify` and compile `synplify.vhd` into it. The `synplify.vhd` file is located in `install_dir/lib/vhdl_sim`.
  - Create a library called `UNISIM`, and compile the UNISIM simulation library provided by Xilinx into it.
3. Compile the `.vhm` file into work. For example:

```
vcom -work synplify install_dir/lib/vhdl_sim/synplify.vhd
```



## Working with Xilinx Place-and-Route Software

The following procedure shows you how to run the Xilinx place-and-route tool from within the synthesis software.

1. Set the XILINX environment variable to point to your Xilinx software installation directory.
2. Start the synthesis software and open a synthesized design.
3. Start the place-and-route software:
  - To start Xilinx Design Manager, select Options->Xilinx->Start Design Manager.
  - To start Xilinx floorplanner, select Options->Xilinx->Start Floorplanner.
  - To start the ISE tool, select Options->Xilinx->Start ISE Project Navigator.

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 19

# 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 8, \*Inferring High-Level Objects\*](#).

This chapter describes the following:

- [Passing Information to the P&R Tools](#), on page 832
- [Generating Vendor-Specific Output](#), on page 836
- [Invoking Third-Party Vendor Tools](#), on page 838

## Passing Information to the P&R Tools

The following procedures show 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:

- [Specifying Pin Locations](#), on page 832
- [Specifying Locations for Actel Bus Ports](#), on page 833
- [Specifying Macro and Register Placement](#), on page 833
- [Passing Technology Properties](#), on page 834
- [Specifying Padtype and Port Information](#), on page 834

### Specifying Pin Locations

In certain technologies you can specify pin locations that are forward-annotated to the corresponding place-and-route tool. The following procedure shows you how to specify the appropriate attributes. For information about other placement properties, see [Specifying Macro and Register Placement, on page 833](#).

1. Start with a design using one of the following vendors and technologies: Actel, Altera, Xilinx, Lattice, or QuickLogic families.
2. Add the appropriate attribute to the port. For a bus, list all the bus pins, separated by commas. To specify Actel bus port locations, see [Specifying Locations for Actel Bus Ports, on page 833](#).
  - To add the attribute from the SCOPE interface, click the Attributes tab and specify the appropriate attribute and value.
  - To add the attribute in the source files, use the appropriate attribute and syntax. See the *Reference Manual* for syntax details.

| Family | Attribute and Value                               |
|--------|---------------------------------------------------|
| Actel  | syn_loc {pin_number}<br>or<br>alspin {pin_number} |
| Altera | syn_loc {pin_number}                              |

---

|            |                                                                                                                                                                        |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lattice    | <code>loc {pin_number}</code>                                                                                                                                          |
| QuickLogic | <code>ql_placement {pin_number}</code>                                                                                                                                 |
| Xilinx     | <code>syn_loc {pin_number}</code><br>or<br><code>xc_loc {pin_number}</code><br>See <a href="#">Specifying RLOCs, on page 819</a> for details about relative placement. |

---

## Specifying Locations for Actel Bus Ports

You can specify pin locations for Actel bus ports. To assign pin numbers to a bus port, or to a single- or multiple-bit slice of a bus port, do the following:

1. Open the constraint file and add these attributes to the design.
2. Specify the `syn_noarrayports` attribute globally to bit blast all bus ports in the design.

```
define_global_attribute syn_noarrayports {1};
```

3. Use the `alspin` attribute to specify pin locations for individual bus bits. This example shows locations specified for individual bits of bus ADDRESS0.

```
define_attribute {ADDRESS0[4]} alspin {26}
define_attribute {ADDRESS0[3]} alspin {30}
define_attribute {ADDRESS0[2]} alspin {33}
define_attribute {ADDRESS0[1]} alspin {38}
define_attribute {ADDRESS0[0]} alspin {40}
```

The software forward-annotates these pin locations to the place-and-route software.

## Specifying Macro and Register Placement

You can use attributes to specify macro and register placement in Actel and QuickLogic designs. The information here supplements the pin placement information described in [Specifying Pin Locations, on page 832](#) and bus pin placement information described in [Specifying Locations for Actel Bus Ports, on page 833](#).

| For...                                                               | Use...                                                                                 |
|----------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Relative placement of Actel macros and IP blocks                     | <a href="#">alsloc Attribute</a><br>define_attribute {u1} alsloc {R15C6}               |
| Placement of Lattice ORCA input or output registers next to I/O pads | <a href="#">orca_padtype Attribute</a><br>define_attribute { load } orca_padtype "IBT" |

## Passing Technology Properties

The following table summarizes the attributes used to pass technology-specific information for certain vendors. For details about the attributes in the table, see the *Reference Manual*.

| Vendor       | Attribute for passing properties                                                                                                                                                                                      |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lattice ORCA | <a href="#">orca_props Attribute</a><br>define_attribute {p:data_in} orca_props {LEVELMODE=LVDS}                                                                                                                      |
| Xilinx       | Specify the Xilinx properties directly in the source code. The software passes them to the place-and-route tool. For example:<br>attribute INIT of RAM1 : label is "0000";<br>or<br>/* synthesis INIT_xx = "value" */ |

## Specifying Padtype and Port Information

For many vendors, you can use attributes to specify technology-specific port information or padtype.

| Information | Vendor       | Attribute                                                                                |
|-------------|--------------|------------------------------------------------------------------------------------------|
| Padtype     | Lattice ORCA | <a href="#">orca_padtype Attribute</a><br>define_attribute {AIN[3]} orca_padtype {IBT}   |
|             | QuickLogic   | <a href="#">ql_padtype Attribute</a><br>define_attribute {clk} ql_padtype {CLOCK}        |
|             | Xilinx       | <a href="#">xc_padtype Attribute</a><br>define_attribute {a[3:0]} xc_padtype {IBUF_GTLP} |

---

| <b>Information</b> | <b>Vendor</b> | <b>Attribute</b>                                                                                      |
|--------------------|---------------|-------------------------------------------------------------------------------------------------------|
| Ports              | Altera        | <a href="#">altera_io_opendrain Attribute</a><br>define_attribute {alucout} altera_io_opendrain {1}   |
|                    | Altera        | <a href="#">altera_io_powerup Attribute</a><br>define_attribute {seg [31:0]} altera_io_powerup {high} |
|                    | Xilinx        | <a href="#">xc_isgsr Directive</a><br>define_attribute {bbgsr.gsrin} xc_isgsr {1}                     |
|                    | Xilinx        | <a href="#">xc_pullup/xc_pulldown Attribute</a><br>define_attribute { port_name } xc_pulldown { 1 }   |

---

## Generating Vendor-Specific Output

The following topics describe generating vendor-specific output in the synthesis tools.

- [Targeting Output to Your Vendor](#), on page 836
- [Customizing Netlist Formats](#), on page 837

### 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                 |
|---------------------------------------|----------------------------|--------------------------|
| Actel                                 | EDIF (.edn)<br>*_sdc.sdc   | Designer Series          |
| Altera Flex and Acex                  | EDIF (.edf)<br>AHDL (.tdf) | MAX+PLUSII or Quartus II |
| Altera Apex, Stratix, Max-II, Cyclone | Verilog (.vqm)             | Quartus II               |
| Altera Max                            | EDIF (.edf)<br>AHDL (.tdf) | MAX+PLUSII               |
| Lattice                               | EDIF (.edf)                | ispExpert                |
| Lattice Mach                          | EDIF (.edf) or .src        | ispExpert                |
| Lattice Orca                          | EDIF (.edn)                | ispLEVER                 |
| QuickLogic                            | EDIF (.qdf or .edf)        | SpDE                     |



| Vendor                                  | Output Netlist            | P&R Tool                                        |
|-----------------------------------------|---------------------------|-------------------------------------------------|
| Xilinx CoolRunner                       | EDIF (.edf) or .src       | Web Fitter for EDIF files, Minc for *.src files |
| Xilinx Spartan and XC4000, XC4500, etc. | EDIF (.edf) or XNF (.xnf) | Design Manager or ISE Project Navigator         |
| Xilinx Virtex and Spartan-3             | EDIF (.edf)               | Design Manager or ISE Project Navigator         |

- To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

See [Specifying Result Options](#), on page 296 for details about setting the option. For more information about constraint file output formats and how constraints get forward-annotated, see [Generating Constraint Files for Forward Annotation](#), on page 105.

## Customizing Netlist Formats

The following table lists some attributes for customizing your Actel, Altera, and Xilinx output netlists:

| For...                   | Use...                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Netlist formatting       | <a href="#">syn_netlist_hierarchy Attribute</a> (Altera, Xilinx, Actel)<br>define_global_attribute syn_netlist_hierarchy {0}                                                                                                                                                                                                                                        |
| EDIF formatting (Xilinx) | <a href="#">syn_edif_bit_format Attribute</a> (Xilinx)<br>define_global_attribute syn_edif_bit_format {%n<%i>}<br><br><a href="#">syn_edif_name_length Attribute</a> (Xilinx)<br>define_global_attribute syn_edif_name_length {restricted }<br><br><a href="#">syn_edif_scalar_format Attribute</a> (Xilinx)<br>define_global_attribute syn_edif_scalar_format {%u} |
| Bus specification        | <a href="#">syn_noarrayports Attribute</a> (Altera, Xilinx, Actel)<br>define_global_attribute syn_noarrayports {1}                                                                                                                                                                                                                                                  |

# Invoking Third-Party Vendor Tools

You can invoke third-party tools from within the Synopsys FPGA synthesis products, and configure the locations and common arguments for the tools. This capability lets you modify source files or libraries or debug problems from within the third-party tool, without leaving the synthesis environment.

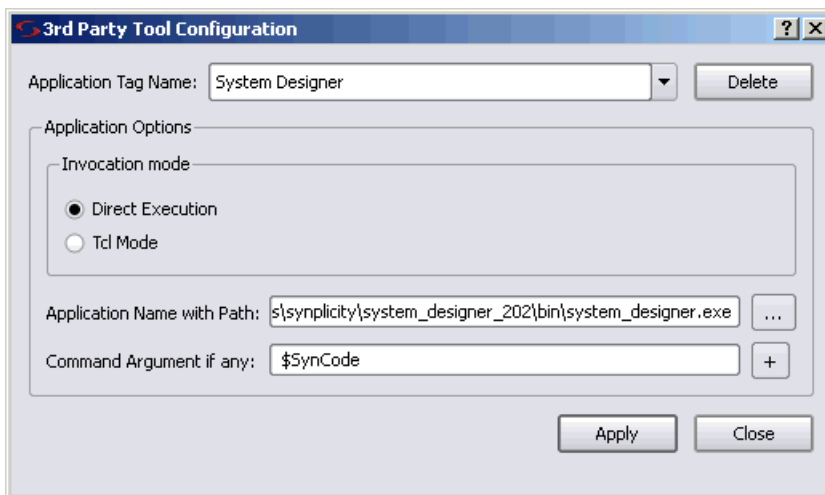
You can invoke preconfigured tools, or to add your own. The process consists of two steps:

- [Configuring Tool Tags](#), on page 838
- [Invoking a Third-Party Tool](#), on page 839

## Configuring Tool Tags

A tool tag is a configuration definition for a tool you want to invoke from the synthesis interface. You define a tool tag to set up the third-party tool you want to use. The synthesis software has some popular applications already configured for you to use when the synthesis software starts up: System Designer, Altera Megawizard, Xilinx EDK, and Xilinx Coregen. The following procedure shows you how to define your own tool tags, or add command arguments. Use this to specify other tools, other versions of a tool, or to run a tool with different arguments.

1. Select Options->Configure 3rd Party Tool Options from the Project view.



2. Define the application tag information for the tool you want to invoke.
  - Specify the application you want to invoke in Application Tag Name.
  - Specify how you want to invoke the application tool. If you want to run the tool directly from the UI, select Direct Execution. If your application is a Tcl procedure, select TCL Mode.
  - Specify the location of the application executable or Tcl procedure name in Application Name with Path or Tcl Procedure Name.
  - Specify any command arguments you want in the Command Argument if any field. You can use this to define a new tool tag or to add arguments to a tool tag that is already defined.

For a list of predefined command arguments, click the + button and select them from the list. Otherwise, type the command arguments. For the System Designer and other internal Synopsys tools, you must select \$SynCode from the Command Argument if any field.

- Click Apply.
- Click Close.

The tool saves these settings in the FPGA synthesis tool .ini file and retrieves them for subsequent invocations. For information about invoking a third-party tool, see [Invoking a Third-Party Tool, on page 839](#), next.

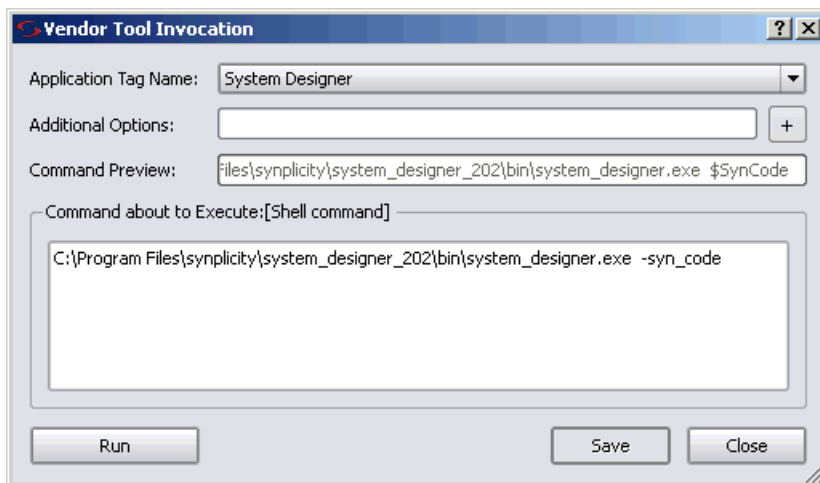
## Invoking a Third-Party Tool

You can define tool tags globally and then use these tool tags to run the third-party tool from the Project view for the specified tool tag only. Some common tool tags are preconfigured and are read when the application starts up. You can add or modify existing tool tags or define your own Tcl procedures to invoke within the FPGA synthesis tools.

1. Define a tool tag for your application, as described in [Configuring Tool Tags, on page 838](#).
2. Right-click in the Project view on a file or folder which is configured to run the vendor tool, and select Launch Tools->Run Vendor Tool from the popup menu.

This dialog box automatically displays tool tag information associated with the file or folder. If no tool tag information is specified, look for the parent hierarchy and edit or change it, if possible.

3. To associate a file or folder with a particular third-party tool, do the following:
  - Select the file or folder in the Project view. If you select a folder, the third-party tool is associated with all the files in the folder. If you associate a tool with a file, this setting overrides the folder setting.
  - Right-click a file or folder and select Launch Tools->Run Vendor Tool from the popup menu.



- In the Vendor Tool Invocation dialog box, select the application in Application Tag Name.
  - Include any additional options you want to use with this file when you invoke the vendor tool. You can set command arguments now, if you did not configure them earlier.
  - Verify the command string in the dialog box.
  - Click Save, and Close. The third-party tool is associated with the file or folder and appears in the Launch Tools menu.
4. To invoke an associated third-party tool for a file or folder, do the following:
    - Right-click the file or folder in the Project view.
    - Select Launch Tools-><Third-Party Tool> from the popup menu. The synthesis tool automatically runs the tool or Tcl procedure as specified.

5. To invoke the tool at the same time that you associate a third-party tool with a file or folder, or to add additional arguments on the fly, do the following:
  - Right-click a file or folder and select Launch Tools->Run Vendor Tool from the popup menu.
  - In the Vendor Tool Invocation dialog box, select the application in Application Tag Name.
  - Include any additional options you want to use with this file when you invoke the vendor tool. You can set command arguments now, if you did not configure them earlier.
  - Verify the command string in the dialog box.
  - Click Save. The tool and arguments you specified is associated with the file or folder and appears in the Launch Tools menu.

If you defined a new tool tag, the 3rd Party Tool Configuration dialog box appears. After saving the settings here, go back to the Vendor Tool Invocation dialog box. You are prompted to save this information to the project file before invoking the third-party tool.

- Click the Run button in the Vendor Tool Invocation dialog box. The synthesis tool launches the third-party tool or runs the Tcl procedure with the arguments you specified.

These settings are saved in the FPGA synthesis tool .ini file, from where it can be retrieved for subsequent invocations.



**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 20

# Running Post-Synthesis Operations

---

The following describe post-synthesis operations:

- [VIF Formal Verification Flow](#), on page 844
- [Running Place-and-Route after Synthesis](#), on page 849
- [Simulating with the VCS Tool](#), on page 851
- [Resynthesizing with QuickLogic Information](#), on page 856
- [Quartus II Incremental Compilation](#), on page 857
- [Working with Xilinx Incremental Flows](#), on page 862
- [Working with the Identify RTL Debugger](#), on page 868

## VIF Formal Verification Flow

During synthesis, the Synplify Pro tool performs several sequential optimizations and design transformations to improve delay and area. These transformations make it difficult for a formal verification tool to match registers in the result netlist with the corresponding registers in the source HDL (a prerequisite for verifying equivalence). To solve this, the Synplify Pro software provides a Tcl file interface that lets you integrate with verification tools. This proprietary format is called the Verification Interface Format or VIF. This feature is currently available for only Xilinx and Altera technologies.

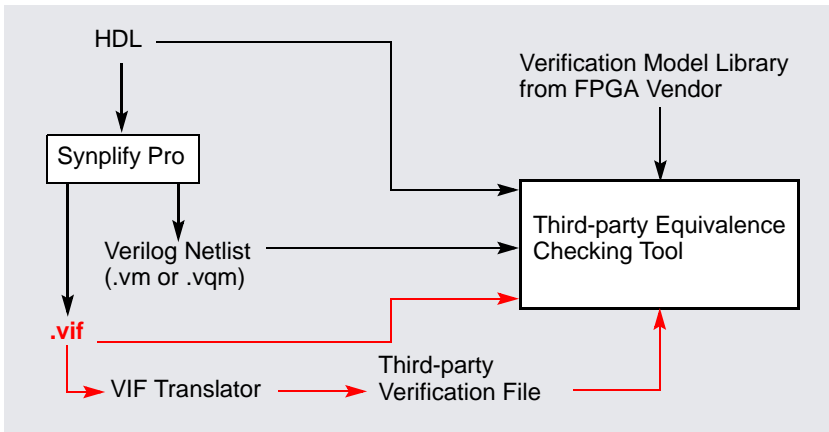
This section describes the following:

- [Overview of the VIF Flow](#), on page 844
- [Generating a VIF File](#), on page 845
- [Generating a VIF File](#), on page 845
- [Using a Tcl Script for VIF Conversion](#), on page 847
- [Handling Equivalency Check Failures](#), on page 848

### Overview of the VIF Flow

The Synplify Pro VIF flow is based on a Tcl file generated during synthesis. This file has a .vif extension. It contains a vendor-independent list of the design transformations performed during synthesis so that the verification tool can do equivalence checking and match up the post-synthesis registers with the original golden netlist. The following diagram summarizes the two ways in which you can use the .vif file as input.





## Generating a VIF File

1. In the Synplify Pro interface, select Project->Implementation Options and set the following on the Device tab:

| Option                           | Value                               |                          |
|----------------------------------|-------------------------------------|--------------------------|
| Annotated Properties for Analyst | <input checked="" type="checkbox"/> |                          |
| Fanout Guide                     | 10000                               |                          |
| Disable I/O Insertion            | <input type="checkbox"/>            |                          |
| Pipelining                       | <input checked="" type="checkbox"/> |                          |
| Update Compile Point Timing Data | <input type="checkbox"/>            |                          |
| Verification Mode                | <input type="checkbox"/>            | Enable Verification Mode |
| Modular Design                   | <input type="checkbox"/>            |                          |
| Retiming                         | <input type="checkbox"/>            | Disable Retiming         |
| Disable Sequential Optimizations | <input type="checkbox"/>            |                          |
| Fix gated clocks                 | 3                                   |                          |
| Fix generated clocks             | 3                                   |                          |

Figure 20-1: Device options for generating VIF output

- Set Technology to an Altera or Xilinx family that supports the VIF flow.
- Disable Retiming. This is an optional, but recommended step. Register retiming optimizations are hard to verify. The disadvantage is that you may lose performance when you disable retiming.

- Enable the Verification Mode option. This is another optional step that disables various sequential optimizations that can not be easily verified; the inference of resettable SRLs for example. The trade-off when you enable the Verification Mode option is that you may sacrifice performance or area, because the optimizations are not performed.

The reason for disabling sequential optimizations is to make it easy for the verification tool to sync up registers. Sequential optimizations are hard to verify because registers are moved or optimized away. For a list of VIF optimization commands, see step 4, below.

2. Go to the Implementation Results tab and enable Write Verification Interface Format (VIF) File.

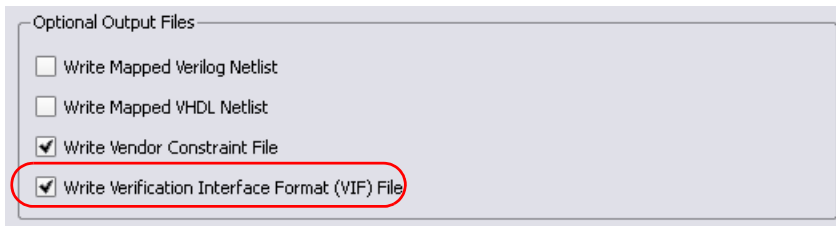


Figure 20-2: Implementation Results Options for Generating VIF Output

For Altera designs, make sure to use .vqm as the output format, not .vm.

3. Synthesize the design as usual.

The Synplify Pro software generates the .vif file and stores it in the project/verif directory.

4. Check the .vif file to see how the optimizations were handled.

The following table lists the VIF commands used to map some synthesis optimizations. For details of the command syntax, refer to [Tcl VIF Commands](#), on page 1275 in the *Reference Manual*.

| Optimization         | VIF Command       |
|----------------------|-------------------|
| FSM register mapping | vif_set_fsmreg    |
| FSM state encoding   | vif_set_state_map |
| Register merging     | vif_set_merge     |
| Register replication | vif_set_equiv     |

| Optimization                      | VIF Command                           |
|-----------------------------------|---------------------------------------|
| Pruning of duplicate registers    | vif_set_constant, vif_set_transparent |
| Black boxes for undefined modules | vif_set_map_point                     |
| Port direction changes            | vif_set_port_dir                      |

- Use the .vif file as input to any formal verification tool that supports a Tcl interface. Do one of the following:
  - If you are using the Cadence Conformal tool, run the translation script vif2conformal.tcl which is in the *install\_dir/lib* directory (see [Using a Tcl Script for VIF Conversion, on page 847](#) for details). This translates the .vif file commands to commands for the Conformal tool.
  - If your verification tool does not directly support VIF commands, create a script that translates the .vif file commands to native Tcl commands.
  - If the verification tool supports the VIF commands in its Tcl framework, use the file directly.
- In the verification tool, use the information from the .vif file along with the synthesis output when you check logic equivalence against the golden netlist.

## Using a Tcl Script for VIF Conversion

The Synplify Pro software includes Tcl scripts for use with the Cadence Conformal tool. You can convert .vif files manually or automatically.

### Converting VIF Manually

The following procedure describes how to convert .vif files manually.

- Source the vif2conformal.tcl file by typing one of the following commands in the Synplify Pro Tcl window:

```
source synplify_pro_install_dir/lib/vif2conformal.tcl
```

or

```
source $LIB/vif2conformal.tcl
```

2. In the Tcl window, navigate to the verification folder containing the *design.vif* file, and type the following command:

```
vif2conformal design.vif
```

The *vif2conformal.tcl* script runs on the *design.vif* file and translates the information into Conformal side files (\*.vtc, \*.vsc, \*.vmc, and so on). You can now run Conformal using these files.

## Automating VIF Conversion with Synhooks

You can create a script using the *synhooks.tcl* file (see [Automating Flows with \*synhooks.tcl\*, on page 884](#)) to automate the generation of verification files. An example of this file, *synhooks\_for\_vif2conformal.tcl*, is located in the *install\_dir/examples* directory.

The *synhooks\_for\_vif2conformal.tcl* Tcl script sets your environment to automatically convert the Synplify Pro generated *.vif* file to Conformal-specific side files at the end of each synthesis run. Use either of the following methods to convert your files:

- Set the environment variable `SYN_TCL_HOOKS` to point to the *synhooks\_for\_vif2conformal.tcl* file. For example:

```
SYN_TCL_HOOKS=install_dir/examples/synhooks_for_vif2conformal.tcl
```

- Source the *synhooks\_for\_vif2conformal.tcl* file in the Synplify Pro Tcl window to set up automatic conversion. For example:

```
% source install_dir/examples/synhooks_for_vif2conformal.tcl
```

For this method, you must source the *synhooks\_for\_vif2conformal.tcl* file every time you start a new project; otherwise the tool is reopened. The automatic conversion setup is lost once you close a Synplify Pro project or restart the tool.

## Handling Equivalency Check Failures

If your design fails the equivalency check, try the following tips and techniques to debug the results.

- Check the log file report and fix the errors reported.
- Check the optimization mapping in the *vif* file. See step 4 of [Overview of the VIF Flow, on page 844](#) for a list of commands.

## Running Place-and-Route after Synthesis

For Altera, Actel, and Xilinx technologies, you can create a place-and-route implementation to run the tool automatically after synthesis. You can run place-and-route from within the tool or in batch mode. This feature is only available in the Synplify Pro and Synplify Premier tools.

You can run the place-and-route tool for your target technology automatically after synthesis.

1. Check the Release Notes and make sure that you are using the correct version of the P&R tool.
2. To manually launch the P&R tool in Altera and Xilinx technologies, do the following:
  - For Altera designs, select the run option you want from the Options-><Altera\_tool> menu. The tool launches and displays the P&R tool interface. You can configure your settings and run P&R. See [Guidelines for Running Xilinx P&R, on page 850](#) for some tips.
  - For Xilinx designs, select the run option you want from the Options->Xilinx menu. The tool launches and displays displays the place-and-route tool user interface, places the synthesis-generated netlist in a Xilinx project, and names the project. Configure your Xilinx project settings in the place-and-route tool. Run the Xilinx place-and-route tool.
3. To automatically run the P&R tool after synthesis completes, do the following:
  - If necessary, set up a place-and-route implementation as described in [Creating a Place and Route Implementation, on page 332](#). You need a P&R implementation for physical synthesis flows, or if you want to use post-P&R data for backannotation.
  - Click on the Add P&R Implementation button. In the dialog box, select the P&R implementation you want to run and enable Run Place & Route following synthesis.
  - Synthesize the design.

The tool automatically runs P&R after synthesis.

## Guidelines for Running Xilinx P&R


The following table lists some tips for running various phases of Xilinx place-and-route:

|          |                                                                                                                                                                                                                                                                                                                                      |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NGDBuild | Use the user constraint file ( <code>synplify.ucf</code> ) generated after synthesis. From the command line, read the <code>.ucf</code> file into Xilinx place-and-route NGDBuild with the <code>-uc</code> command.                                                                                                                 |
| MAP      | Do not map to 5-input functions. Do not use the <code>-k</code> command line option. For information about using <code>map</code> with compile points, see <a href="#">Logical Compile-Point Synthesis, on page 560</a> .                                                                                                            |
| PAR      | Do not use the default effort level of <code>std</code> . Instead, set it to high using the <code>-ol high</code> command line option. For information about using <code>par</code> with the Synplify Premier and Synplify Premier compile-point synthesis flows, see <a href="#">Logical Compile-Point Synthesis, on page 560</a> . |

## 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.

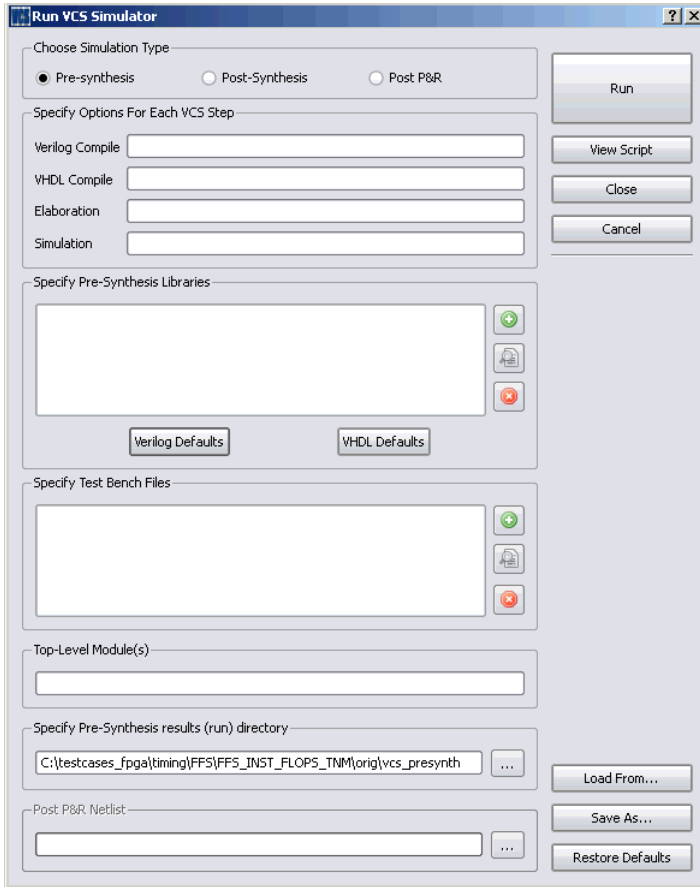
- Install the VCS software and set up the \$VCS\_HOME environment variable to define the location of the software.
- Set up the place-and-route tool.
- In the synthesis software, either select Run->Configure and Launch VCS Simulator, or click the  icon.

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, on page 212](#) of the *Reference Manual*.

2. In the dialog box, 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       |

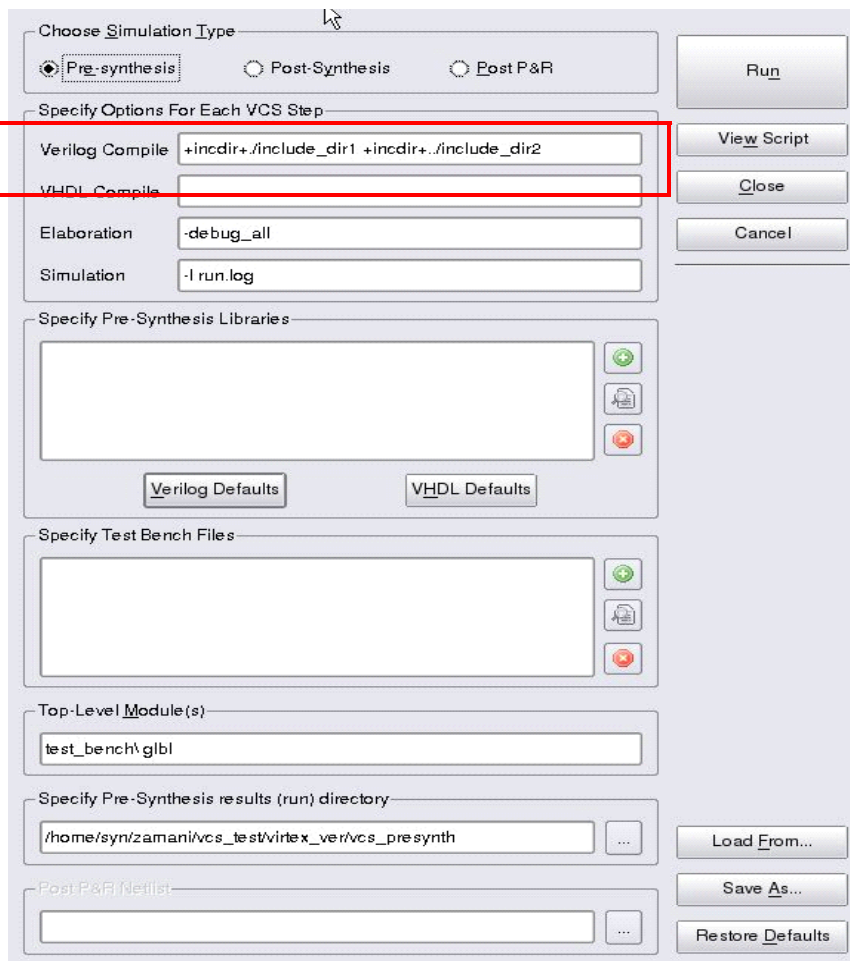


- Set options for VCS commands in the Specify options for each VCS Step section. These options are written out as VCS commands in the script. See the VCS documentation for details of command options.

| 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            |



3. If your project has Verilog files with ``include` statements, you must use the `+incdir+ <file name>` argument when you specify the `vlogan` command. You enter the `+incdir+` in the Verilog Compile field in the VCS popup window, as shown below:



**Example Verilog File:**

```

`include "component.v"

module Top (input a, output x);

...

endmodule

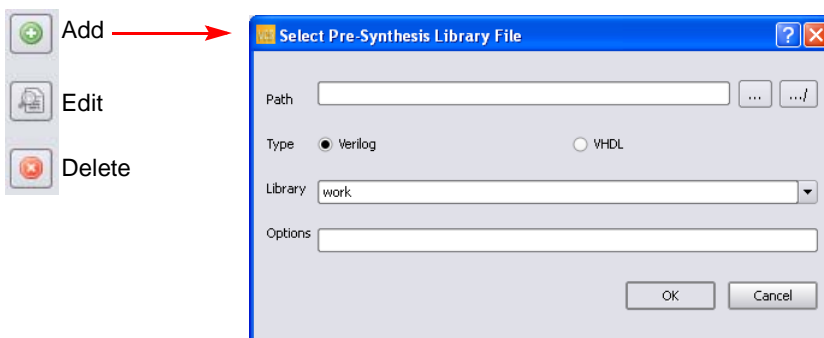
```

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, specify them in the Specify Test Bench Files section. Use the buttons on the side to add, delete, or edit the files.


5. Specify the top-level module and run directory.

- Specify the top-level module or modules for the simulation in Top Level Module(s).
- If necessary, edit the default run directory listed in the Specify Run Directory field at the bottom of the dialog box. The default location is in the implementation 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.

## Resynthesizing with QuickLogic Information

For QuickLogic designs, you can use pad placement information from the place-and-route run when you resynthesize your design. You might want to use this methodology to redesign a part so that it works in an existing system, without having to change FPGA connections.

1. After synthesis, place and route your design with SpDE.
2. Check the following in the `.scp` command file generated by SpDE:
  - Make sure the object names and the case in the `.scp` file match the names and case in the source file.
  - Use the `portprop` command to specify pad placement and pad type.
  - Specify fixed placement for I/O pads with the `instprop` command.

For the syntax of these commands, see the *Reference Manual*.

3. Include the `.scp` command file in your project by doing one of the following:
  - Add the `include` directive to your project file, and specify the `.scp` file with the pad placement information.
  - Add the `include` directive to a Tcl script file, and specify the `.scp` file with the pad placement information. Read the Tcl script into your project.

For more information about the `include` directive, see the *Reference Manual*.

4. Resynthesize your design.

When you modify and resynthesize the design, the software keeps the pin locations specified in the included `.scp` file.

# Quartus II Incremental Compilation

The Altera Quartus II Incremental Compilation feature preserves design implementation data so as to make incremental place-and-route updates. This section describes how to use the Quartus II Incremental Compilation flow in the Synplify Pro or Synplify Premier synthesis environment. For complete details on this feature, see the Altera product documentation.

Use the Quartus II Incremental Compilation flow if you make design changes, such as HDL changes to a small number of modules, moving a pin, changing an attribute, or changing a timing constraint. You can also apply changes to the critical timing path. This feature compares partitions (individual design modules based on RTL hierarchies) in the previous and current implementations. It preserves partitions that are the same in the two implementations, and re-implements any partitions that have been resynthesized in the current implementation. It places and routes the individual partitions.

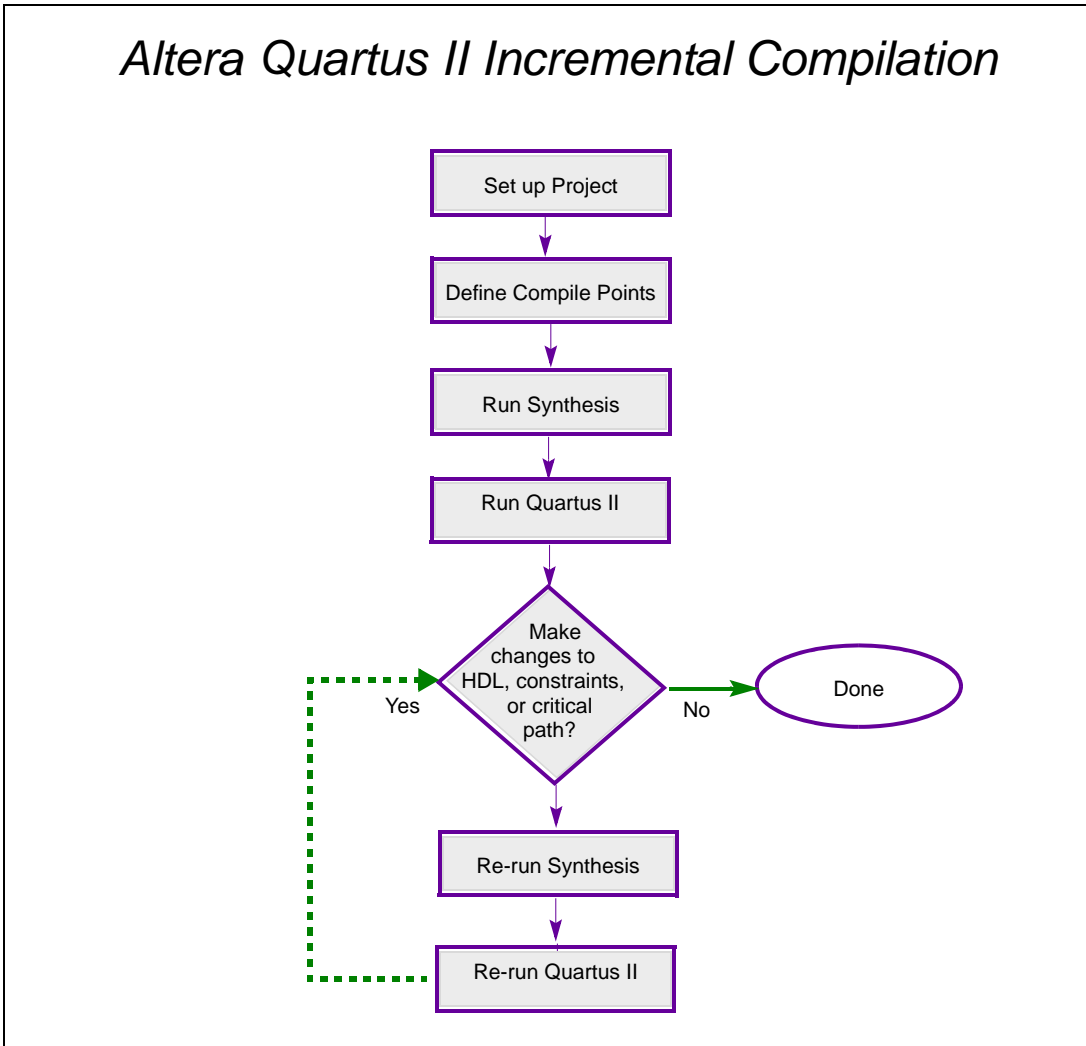
## Quartus II Incremental Compilation Flow

In the Altera Quartus II Incremental Compilation flow, the synthesis software automatically generates a Tcl script which creates design partition assignments. The Quartus II software use this Tcl script to determine if partitions should be preserved from previous place and route results when you recompile the design. If a partition is removed, the Tcl script detects this and an updated VQM netlist is forward annotated. This feature can save a significant amount of time in that placement and routing for the entire design does not need to be rerun.

See the following topics for details on this flow:

- [Altera Quartus II Incremental Compilation Diagram](#), below
- [Synthesizing, Placing and Routing in the Quartus Incremental Flow](#), on page 859

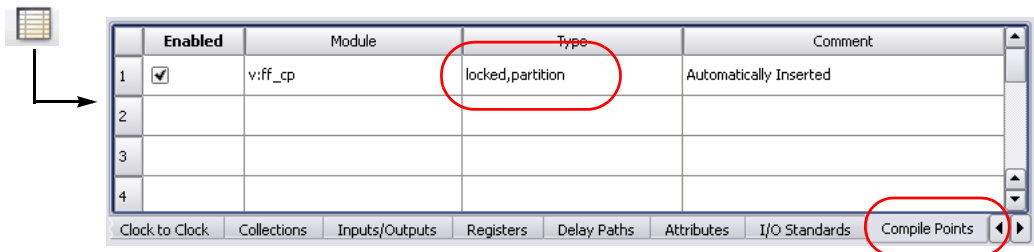
### Altera Quartus II Incremental Compilation Diagram



## Synthesizing, Placing and Routing in the Quartus Incremental Flow

1. Set up the project file (.prj). See [Setting Up Project Files, on page 270](#) if you need details.
  - Select the target device. Quartus II Incremental Compilation supports most Stratix and subsequent device families.
  - Add the appropriate HDL source files to the project.
  - Set the implementation options.
  - Compile the design.
2. Define compile points in the top-level constraint file using the SCOPE UI. For details, see [Using Compile-point Synthesis, on page 573](#).
  - Click the Compile Points tab, and set compile points.

The compile points are design module partitions, which you define based on the RTL-level hierarchies. On the Compile Points tab for the required module select {locked,partition} from the Type field. See [About Compile Points, on page 562](#) for details about compile points. Partitioning the design provides a mechanism where only modified sections of the design need be updated, thus saving time. The following example shows the compile point for v:ff\_cp.



- Create a compile point constraint file for each compile point in the SCOPE UI. For the details, see [Set Constraints, on page 577](#).
3. Click Run to synthesize the design and check the compile point summary in the log file.

Check the log file for messages like those displayed below.

## Mapper Messages

```
@N:MF104 : ff_cp_v\(1\) | Found Compile Point of type locked,partition on View view:work.ff_cp(verilog)

@N:MF105 : | Performing bottom up mapping of Compile point view:work.ff_cp(verilog)
@N:MF106 : ff_cp_v\(1\) | Mapping Compile point view:work.ff_cp(verilog) because
 No database from previous run found.
```

## First Run Log Summary

```
Summary of Compile Points

Name Status Reason

ff_cp Mapped No database
test Mapped No database
=====
```

The synthesis tool generates a VQM file where the compile points are defined with "syn\_hier=locked,physical" attributes. For subsequent synthesis runs, the tool also automatically generates a Tcl script which creates design partition assignments.

4. Run the Quartus II place-and-route tool.

The Quartus II software uses the compile point VQM files, as well as the Tcl script file generated from the synthesis tool to determine if partitions should be preserved from previous place and route results when you recompile the design.

5. Go back to the synthesis tool and re-synthesize the modified design.

The synthesis tool only resynthesizes and optimizes the updated modules. In the VQM file generated for this synthesis run, the tool does not change the timestamp of a compile point if it has not changed since the previous run; it preserves the old timestamp. Updated modules get a new timestamp.

For an incremental run, the software only resynthesizes compile points whose logic, implementation options, or timing constraints have changed.

The following figure illustrates incremental synthesis by comparing compile point summaries. After the first run, a logic change in the `ff_cp`



module. The figure shows that incremental synthesis resynthesizes ff\_cp (logic change), but does not resynthesize test because the logic did not change.

First Run Log Summary

```

Summary of Compile Points

Name Status Reason

ff_cp Mapped No database
test Mapped No database
=====

```

Incremental Run Log Summary

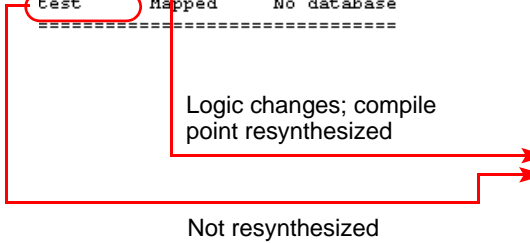
```

Summary of Compile Points

Name Status Reason

ff_cp Remapped Design changed
test Unchanged -
=====

```



6. Rerun the Quartus II tool and place and route the design.

The Altera Quartus II software performs a checksum on the compile point file with the corresponding file from the previous run, and incrementally places and routes only those partitions from updated files. It leaves the other partitions untouched.

## Working with Xilinx Incremental Flows

This section describes how to use the Xilinx SmartCompile feature in the synthesis environment. SmartCompile is a feature introduced in Xilinx ISE version 9.1, that preserves design implementation data so as to make incremental place and route updates. For complete details on SmartCompile, see the Xilinx product documentation. You can use the following methods:

- [Incremental Flow for Xilinx Designs](#), on page 862. Use this flow for minor changes to a small number of modules.
- [SmartGuide Global Placement Flow](#), on page 863. Use this flow for minor changes to a small number of modules.
- [Partition Flow](#), on page 863. Use this flow for changes to the critical timing path, or major constraint changes, such as adding area groups.

### Incremental Flow for Xilinx Designs

The incremental flow is only available in the Synplify Premier tool. In this flow, you use the results of the previous P&R run, and only run P&R incrementally. This flow is similar to the SmartGuide flow ([SmartGuide Global Placement Flow, on page 863](#)).

Do the following:

1. Start with a design that has been through synthesis and global placement.
2. In the Synplify Premier UI, enable the Physical Synthesis option.  
  
This flow is a physical synthesis flow, so you must have this option turned on.
3. Update or make minor changes to the design.
4. Click Implementation Options, and enable Incremental Flow on the Device tab.
5. Set up place and route to run automatically after synthesis, and resynthesize the design.

Alternatively, you can synthesize the design and run P&R separately. The Xilinx tool uses the previous results and does an incremental placement run, and then routes the design.

## SmartGuide Global Placement Flow

The SmartGuide flow is an incremental flow you can use in the Xilinx tool. You can use this with both logic synthesis designs (Synplify Pro) as well as physical synthesis designs. Use the SmartGuide option when you need to make minor changes to a design, such as HDL changes to a small number of modules (logic changes of approximately 10% or less), moving a pin, changing an attribute, or changing a timing constraint. The SmartGuide flow is similar to the incremental flow described in [Incremental Flow for Xilinx Designs, on page 862](#), but that flow is only available in the Synplify Premier tool.

The following procedure shows you how to use this flow with the synthesis tools:

1. Do an initial synthesis run with placement and routing.
2. Set the SmartGuide option in the Xilinx software.
3. Rerun synthesis and P&R.

SmartGuide is enabled, so the tool does not redo global placement, but uses the settings from the previous global placement run and only runs incremental global placement. This flow compares the previous implementation to the current design and preserves any common components. Modified components are incrementally re-implemented. For complete details on SmartGuide, see the Xilinx documentation.

## Partition Flow

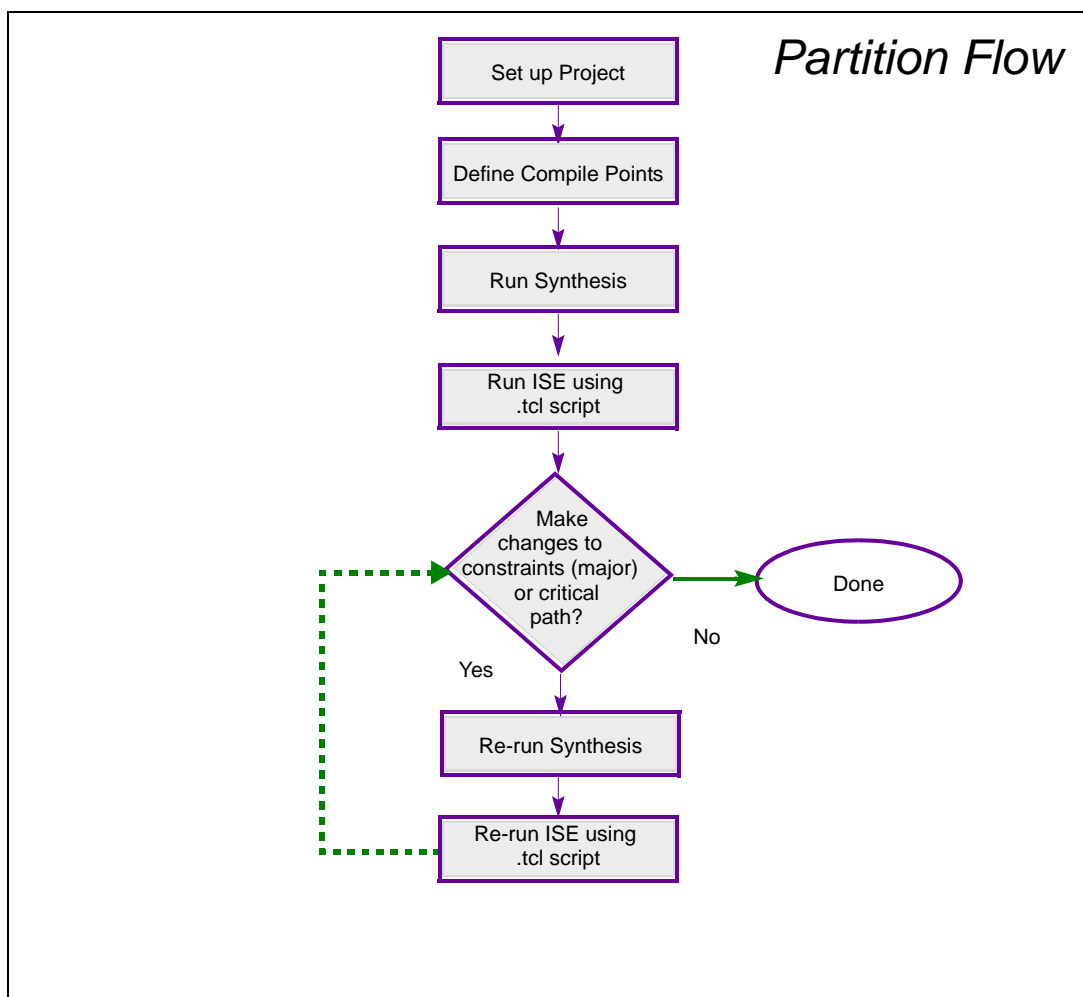
Use the Partition flow if you make changes to the critical timing path, or if you make major constraint changes, such as adding area groups. This flow compares partitions (individual design modules based on RTL hierarchies) in the previous and current implementations. It preserves partitions that are the same in the two implementations, and re-implements any partitions that have been resynthesized in the current implementation. It places and routes the individual partitions.

The Xilinx partition feature uses a block-based flow to determine when incremental place-and-route updates are needed. Use this flow for major design changes; for minor changes, use the Smart Guide Incremental Flow explained above.

The design partitions, or modules, are defined before you run synthesis. Using the block-based flow, if design changes are required after running synthesis and place and route, only the affected modules are resynthesized. Subsequently, only resynthesized modules are re-implemented during place and route. Partitions that have not changed are preserved from the previous implementation. This feature saves a significant amount of time in that place and route for the entire design does not need to be rerun. See the following for topics details on this flow:

- [Partition Flow Diagram](#), next
- [Running Synthesis, and Place-and-Route in the Partition Flow](#), on page 865

## Partition Flow Diagram



## Running Synthesis, and Place-and-Route in the Partition Flow

1. If necessary, set the `SYN_XILINX_PR_USE_XTCLSH` environment variable to 1.

This is the default in the latest releases of the synthesis tools, so you do not have to set it. The 1 setting ensures that the tool generates the

`run_ise.tcl` script file after synthesis. For subsequent synthesis runs, the tool recognizes the script file and does not overwrite it.

2. Set up the design.

- Set up the project file (`.prj`). See [Setting Up Project Files, on page 270](#) for details.
- Define compile points using the SCOPE UI. For details, see [Xilinx Compile-point Synthesis Flow, on page 583](#) and [Using Compile-point Synthesis, on page 573](#).

The compile points are design module partitions, which you define based on the RTL-level hierarchies. To define a compile point as a partition using the SCOPE UI, on the Compile Points tab for the required module select {locked, partition} from the Type field. Partitioning the design provides a mechanism where only modified sections of the design need be updated, thus saving time.

- Set up the rest of the design; set the desired switches and ensure that the appropriate source, constraint and library files have been added to the project.

3. Click Run to synthesize the design.

The synthesis tool generates an EDIF file where the compile points are defined with "PARTITION" properties. Each compile point also includes a time stamp for when the module was last synthesized. Later, the place and route tool uses the time stamp as the basis for comparison to determine which modules need to be incrementally updated.

4. Make sure you set the system path variables for the Xilinx place-and-route tool and run the P&R project `.tcl` script using the following command:

```
xtclsh.exe
```

5. Perform the following tasks when ISE placement and routing completes.

- Check generated reports.
- Make any necessary major design changes to the source or constraint files and place and route the design.

6. Go back to the synthesis tool and re-synthesize the modified design.

The synthesis tool only resynthesizes and optimizes the updated modules.

In the EDIF file generated for this synthesis run, the tool does not change the timestamp of a compile point if it has not changed since the previous run; it preserves the old timestamp. Updated modules get a new timestamp.

7. Rerun the ISE `.tcl` script and place and route the design.

The Xilinx tool compares the compile point timestamps to the corresponding timestamps from the previous run, and incrementally places and routes only those blocks with updated timestamps. It leave the other blocks untouched. The EDIF file also specifies whether an updated compile point needs to be re-placed and re-routed, or only re-routed. The default specifies incremental placement and routing.

## Working with the Identify RTL Debugger

The Synopsys Identify RTL Debugger is a dual-component system that is a valuable part of the HDL design flow process. The system consists of the Identify Instrumentor and Identify Debugger tools.

- The Identify Instrumentor allows you to select your design instrumentation at the HDL level and then create an on-chip hardware probe.
- The Identify Debugger interacts with the on-chip hardware probe and lets you do live debugging of the design.

The combination of these tools allows you to probe your HDL design in the target environment. The combined system allows you to debug your design faster, easier, and more efficiently.

You can run Identify independently, but the Synplify, Synplify Pro, and Synplify Premier synthesis tools have integrated the Identify Instrumentor into the synthesis user interface. This section describes how to take advantage of this integration and use the Identify Instrumentor:

- [Launching from the Synplify Pro or Synplify Premier Tool](#), on page 868
- [Launching from the Synplify Tool](#), on page 870
- [Handling Problems with Launching Identify](#), on page 872
- [Using the Identify Tool](#), on page 873

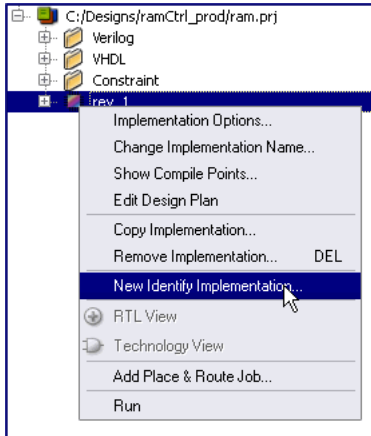
### Launching from the Synplify Pro or Synplify Premier Tool

The integration of the synthesis tools with Identify varies, depending on which tool you have. For the Synplify Pro and Synplify Premier tools, you must create an Identify implementation in order to run the Identify Instrumentor. If you already have an Identify implementation, open it and use the Identify tool as described in [Using the Identify Tool, on page 873](#).

Do the following to add an Identify implementation:

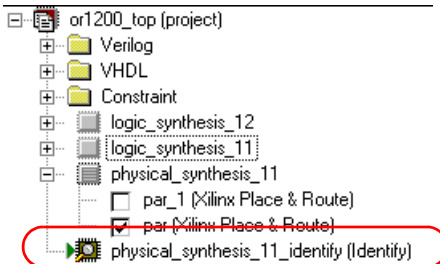
1. In the synthesis interface, open the design you want to debug.
2. Do one of the following tasks to add an Identify implementation:
  - With the project implementation selected, right-click and select New Identify Implementation from the pop-up menu.






- Select Project->New Identify Implementation.

An Implementation Options dialog box appears where you can set the options for your implementation. Note that the options apply only for logic synthesis and not for physical synthesis. An Identify implementation is created.



3. To run Identify Instrumentor, select the Launch Identify Instrumentor icon () in the toolbar or select Run->Identify Instrumentor.

The Identify interface opens. You can now use the Identify tool as described in [Using the Identify Tool, on page 873](#) For complete details, consult the Identify documentation.

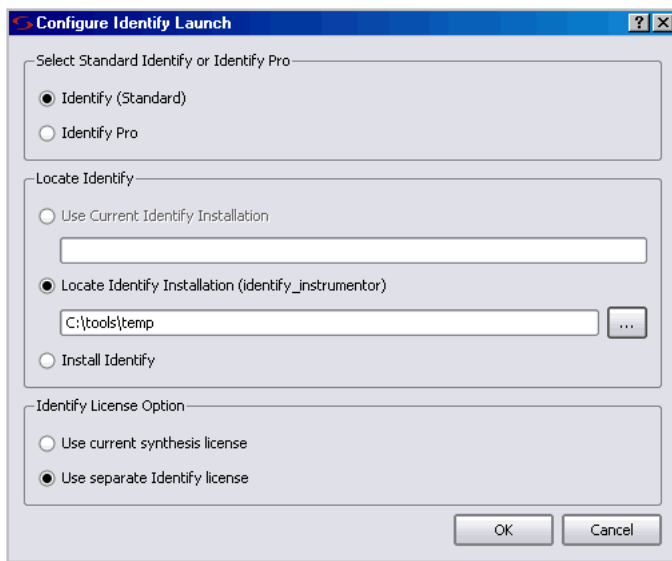
If you run into problems while launching the Identify Instrumentor, refer to [Handling Problems with Launching Identify, on page 872](#).

## Launching from the Synplify Tool

The Synplify tool does not support multiple implementations. The following procedures describe how to launch an Identify implementation from Synplify, and how to modify an existing implementation.

### Creating a New Identify Implementation in Synplify

1. Make sure that your PATH environment variable points to the Identify bin directory.
2. Create or open a project in Synplify.
3. Right-click on the implementation and select New Identify Implementation from the popup menu.
4. Set any implementation options and close the dialog box; dismiss the multiple implementation warning.
5. Select Options->Configure Identify Launch to display this dialog box:



6. Fill out the details.
  - Check the Identify installation. If the Use current Identify Installation field entry in the dialog box is not correct, click the Locate Identify Installation

button and enter the path to the Identify executable (*identify\_install/bin*).

- Select the appropriate license option and click OK to launch the Identify Instrumentor for the new implementation.
7. Instrument the design as required, save the instrumentation, and exit the Identify Instrumentor. See [Using the Identify Tool, on page 873](#) for an overview, or the Identify documentation for details.
  8. Synthesize the instrumented implementation (*rev\_n\_identify*) in Synplify (the schematic will show the added IICE circuitry).

After the design has been synthesized, place and route your design. Program the device, install the device in the target system, and complete the JTAG cable interface. You can now run the Identify Debugger on the instrumented design (*designName.bsp*) to verify correct operation.

## Modifying or Re-instrumenting an Existing Design


To modify or re-instrument an existing design:

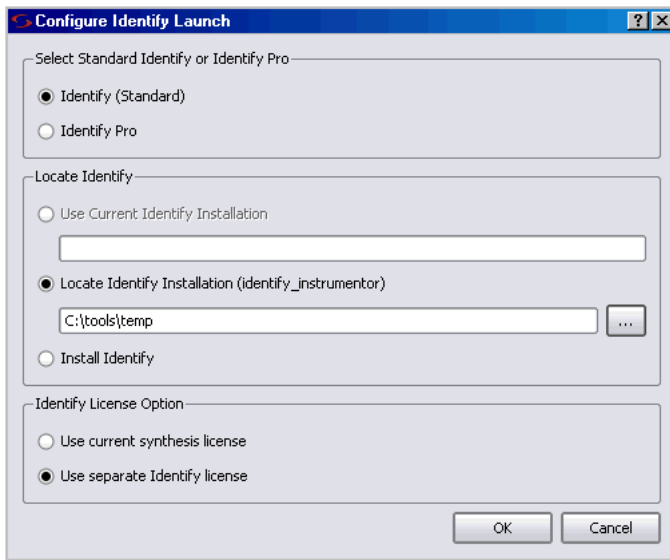
1. Load the project containing the Identify implementation into Synplify.
2. From the Synplify menu bar, select Run->Run TCL Script.
3. Navigate to the Synplify lib directory and run (open) the *relaunch\_identify.tcl* script to launch the Identify Instrumentor.
4. Re-instrument the design as required, save the instrumentation, and exit the Identify Instrumentor.
5. Re-synthesize the instrumented implementation (*rev\_n\_identify*) in Synplify.

After the design has been re-synthesized, place and route your design. Program the device and reinstall the device in the target system. You can now rerun the Identify Debugger on the instrumented design (*design-Name.bsp*) to verify correct operation.

## Handling Problems with Launching Identify

If you have not installed Identify correctly, you might run into problems when you try to launch it from the synthesis tools. The following describe some situations:

- If the Launch Identify Instrumentor icon () and the Run->Identify Instrumentor menu command are inaccessible, you are either on an unsupported platform or you are using a technology that does not support this feature.
- If you have the Identify software installed but the synthesis application cannot find it, select Options->Configure Identify Launch.



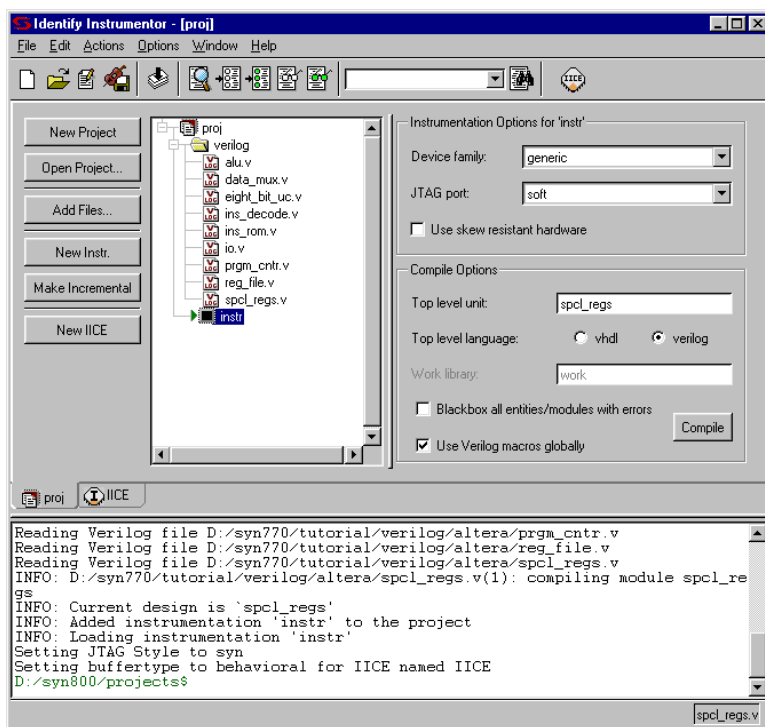
In the resulting dialog box, specify the correct location in the Locate Identify Installation field. You can use the Browse button to open the Select Identify Installation Directory dialog box and navigate to your current Identify installation directory.

- If you have not installed the Identify software, select Options->Configure Identify Launch, and select Install Identify. Follow the directions and install the software before going through the procedure.

## Using the Identify Tool

The following is an overview of the process to use the Identify Instrumentor. For detailed information about the tool, refer to the Identify RTL Debugger documentation.

1. The Identify Instrumentor software interface opens, with an Identify project automatically set up for the design to be instrumented and debugged (IICE tab). The following figure shows the main project window.



2. Do the following in the Identify Instrumentor interface:
  - Instrument the design. For details of using the Identify Instrumentor, refer to the Identify RTL Debugger documentation.
  - Save the instrumented design.

The Identify Instrumentor tool exports the instrumented design to the synthesis software. It creates an instrumentation subdirectory under your synthesis working directory called *designName\_instr*, which contains the following:

- A synthesis project file
  - An `instr_sources` subdirectory for the instrumented HDL files
  - Tcl scripts for loading the instrumented design
3. Return to the synthesis interface and view the instrumented design that contains the debugging logic.
    - In the synthesis interface, open the project file for the instrumented design, which is in the `instr_sources` subdirectory listed in the Implementations Results view for your original synthesis project.
    - Synthesize the design.
    - Open the RTL view to see the inserted debugging logic.
  4. Place and route the instrumented design after synthesis.
  5. Use the Identify Debugger tool to debug the instrumented design.

**Synopsys, Inc.**

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

## CHAPTER 21

# Process Optimization and Automation

---

This chapter covers topics that can help the advanced user improve productivity and inter operability with other tools. It includes the following:

- [Using Batch Mode](#), on page 876
- [Working with Tcl Scripts and Commands](#), on page 878
- [Automating Flows with synhooks.tcl](#), on page 884

## Using Batch Mode

Batch mode is a command-line mode in which you run scripts from the command line. You might want to set up multiple synthesis runs with a batch script. You can run in batch mode if you have a floating license, but not with a node-locked license.

Batch scripts are in Tcl format. For more information about Tcl syntax and commands, see [Working with Tcl Scripts and Commands](#), on page 878.

This section describes the following operations:

- [Running Batch Mode on a Project File](#), on page 876
- [Running Batch Mode with a Tcl Script](#), on page 877

### Running Batch Mode on a Project File

Use this procedure to run batch mode if you already have a project file set up. You can also run batch mode from a Tcl script, as described in [Running Batch Mode with a Tcl Script](#), on page 877.

1. Make sure you have a project file (.prj) set up with the implementation options. For more information about creating this Tcl file, see [Creating a Tcl Synthesis Script](#), on page 879.
2. From a command prompt, go to the directory where the project files are located, and type one of the following, depending on which product you are using:

```
synplify -batch project_file_name.prj
synplify_pro -batch project_file_name.prj
synplify_premier -batch project_file_name.prj
synplify_premier_dp -batch project_file_name.prj
```

The software runs synthesis in batch mode. Use absolute path names or a variable instead of a relative path name.

The software returns the following codes after the batch run:

- 0 - ok
- 2 - error
- 3 and above - abnormal exit (but no details).



3. If there are errors in the source files, check the standard output for messages. On Solaris/Linux systems, this is generally the monitor; on Windows systems, it is the `stdout.log` file.
4. After synthesis, check the `result_file.srr` log file for error messages about the run.

## Running Batch Mode with a Tcl Script

The following procedure shows you how to create a Tcl batch script for running synthesis. If you already have a project file set up, use the procedure described in [Running Batch Mode on a Project File, on page 876](#).

1. Create a Tcl batch script. See [Creating a Tcl Synthesis Script, on page 879](#) for details.
2. Save the file with a `*.tcl` extension to the directory that contains your source files and other project files.
3. From a command prompt, go to the directory with the files and type the following:

```
synplify -batch Tcl_script.tcl
synplify_pro -batch Tcl_script.tcl
synplify_premier -batch Tcl_script.tcl
synplify_premier_dp -batch project_file_name.prj
```

The software runs synthesis in batch mode. The synthesis (compilation and mapping) status results and errors are written to the log file `result_file.srr` for each implementation. The synthesis tool also reports success and failure return codes.

4. Check for errors.
  - For source file or Tcl script errors, check the standard output for messages. On Solaris/Linux systems, this is generally the monitor in addition to the `stdout.log` file; on Windows systems, it is the `stdout.log` file.
  - For synthesis run errors, check the `result_file_name.srr` log file. The software uses the following error codes:
    - 0 - ok
    - 2 - error
    - 3 and above - abnormal exit (but no details).

## Working with Tcl Scripts and Commands

The software uses extensions to the popular Tcl (Tool Command Language) scripting language to control synthesis and for constraint files. See the following for more information:

- [Using Tcl Commands and Scripts](#), next
- [Generating a Job Script](#), on page 879
- [Creating a Tcl Synthesis Script](#), on page 879
- [Using Tcl Variables to Try Different Clock Frequencies](#), on page 881
- [Using Tcl Variables to Try Several Target Technologies](#), on page 882
- [Running Bottom-up Synthesis with a Script](#), on page 883

You can also use synhooks Tcl scripts, as described in [Automating Flows with synhooks.tcl](#), on page 884.

### Using Tcl Commands and Scripts

1. To get help on Tcl syntax, do any of the following:
  - Refer to the online help (Help->Tcl Help) for general information about Tcl syntax.
  - Refer to the *Reference Manual* for information about the synthesis commands.
  - Type `help *` in the Tcl window for a list of all the Tcl synthesis commands. The Tcl window is not available in Synplify.
  - Type `help commandName` in the Tcl window to see the syntax for an individual command.
2. To run a Tcl script, do the following:
  - Create a Tcl script. Refer to [Generating a Job Script](#), on page 879 and [Creating a Tcl Synthesis Script](#), on page 879.
  - Run the Tcl script by either typing `source Tcl_scriptfile` in the Tcl script window, or by selecting File->Run Tcl Script, selecting the Tcl file, and clicking Open.

The software runs the selected script by executing each command in sequence. For more information about Tcl scripts, refer to the following sections.

## Generating a Job Script

You can record Tcl commands from the interface and use it to generate job scripts.

1. In the Tcl script window, type `recording -file logfile` to write out a Tcl log file.
2. Work through a synthesis session.

The software saves the commands from this session into a Tcl file that you can use as a job script or as a starting point for creating other Tcl files.

## Creating a Tcl Synthesis Script

Tcl scripts are text files with a \*.tcl extension. You can use the graphic user interface to help you create a Tcl script. Interactive commands that you use actually execute Tcl commands, which are displayed in the Tcl window as they are run. You can copy the command text and paste it into a text file that you build to run as a Tcl script. For example:

```
add_file prep2.v
set_option -technology STRATIX
set_option -part EP1SGX40D
set_option -package FC1020

project -run
```

The following procedure covers general guidelines for creating a synthesis script.

1. Use a text file editor or select File->New, click the Tcl Script option, and type a name for your Tcl script.
2. Start the script by specifying the project with the `project -new` command. For an existing project, use `project -load project.prj`.

3. Add files using the `add_file` command. The files are added to their appropriate directories based on their file name extensions (see [add\\_file, on page 1200](#) in the *Reference Manual*). Make sure the top-level file is last in the file list:

```
add_file statemach.vhd
add_file rotate.vhd
add_file memory.vhd
add_file top_level.vhd
add_file design.sdc
```

For information on constraints and vendor-specific attributes, see [Using a Text Editor for Constraint Files, on page 100](#) for details about constraint files.

4. Set the design synthesis controls and the output:
  - Use the `set_option` command for setting implementation options and vendor-specific controls as needed. See the appropriate vendor chapter in the *Synplify Reference Manual* for details.
  - Set the output file information with `project -result_file` and `project -log_file`.
5. Set the file and run options:
  - Save the project with a `project -save` command
  - Run the project with a `project -run` command
  - Open the RTL and Technology views:

```
open_file -rtl_view
open_file -technology_view
```

6. Check the syntax.
  - Check case (Tcl commands are case-sensitive).
  - Start all comments with a hash mark (#).
  - Always use a forward slash (/) in directory and pathnames, even on the Windows platform.

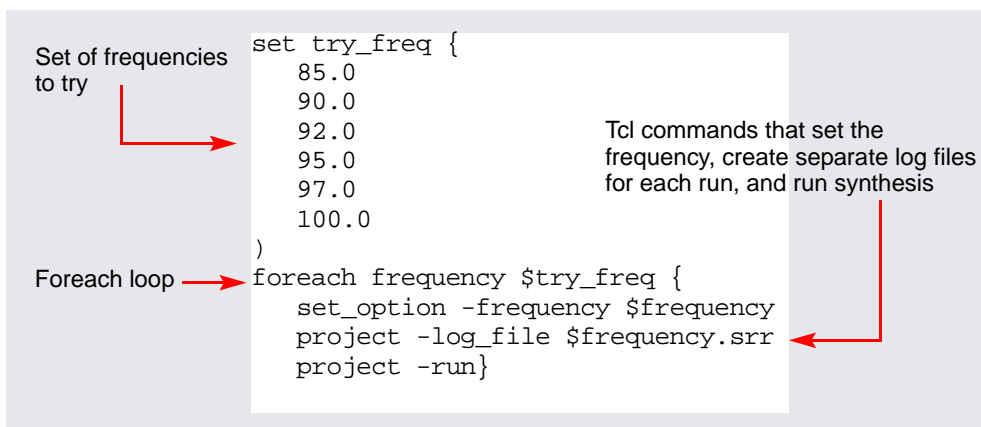
## Using Tcl Variables to Try Different Clock Frequencies

To create a single script for multiple synthesis runs with different clock frequencies, you need to create a Tcl variable for the different settings you want to try. For example, you might want to try different target technologies.

1. To create a variable, use this syntax:

```
set variable_name {
 first_option_to_try
 second_option_to_try
 ...}
```

2. Create a `foreach` loop that runs through each option in the list, using the appropriate Tcl commands. The following example shows a variable set up to synthesize a design with different frequencies. It also creates a separate log file for each run.



The following code shows the complete script:

```
project -load design.prj
set try_these {
 20.0
 24.0
 28.0
 32.0
 36.0
 40.0
}
```

```
foreach frequency $try_these {
 set_option -frequency $frequency
 project -log_file $frequency.srr
 project -run
 open_file -edit_file $frequency.srr
}
```

## Using Tcl Variables to Try Several Target Technologies

This technique used here to run multiple synthesis implementations with different target technologies is similar to the one described in [Using Tcl Variables to Try Different Clock Frequencies, on page 881](#). As in that section, you use a variable to define the target technologies you want to try.

1. Create a variable called `try_these` with a list of the technologies.

```
set try_these {
 STRATIXII CYCLONEII VIRTEX2 # list of technologies
}
```

2. Add a `foreach` loop that creates a new implementation for each technology and opens the RTL view for each implementation.

```
foreach technology $try_these {
 impl -add
 set_option -technology $technology
 project -run -fg
 open_file -rtl_view
}
```

The following code example shows the script:

```
Open a new project, set frequency, and add files.
project -new
set_option -frequency 33.3
add_file -verilog D:/test/simplestest/prep2_2.v

Create the Tcl variable to try different target technologies.
set try_these
 STRATIXII CYCLONEII VIRTEX2 # list of technologies
}
```

```
Loop through synthesis for each target technology.
foreach technology $try_these {
 impl -add
 set_option -technology $technology
 project -run -fg
 open_file -rtl_view
}
```

## Running Bottom-up Synthesis with a Script

To run bottom-up synthesis, you create Tcl scripts for individual logic blocks, and a script for the top level that reads the other Tcl scripts.

1. Create a Tcl script for each logic block. The Tcl script must synthesize the block. See [Creating a Tcl Synthesis Script, on page 879](#) for details.
2. Create a top-level script that reads the block scripts. Create the script with the with the `project -new` command.
3. Add the top-level data:
  - Add source and constraint files with the `add_file` command.
  - Set the top-level options with the `set_option` command.
  - Set the output file information with `project -result_file` and `project -log_file`.
  - Save the project with a `project -save` command.
  - Run the project with a `project -run` command.
4. Save the top-level script, and then run it using this syntax:

```
source block_script.tcl
```

When you run this command, the entire design is synthesized, beginning with the lower-level logic blocks specified in the sourced files, and then the top level.

## Automating Flows with synhooks.tcl

This procedure provides the advanced user with callbacks that let you customize your design flow or integrate with other products. For example, you might use the callbacks to send yourself email when a job is done (see [Automating Message Filtering with a Tcl Script, on page 607](#)), or to automatically copy files to another location after mapping. You can use the callback functions to integrate with a version control system, or generate the files needed to run formal verification with the Cadence Conformal tool. The procedure is based on a file called `synhooks.tcl`, which contains the Tcl callbacks.

1. Copy the `synhooks.tcl` file from the `install_dir/examples` directory to a new location.

You must copy the file to a new location so that it does not get overwritten by subsequent product installations and you can maintain your customizations from version to version. For example, copy it to `C:/work/synhooks.tcl`.

2. Define an environment variable called `SYN_TCL_HOOKS`, and point it to the location of the `synhooks.tcl` file.
3. Open the `synhooks.tcl` file in a text editor, and edit the file so that the commands reflect what you want to do. The default file contains examples of the callbacks, which provide you with hooks at various points of the design process.
  - Customize the file by deleting the ones you do not need and by adding your customized code to the callbacks you want to use. The following table summarizes the various design phases where you can use the callbacks and lists the corresponding functions. For details of the syntax, refer to [Tcl synhooks File Syntax, on page 1271](#) in the *Reference Manual*.

| Design Phase                   | Tcl Callback Function                         |
|--------------------------------|-----------------------------------------------|
| <b>Project Setup Callbacks</b> |                                               |
| Settings defaults for projects | <code>proc syn_on_set_project_template</code> |
| Creating projects              | <code>proc syn_on_new_project</code>          |
| Opening projects               | <code>proc syn_on_open_project</code>         |



| Design Phase                                                                                            | Tcl Callback Function         |
|---------------------------------------------------------------------------------------------------------|-------------------------------|
| Closing projects                                                                                        | proc syn_on_close_project     |
| <b>Application Callbacks</b>                                                                            |                               |
| Starting the application after opening a project                                                        | proc syn_on_start_application |
| Exiting the application                                                                                 | proc syn_on_exit_application  |
| <b>Run Callbacks</b>                                                                                    |                               |
| Starting a run. See <a href="#">Example: proc syn_on_start_run, on page 885</a> .                       | proc syn_on_start_run         |
| Ending a run                                                                                            | proc syn_on_end_run           |
| <b>Key Assignment Callbacks</b>                                                                         |                               |
| Setting an operation for Ctrl-F8. See <a href="#">Example: proc syn_on_press_ctrl_f8, on page 886</a> . | proc syn_on_press_ctrl_f8     |
| Setting an operation for Ctrl-F9                                                                        | proc syn_on_press_ctrl_f9     |
| Setting an operation for Ctrl-F11                                                                       | proc syn_on_press_ctrl_f11    |

- Save the file.

As you synthesize your design, the software automatically executes the function callbacks you defined at the appropriate points in the design flow.

### Example: proc syn\_on\_start\_run

The following code example gets selected files from the project browser at the start of a run:

```
proc syn_on_start_run {compile c:/work/prep2.prj rev_1} {
 set sel_files [get_selected_files -browser]
 while {[expr [llength $sel_files] > 0]} {
 set file_name [lindex $sel_files 0]
 puts $file_name
 set sel_files [lrange $sel_files 1 end]
 }
}
```

### Example: proc syn\_on\_press\_ctrl\_f8

The following code example gets all the selected files from the project browser and project directory when the Ctrl-F8 key combination is pressed:

```
proc syn_on_press_ctrl_f8 {} {
 set sel_files [get_selected_files]
 while {[expr [llength $sel_files] > 0]} {
 set file_name [lindex $sel_files 0]
 puts $file_name
 set sel_files [lrange $sel_files 1 end]
 }
}
```



#### Synopsys, Inc.

600 West California Avenue, Sunnyvale, CA 94086 USA  
Phone: +1 408 215-6000, Fax: +1 408 222-068  
[www.solvnet.com](http://www.solvnet.com)

Copyright © 2009 Synopsys, Inc. All rights reserved. Specifications subject to change without notice. Synopsys, Behavior Extracting Synthesis Technology, Certify, DesignWare, HDL Analyst, Identify, SCOPE, "Simply Better Results", SolvNet, Synplicity, the Synplicity logo, Synplify, Synplify ASIC, Synplify Pro, Synthesis Constraints Optimization Environment, and VCS are registered trademarks of Synopsys, Inc. BEST, Confirma, HAPS, HapsTrak, High-performance ASIC Prototyping System, IICE, MultiPoint, Physical Analyst, System Designer, and TotalRecall are trademarks of Synopsys, Inc. All other names mentioned herein are trademarks or registered trademarks of their respective companies.

# Index

---

*See also* RAMs  
multi-port

## Symbols

.acf file [106](#)  
.adc file [737](#)  
.bit file [204](#)  
.bmm file [204](#)  
.edf file [804](#)  
.lmf file [784](#)  
.lpf file [106](#), [796](#)  
.ndf file [804](#)  
.sdc file [213](#)  
.vhm file [828](#)  
.vqm file  
  Clearbox [175](#)  
\_conv.prj file [261](#)  
\_conv.sdc file [262](#)

## Numerics

3rd party vendor tools  
  invoking [838](#)

## A

Actel  
  ACTgen macros [761](#)  
  I/O pad type [228](#)  
  macro libraries [760](#)  
  output netlist [836](#)  
  pin numbers for bus ports [833](#)  
ACTgen macros [761](#)  
Adder/Subtractor  
  compiling with SYNCORE [130](#)  
adders  
  SYNCORE [130](#)  
  wide. *See* wide adders/subtractors.

adjust pin view (Design Planner) [492](#)  
alspin  
  bus port pin numbers [833](#)  
Alt key  
  column editing [86](#)  
  mapping [650](#)  
Altera  
  Apex design tips [765](#)  
  Clearbox. *See* Clearbox  
  converting pin assignments to SDC [348](#)  
  converting PIN files [348](#)  
  design tips [764](#)  
  EAB VHDL mapping example [768](#)  
  EABs [767](#)  
  ESBs [767](#)  
  FLEX design tips [765](#)  
  forward-annotation [106](#)  
  grey box *See* grey box  
  I/O packing [774](#)  
  instantiating LPMs as black boxes [417](#)  
  LCELLs. *See* LCELLs.  
  LPM megafunction example (Verilog)  
    [417](#)  
  LPM megafunction example (VHDL) [419](#)  
  mapping EABs, Verilog [767](#)  
  multi-port RAMs [388](#)  
  netlist [836](#)  
  packing I/Os [774](#)  
  physical synthesis flows [60](#)  
  place-and-route option file [337](#)  
  PLLs. *See* altplls  
  Quartus batch mode [781](#)  
  Quartus integrated flow [779](#)  
  Quartus interactive flow [780](#)  
  RAMs [386](#)  
  ROMs [765](#)  
  shift registers [410](#)  
  simulating LPMs [777](#)  
  Stratix III LUTRAMs [389](#)  
  Stratix RAM [379](#)  
  Verilog LPM library [423](#)  
Altera APEX and APEX II (Design  
  Planner)

---

- tips for physical constraints [523](#)
- Altera MegaWizard
  - generating LPM files [417](#)
- Altera shift registers
  - report [412](#)
- Altera STRATIX (Design Planner)
  - additional tips [519](#)
- altera\_implement\_in\_eab attribute [767](#)
- altera\_implement\_in\_esb attribute [767](#)
- altpll
  - component declaration files [769](#)
  - constraints [770](#)
  - using [769](#)
- altshift\_tap, set implementation style [410](#)
- ALTSYNCRAM for LPMs [417](#)
- ALTSYNCRAM, Altera Stratix [380](#)
- analysis design constraint file (.adc) [737](#)
- analyzing netlists (Physical Analyst) [721](#)
- APEX
  - netlist [836](#)
- archiving projects [315](#)
- area estimation, Design Planner [488](#)
- area, optimizing [427](#)
- asterisk wildcard
  - Find command [640](#)
- attributes
  - adding [304](#)
  - adding in constraint files [102](#)
  - adding in SCOPE [308](#)
  - adding in Verilog [307](#)
  - adding in VHDL [305](#)
  - altera\_implement\_in\_eab [767](#)
  - altera\_implement\_in\_esb [767](#)
  - collections [240](#)
  - effects of retiming [437](#)
  - for FSMs [369](#), [458](#)
  - pipelining [431](#)
  - syn\_hier (on compile points) [568](#)
  - VHDL package [305](#)
- audience for the document [26](#)
- auto constraints, using [251](#)
- Auto route cross probe insts command [713](#)

## B

- B.E.S.T [654](#)
- back annotation
  - coreloc.sdc constraint file [353](#)
  - place-and-route data [353](#)
- backslash
  - escaping dot wildcard in Find command [640](#)
  - in Find command (Physical Analyst) [706](#)
- batch mode [876](#)
- Behavior Extraction Synthesis Technology. *See* B.E.S.T
- bit slicing [543](#)
  - legal primitives [544](#)
- bit-slicing
  - critical paths in regions [531](#)
- black boxes [358](#)
  - adding constraints [362](#)
  - adding constraints in SCOPE [365](#)
  - adding constraints in Verilog [365](#)
  - adding constraints in VHDL [363](#)
  - EDIF naming consistency [367](#)
  - EDK cores [206](#)
  - encrypted Xilinx cores [207](#)
  - for IP cores [804](#)
  - gated clock attributes [471](#)
  - instantiating in Verilog [358](#)
  - instantiating in VHDL [360](#)
  - internal startup blocks [367](#)
  - pin attributes [366](#)
  - prepared component method (Altera) [422](#)
  - SOPC components [193](#)
  - specifying timing information for Xilinx cores [804](#)
  - timing constraints [362](#)
  - Xilinx physical synthesis [78](#)
- block mult
  - assigning to regions [539](#)
  - viewing resources in Design Planner [539](#)
- block multipliers (Design Planner) [539](#)
- block RAM
  - assigning to regions [534](#)
  - dual-port, mapping with registered address [392](#)
  - glue logic in [393](#)
  - mapping dual port coding style [398](#)

- mapping ROM (Xilinx) 398
  - mapping to single-output dual-port 398
  - mapping to single-port 396
  - single-port
    - mapping with registered address 391
    - using registered addresses 391
    - using registered output 393
  - block RAM (Synplify Premier) 534
  - block RAM regions (Design Planner)
    - assigning 536
    - creating 535
  - bookmarks
    - in source files 86
    - using in log files 598
  - bottom-up design and compile-point synthesis 560
  - BRAMs (Synplify Premier) 534
  - breaking up large primitives (Synplify Premier) 543
  - browsers 627
  - buffering
    - controlling 448
  - BUFG
    - for fanouts 449
  - BUFGDLL 822
  - BUFGMUX\_1 inference 818
  - buses
    - INIT values for bits 811
    - RLOC values for bits 821
- ## C
- c\_diff command, examples 243
  - c\_intersect command, examples 243
  - c\_list command
    - different from c\_print 248
    - example 250
    - using 249
  - c\_print command
    - different from c\_list 248
    - using 249
  - c\_syndiff command, examples 243
  - c\_union command, examples 242
  - callback functions, customizing flow 884
  - carry chain DRC (Synplify Premier) 530
  - carry chains
    - inferring 788
  - case sensitivity
    - Find command (Tcl) 245
  - cells
    - enhancing display in Physical Analyst 691
  - chip regions (Design Planner) 510
  - Clearbox
    - adding instantiated file 174
    - implementing megafunctions with 165
    - inferring megafunctions 166
    - instantiating megafunctions 170
    - instantiating with netlist 173
    - SOPC components 193
    - using 165
    - using instantiated netlists in Quartus 175
  - clock buffers 822
  - clock constraints
    - edge-to-edge delay 220
    - false paths 235
    - setting 220
  - clock DLLs 822
  - clock domains
    - setting up 225
  - clock groups
    - effect on false path constraints 235
    - for global frequency clocks 222
    - Xilinx DCMs and DLLs 225
  - clock path skew (Synplify Premier) 610
  - clock pins (Design Planner) 498
  - clock skew (Synplify Premier) 610
  - clock skew example (Synplify Premier) 824, 825
  - clock trees 733
  - clocks
    - asymmetrical 223
    - defining 222
    - for DCMs and DLLs 224
    - for PLLs 224
    - frequency 223
    - gated. *See* gated clocks
    - gated. *See* gated clocks.
    - implicit false path 235
    - limited resources 225
    - overriding false paths 236

---

- collections
  - adding attributes to [240](#)
  - adding objects [241](#)
  - concatenating [241](#)
  - constraints [240](#)
  - copying [249](#)
  - creating from common objects [242](#)
  - creating from other collections [239](#)
  - creating in SCOPE [238](#)
  - creating in Tcl [241](#)
  - crossprobing objects [239](#)
  - definition [237](#)
  - diffing [242](#)
  - highlighting in HDL Analyst views [247](#)
  - iterating through objects [250](#)
  - listing objects [249](#)
  - listing objects and properties [248](#)
  - listing objects in a file [249](#)
  - listing objects in columnar format [248](#)
  - listing objects with `c_list` [248](#)
  - special characters [244](#)
  - Tcl window and SCOPE comparison [237](#)
  - using Tcl `expand` command [246](#)
  - using Tcl `find` command [244](#)
  - viewing [247](#)
- column editing [86](#)
- commands
  - Auto route cross probe insts (Physical Analyst) [713](#)
  - Expand Path Backward [755](#)
  - Expand Path Forward [753](#)
  - Go to Location (Physical Analyst) [708](#)
  - Highlight Visible Net Instances (Physical Analyst) [725](#)
  - Markers [709](#)
  - Select Net Instances [725](#)
  - Send Crossprobes when selecting (Physical Analyst) [713](#)
  - Signal Flow [696](#)
  - `slice_primitive` [543](#)
- comments
  - source files [86](#)
- compile point types
  - hard [565](#)
  - locked [565](#)
  - locked,partition [567](#)
- compile points
  - advantages [563](#)
  - child [563](#)
  - constraints for forward-annotation [572](#)
  - constraints, internal [572](#)
  - creating constraint file [578](#)
  - defining in constraint files [575](#)
  - definition [562](#)
  - feature summary [567](#)
  - nesting [562](#)
  - optimization [571](#)
  - order of synthesis [571](#)
  - parent [563](#)
  - preserving with `syn_hier` [579](#)
  - Quartus II Incremental Compilation [859](#)
  - resynthesis [571](#)
  - setting constraints [579](#)
  - `syn_hier` [568](#)
  - types [564](#)
  - using `syn_allowed_resources` attribute [568](#)
  - Xilinx incremental flows [866](#)
- compile points types
  - soft [564](#)
- compile-point flow
  - Xilinx [583](#)
- compile-point synthesis [560](#)
  - and bottom-up flow [561](#)
  - bottom-up flow [560](#)
  - interface logic models [570](#)
- compile-point synthesis flow [573](#)
  - compiling the design for initialization [574, 575](#)
  - defining compile points [575](#)
  - setting constraints [577](#)
  - setting implementation options [574](#)
- compiler directives (Verilog)
  - specifying [300](#)
- Conformal [847](#)
- connectivity-based timing report [743](#)
- constants
  - extracting from VHDL source code [302](#)
- constraint file
  - `coreloc.sdc` [354](#)
- constraint files [98](#)
  - See also* SCOPE
  - Altera QSF [253](#)
  - applying to a collection [240](#)
  - black box [362](#)
  - compile point [572](#)
  - creating in a text editor [100](#)
  - creating with SCOPE [212](#)

- defining clocks [101](#)
- defining register delays [101](#)
- editing [215](#)
- effects of retiming [437](#)
- forward-annotating [105](#)
- opening [213](#)
- options [295](#)
- setting for compile points [579](#)
- specifying through points [232](#)
- types of [214](#)
- vendor-specific [105](#)
- when to use [98](#)
- constraints
  - Actel physical synthesis [763](#)
  - altplls [770](#)
  - checking [104](#)
  - translating Altera QSF [253](#)
  - translating with ucf2sdc\_old [264](#)
  - translating Xilinx constraints for logic synthesis [255](#)
  - translating Xilinx constraints for physical synthesis [258](#)
- constraints (Actel)
  - checking [349](#)
- context
  - for object in filtered view [657](#)
- context window (Physical Analyst) [687](#)
- control panel
  - Physical Analyst view [684](#)
- CoreGen [804](#)
- coreloc.sdc [354](#)
- coreloc.sdc file [353](#)
- cores
  - incorporating Xilinx EDK cores [199](#)
  - integrating subsystems as submodules (Xilinx) [205](#)
  - integrating subsystems as top level modules (Xilinx) [204](#)
- cores, instantiating in Xilinx designs [804](#)
- counters
  - compiling with SYNCore [137](#)
  - SYNCore [137](#)
- critical paths
  - splitting between regions [527](#)
- critical paths [529](#)
  - carry chains in regions [530](#)
  - cascading cells in regions [530](#)
  - delay [734](#)
  - enable registers in regions [529](#)
  - flat view [734](#)
  - hierarchical view [734](#)
  - islands [743](#)
  - large muxes in regions [533](#)
  - multiple paths in regions [533](#)
  - pipelining in regions [532](#)
  - slack time [734](#)
  - splitting between regions [528](#)
  - using -route [428](#)
  - viewing [733](#)
  - with bit-slicing in regions [531](#)
- critical paths (Design Planner)
  - assigning to regions [515](#)
- critical paths (Island Timing Analyst)
  - assigning to a region [516](#)
- critical paths (Physical Analyst) [750](#)
  - tracing backward [755](#)
  - tracing forward [753](#)
- critical paths (Synplify Premier)
  - using the island timing report [516](#)
- crossprobing [645](#)
  - and retiming [437](#)
  - collection objects [239](#)
  - filtering text objects for [651](#)
  - from FSM viewer [652](#)
  - from log file [599](#)
  - from message viewer [604](#)
  - from text files [649](#)
  - from text files to Physical Analyst [716](#)
  - Hierarchy Browser [646](#)
  - importance of encoding style [652](#)
  - paths [650](#)
  - Physical Analyst view [713](#)
  - RTL view [647](#)
  - Technology view [647](#)
  - Technology view to Physical Analyst [719](#)
  - Text Editor view [647](#)
  - text file example [650](#)
  - to FSM Viewer [652](#)
  - to place-and-route file [623](#)
  - Verilog file [647](#)
  - VHDL file [647](#)
  - View Cross Probing commands [713](#)
  - within RTL and Technology views [646](#)
- crossprobing (Physical Analyst)
  - auto route crossprobing [719](#)
  - RTL view [713, 717](#)

---

- Technology view [713](#)
- crossprobing commands (Synplify Premier)
  - Physical Analyst view [713](#)
- current level
  - expanding logic from net [661](#)
  - expanding logic from pin [661](#)
  - searching current level and below [637](#)
- custom folders
  - creating [279](#)
  - hierarchy management [279](#)
- customization
  - callback functions [884](#)

## D

- data block [145](#)
- data key [145](#)
- DCMs
  - defining clocks [224](#)
- default enum encoding [301](#)
- define\_attribute [310](#)
- define\_clock\_constraint [101](#)
- define\_compile\_point\_constraint [569](#)
- define\_current\_design\_constraint [569](#)
- define\_false\_paths\_constraint [101](#)
- define\_input\_delay\_constraint [101](#)
- define\_multicycle\_path\_constraint [101](#)
- define\_output\_delay\_constraint [101](#)
- define\_reg\_input\_delay\_constraint [101](#)
- define\_reg\_output\_delay\_constraint [101](#)
- design flow
  - customizing with callback functions [884](#)
- design guidelines [426](#)
- design hierarchy
  - viewing [655](#)
- design plan
  - options [295](#)
- Design Plan Editor view
  - preserving region resources [509](#)
- design plan file
  - creating [494](#)
  - logic synthesis [39](#)
  - physical synthesis [49](#), [494](#)

- Design Planner [488](#)
  - assigning pins [495](#)
  - creating chip regions [510](#)
  - displaying IP core areas [510](#)
  - guidelines [487](#)
  - logic synthesis [39](#)
  - opening [488](#)
  - physical synthesis [49](#)
- design planning [488](#)
- design size
  - amount displayed on a sheet [624](#)
- design views
  - moving between views [623](#)
- DesignWare [107](#)
  - Verilog function inferencing [108](#)
  - VHDL component instantiation [109](#)
- detail placement
  - Xilinx physical synthesis [78](#)
- device options
  - See also* implementation options
- directives
  - adding [304](#)
  - adding in Verilog [307](#)
  - adding in VHDL [305](#)
  - black box [363](#), [364](#)
  - for FSMs [369](#)
  - specifying for Verilog compiler [300](#)
  - syn\_state\_machine [457](#)
  - syn\_tco [364](#)
    - adding black box constraints [363](#)
  - syn\_tpd [364](#)
    - adding black box constraints [363](#)
  - syn\_tsu [364](#)
    - adding black box constraints [363](#)
- Dissolve Instances command
  - using [668](#)
- dissolving [668](#)
- DLLs
  - defining clocks [224](#)
- dot wildcard
  - Find command [640](#)
- drivers
  - preserving duplicates with syn\_keep [440](#)
  - selecting [664](#)
- DSP blocks (Design Planner) [540](#)
- dual-port RAMs



- 388
  - block RAMs with single registered output, Xilinx 398
  - Stratix 381
- E**
- EABs
    - VHDL mapping example 768
  - EABs, inferring 767
  - EDIF
    - structural, for Xilinx IP cores 804
    - synthesizing 826
  - EDIF files
    - reoptimizing with 826
  - Editing window 85
  - EDK
    - EDK hardware flow 200
    - edk2syn. *See* edk2syn
    - embedding processors as subsystems 199
    - encrypted cores 207
    - flow 199
    - ISE flow 200
    - specifying cores as black boxes 206
    - specifying cores as white boxes 206
    - Synplify-EDK flow 199
    - synthesizing cores 200
  - EDK standalone hardware development flow 209
  - EDK-ISE hardware development flow 208
  - EDN core 195
  - emacs text editor 90
  - Embedded development kit. *See* EDK
  - embedded processors
    - integrating subsystems as submodules (Xilinx) 205
    - integrating subsystems as top level modules (Xilinx) 204
  - embedded processors, Xilinx. *See* EDK 199
  - encoding styles
    - and crossprobing 652
    - default VHDL 301
    - FSM Compiler 455
  - encrypted IP objects (Physical Analyst)
    - identifying cells 711
  - encryptIP script
    - controlling output 151
    - encrypting IP 150
    - output methods 151
  - enhanced optimization
    - compared to fast synthesis 35
    - using 36
  - environment variables
    - PAR\_BELDLYRPT 333
    - SYN\_TCL\_HOOKS 884
  - equivalence checking
    - VIF file 844
  - equivalency checking
    - handling failure 848
  - error messages
    - gated clock report 473
  - errors
    - definition 85
    - filtering 603
    - sorting 603
    - source files 84
    - Verilog 84
    - VHDL 84
  - ESBs, inferring 767
  - Expand command
    - connection logic 664
    - connections in Physical Analyst 727
    - pin and net logic 660
    - using 661
  - Expand command (Physical Analyst) 722
  - expand command (Tcl). *See* Tcl expand command
  - Expand Inwards command
    - using 661
  - Expand Path Backward command 755
  - Expand Path Forward command 753
  - Expand Paths command
    - different from Isolate Paths 664
  - expand pin view (Design Planner) 491
  - Expand to Register/Port command
    - using 661
  - Expand to Register/Port command (Physical Analyst) 722
  - expanding
    - connections 664
    - connections (Physical Analyst) 727

---

pin and net logic [660](#)  
pin and net logic (Physical Analyst) [722](#)

## F

### false paths

defining between clocks [235](#)  
I/O paths [236](#)  
impact of clock group assignments [235](#)  
overriding [236](#)  
ports [235](#)  
registers [235](#)  
setting constraints [235](#)

### fanout

replicating instances (Design Planner)  
[515](#)

### fanouts

buffering vs replication [448](#)  
hard limits [447](#)  
soft global limit [446](#)  
soft module-level limit [447](#)  
using `syn_keep` for replication [441](#)  
using `syn_maxfan` [446](#)

### fast synthesis

using [482](#), [483](#)

### feature comparison

FPGA tools [23](#)

### features

Synplify [23](#)  
Synplify Premier [23](#)  
Synplify Pro [23](#)

### FIFOs

compiling with SYNCORE [114](#)

### files

`.acf` [106](#)  
`.adc` [737](#)  
`.lpf` [106](#), [796](#)  
`.prf` file [606](#)  
`.sdc` [213](#)  
altpll component declarations [769](#)  
filtered messages [607](#)  
`fsm.info` [456](#)  
`log` [596](#)  
message filter (`prf`) [606](#)  
output [836](#)  
`rom.info` [630](#)  
searching [312](#)  
`statemachine.info` [674](#)  
`synhooks.tcl` [884](#)

### Tcl [878](#)

*See also* Tcl commands

Tcl batch script [877](#)

Filter Schematic command, using [658](#)

Filter Schematic icon, using [658](#)

### filtering [658](#)

advantages over flattening [658](#)  
using to restrict search [637](#)

filtering (Physical Analyst) [721](#)

### Find command

[637](#)

browsing with [636](#)

hierarchical search [638](#)

long names [636](#)

message viewer [603](#)

Physical Analyst view [704](#)

reading long names [639](#)

search scope, effect of [640](#)

search scope, setting [638](#)

searching the mapped database [639](#)

searching the output netlist [643](#)

setting limit for results [639](#)

using in RTL and Technology views [637](#)

using wildcards [640](#)

wildcard examples [642](#)

### Find command (Physical Analyst)

using Filter Search option [707](#)

using wildcards [706](#)

### Find command (Tcl)

*See also* Tcl find command

### finding information

information organization [27](#)

### finding objects

Physical Analyst view [704](#)

Fix Gated Clocks option. *See* gated clocks

### Flatten Current Schematic command

transparent instances [666](#)

using [666](#)

### Flatten Schematic command

using [666](#)

### flattening [665](#)

*See also* dissolving

compared to filtering [658](#)

hidden instances [667](#)

transparent instances [666](#)

using `syn_hier` [444](#)

- FLEX netlist [836](#)
  - floorplan file. *See* sfp file, design plan file
  - floorplan. *See* Design Planner
  - foreach command [250](#)
  - forward annotation
    - frequency constraints in Xilinx [798](#)
    - vendor-specific constraint files [105](#)
  - forward-annotation
    - compile point constraints [572](#)
    - constraints [795](#)
    - Xilinx core files [198](#)
  - frequency
    - clocks [223](#)
    - defining for non-clock signals [224](#)
    - internal clocks [223](#)
    - setting global [294](#)
  - from constraints
    - specifying [231](#)
  - FSM Compiler
    - advantages [454](#)
    - enabling [455](#)
  - FSM encoding
    - user-defined [371](#)
    - using syn\_enum\_encoding [371](#)
  - FSM Explorer [453](#)
    - running [459](#)
    - when to use [453](#)
  - FSM view
    - crossprobing from source file [649](#)
  - FSM Viewer [670](#)
    - crossprobing [652](#)
  - fsm.info file [456](#)
  - FSMs
    - See also* FSM Compiler, FSM Explorer
    - attributes and directives [369](#)
    - defining in Verilog [368](#)
    - defining in VHDL [369](#)
    - definition [367](#)
    - optimizing with FSM Compiler [453](#)
    - properties [674](#)
    - state encodings [673](#)
    - transition diagram [671](#)
    - viewing [671](#)
- ## G
- gated clocks
    - attributes for black boxes [471](#)
    - conversion example [467](#)
    - conversion report [472](#)
    - conversion requirements [467](#)
    - defining [226](#)
    - error messages in report [473](#)
    - examples [465](#)
    - procedure for fixing [469](#)
    - restrictions [475](#)
    - Synplicity approach [464](#)
  - generated-clock conversion [477](#)
  - generics
    - extracting from VHDL source code [302](#)
  - global comments
    - initializing Xilinx RAM [406](#)
  - global optimization options [292](#)
  - global placement
    - Xilinx physical synthesis [78](#)
  - global range
    - defining [748](#)
  - global sets/resets
    - Xilinx designs [801](#)
  - glue logic
    - Altera Stratix RAM [380](#)
  - Go to Location command [708](#)
    - adding markers [710](#)
  - graph-based physical synthesis
    - Altera [60](#)
    - description [42](#)
    - logic synthesis validation phase [43](#)
    - physical synthesis phase [43](#)
    - Xilinx [69](#)
  - graph-based physical synthesis flow
    - Actel [52](#)
  - grey box
    - netlist file [179](#)
  - grey box flow
    - MegaCore with greybox netlist [179](#)
  - grey boxes
    - using [175](#)
  - greybox flow
    - MegaCore with IP package [181](#)
    - NIOS II cores [186](#)
    - SOPC cores [186](#)
  - group range (Island Timing report)
    - defining [748](#)

---

GSR resources [787](#)

GSR, Xilinx [800](#)

## H

### HDL Analyst

*See also* RTL view, Technology view

critical paths [733](#)

crossprobing [645](#)

filtering schematics [658](#)

Push/Pop mode [630](#), [633](#)

traversing hierarchy with mouse strokes [628](#)

traversing hierarchy with Push/Pop mode [630](#)

using [654](#)

### HDL Analyst tool

deselecting objects [621](#)

selecting/deselecting objects [620](#)

### HDL Analyst views

highlighting collections [247](#)

### help

information organization [27](#)

### hidden instances

consequences of saving [656](#)

flattening [667](#)

restricting search by hiding [637](#)

specifying [656](#)

status in other views [656](#)

### hierarchical design

expanding logic from nets [661](#)

expanding logic from pins [661](#)

### hierarchical instances

dissolving [668](#)

hiding. *See* hidden instances, Hide Instances command

multiple sheets for internal logic [657](#)

pin name display [659](#)

viewing internal logic [656](#)

### hierarchical objects

pushing into with mouse stroke [629](#)

traversing with Push/Pop mode [630](#)

### hierarchical search [637](#)

### hierarchy

flattening [666](#)

netlist restructuring [331](#)

traversing [627](#)

### hierarchy browser

clock trees [733](#)

controlling display [623](#)

crossprobing from [646](#)

defined [627](#)

finding objects [635](#)

traversing hierarchy [627](#)

hierarchy management (custom folders) [279](#)

high fanout in regions [529](#)

Highlight Visible Net Instances command [725](#)

### hyper source

IP design hierarchy [91](#)

## I

I/O insertion [452](#), [794](#)

VHDL manual (Xilinx) [817](#)

### I/O locations

assigning automatically (Xilinx) [812](#)

manually assigning (Xilinx) [817](#)

### I/O pads

specifying I/O standards [228](#)

### I/O paths

false path constraint [236](#)

I/O standards [228](#)

### I/Os

auto-constraining [252](#)

constraining [227](#)

packing in Altera designs [774](#)

packing in Xilinx designs [807](#)

preserving [453](#), [794](#)

specifying pad type (Xilinx) [825](#)

Verilog black boxes [358](#)

VHDL black boxes [360](#)

### I/Os (Design Planner)

critical paths from pin-locked I/Os [515](#)

### Identify

prototyping design flow [80](#)

### implementation options [289](#)

design plan file [295](#)

device [289](#)

global frequency [294](#)

global optimization [292](#)

netlist optimizations [330](#)

part selection [289](#)

specifying results [296](#)

- implementations
    - copying 286
    - deleting 286
    - multiple. *See* multiple implementations.
    - overwriting 286
    - renaming 286
  - include paths
    - updating older project files 277
  - incremental flows
    - partition (Xilinx) 863
    - SmartGuide 863
  - incremental flows (Xilinx) 862
    - incremental flow 862
  - incremental synthesis
    - locked, partition compile points 567
  - inference
    - BUFGMUX/BUFGMUX\_1 818
  - INIT property
    - initializing Xilinx RAMs, Verilog 405
    - initializing Xilinx RAMs, VHDL 407
    - specifying with attributes 408
  - INITvalues
    - Xilinx registers 810
  - input constraints, setting 226
  - instances
    - preserving with `syn_noprune` 440
    - properties 617
    - properties of pins 617
  - instances (Physical Analyst)
    - adding markers 709
    - displaying instances 690
  - interactive Island Timing Analyst 747
  - ILM *See* interface logic models
  - interface logic models 570
  - IP
    - directory structure for package 155
    - encrypting with ReadyIP 143, 148
    - evaluating directly from the synthesis tool 158
    - from EDK 199
    - package file list 155
    - packaging for evaluation 153
    - supplying vendor information 157
    - SYNCore adders 130
    - SYNCore counters 137
    - SYNCore subtractors 130
    - system-level models 157
  - IP core areas (Design Planner) 510
  - IP cores 804
    - Xilinx physical synthesis 78
  - IP design hierarchy
    - hyper source 91
  - IP vendors
    - encrypting IP 148
  - IPs
    - Altera 161
  - IP-XACT models 153
  - ISE
    - generating EDK cores 208
  - Island Timing Analyst 743
    - generating island timing report 745, 747
    - interactive 747
    - supported technologies 743
  - island timing report
    - defining group range and global range 748
    - Island Timing Analyst 747
    - using critical paths 516
  - islands
    - timing report 743
  - Islands/Paths Summary
    - crossprobing in HDL Analyst 745
    - reordering the columns 744
    - selecting islands or paths 745
  - Isolate Paths command
    - different from Expand Paths 664, 665
  - ispLEVER
    - forward-annotating constraints for 795
- ## K
- key assignments
    - customizing 885
  - key block 146
  - keywords
    - completing words in Text Editor 86
- ## L
- Lattice
    - constraint file 106
    - forward annotation 105

---

- I/O insertion [452](#)
- macro libraries [785](#)
- PICs [788](#)
- Lattice netlist [836](#)
- lcell primitive
  - Clearbox [165](#)
- LCELLs
  - mapping for area [783](#)
  - mapping for performance [783](#)
  - mapping logic to [781](#)
- libraries
  - translating Synopsys components [107](#)
  - Xilinx post-synthesis simulation [828](#)
- location constraints
  - RLOC\_ORIGIN [821](#)
  - RLOCs with synthesis attribute [821](#)
  - RLOCs with xc\_attributes [819](#)
- log file
  - gated clock conversion report [472](#)
  - gated clock error messages [473](#)
  - physical synthesis [678](#)
- log files
  - checking FSM descriptions [460](#)
  - checking information [596](#)
  - crossprobing to Physical Analyst [716](#)
  - pipelining description [432](#)
  - retiming report [436](#)
  - setting default display [596](#)
  - shift register report (Altera) [412](#)
  - state machine descriptions [455](#)
  - viewing [596](#)
- Log Watch window [600](#)
  - moving [600](#), [602](#)
  - multiple implementations [286](#)
  - resizing [600](#), [602](#)
- logic
  - expanding between objects [664](#)
  - expanding from net [661](#)
  - expanding from net (Physical Analyst) [725](#)
  - expanding from pin [661](#), [722](#)
- logic preservation
  - syn\_hier [444](#)
  - syn\_keep for nets [440](#)
  - syn\_keep for registers [440](#)
  - syn\_noprune [440](#)
  - syn\_preserve [440](#)
- logic synthesis
  - in Actel physical synthesis flow [56](#)
  - in Altera physical synthesis flow [65](#)
  - in physical synthesis flows [35](#)
  - in Xilinx physical synthesis flow [74](#)
  - translating UCF constraints [255](#)
  - with design plan [39](#)
- logical folders
  - creating [279](#)
- LPM\_RAM\_DQ
  - VHDL example [422](#)
- LPMs
  - Altera megafunction example (Verilog) [417](#)
  - Altera megafunction example (VHDL) [419](#)
  - black box method simulation flow [777](#)
  - comparison of Altera instantiation methods [416](#)
  - generics method, Cypress [421](#)
  - in .vqm [417](#)
  - including in physical synthesis [161](#)
  - instantiating as black boxes [416](#)
  - instantiating as black boxes (Altera) [417](#)
  - instantiating with a Verilog library (Altera methodology) [417](#)
  - instantiating with a Verilog library (Synplicity methodology) [423](#)
  - instantiating with VHDL prepared components [421](#)
  - prepared components (Altera), example [421](#)
  - using in Altera simulation flows [777](#)
  - Verilog library simulation flow [778](#)
  - VHDL prepared component simulation flow [778](#)
  - VHDL prepared components instantiation example [421](#)
- LPMs, Altera [416](#)
- LUTRAMs, inferring [389](#)

## M

- mac\_mult primitive
  - Clearbox [165](#)
- mac\_out primitive
  - Clearbox [165](#)
- macro libraries
  - Lattice [785](#)
- macro libraries (Xilinx) [798](#)

- macros (Xilinx) 798
  - manhattan distance (Physical Analyst) 711
  - Map Logic to LCELLs option 781
  - mapping EABs
    - Verilog 767
  - markers (Physical Analyst)
    - adding 709
    - adding with Go to Location 710
    - deleting 710
    - finding objects with 708
    - measuring with 711
    - moving 710
    - navigating between 711
    - using 709
  - Markers command 709
  - Max netlist 836
  - Max+Plus II
    - configuring (FLEX, ACEX1K) 781
    - configuring (MAX) 784
    - configuring for area 783
    - configuring for speed 783
  - maximum parallel jobs 588
  - MegaCore
    - grey box flow with grey box netlist 179
    - greybox flow with IP package 181
  - megafunctions
    - altplls 769
    - grey boxes 175
    - including in physical synthesis 161
    - inferring Clearbox information 166
    - instantiating Clearbox 170
    - instantiating Clearbox with netlist 173
    - using Clearbox 178
    - using grey box netlist 179
  - Megawizard
    - altplls 769
  - memory usage
    - maximizing with HDL Analyst 670
  - Message viewer
    - filtering messages 604
    - keyboard shortcuts 603
    - saving filter expressions 606
    - searching 603
    - using 602
    - using the F3 key to search forward 603
    - using the Shift-F3 key to search backward 603
  - messages
    - filtering 604
    - saving filter information from command line 606
    - saving filter information from GUI 606
    - writing messages to file 607
  - mixed language files 95
  - restrictions 95
  - models
    - for DesignWare components 107
  - mouse strokes
    - pushing/popping objects 628
  - mouse strokes (Physical Analyst)
    - navigating between views 686
  - multicycle constraints
    - forward-annotating 795
  - multicycle paths
    - setting constraints 221
  - multiple implementations 285
    - running from project 286
    - running from workspace 288
  - multipliers
    - pipelining restriction 429
  - multipliers, pipelining 429
  - multi-port RAMs
    - See also* dual-port RAMs
    - Altera Stratix 388
  - multiprocessing
    - maximum parallel jobs 588
  - multisheet schematics 621
    - for nested internal logic 657
    - searching just one sheet 637
    - transparent instances 622
- ## N
- name spaces
    - output netlist 643
    - technology view 638
  - navigating among design views 623
  - ncf file
    - cores 198
    - output physical constraints 228
    - using as input for logic design 256

---

ncf files  
  translating to sdc 261

netlist restructure files  
  specifying 331

netlists  
  restructuring options 330

netlists (Physical Analyst)  
  analyzing 721

netlists for different vendors 836

nets  
  expanding logic from 661  
  preserving for probing with syn\_probe 440  
  preserving with syn\_keep 440  
  properties 617  
  selecting drivers 664

nets (Physical Analyst)  
  adding markers 709  
  expanding logic from 725  
  resetting the display 696  
  routing 694  
  selecting instances 725  
  signal flow 696  
  unfiltering for Find command 704

New property 619

NGC cores 195

NGO core 195

NIOS II, importing as greybox 186

non-secure core flow  
  synthesis 196

notes  
  filtering 603  
  sorting 603

notes, definition 85

nram primitive. *See* dual-port RAMs, multi-port RAMs

## O

objects  
  finding on current sheet 637  
  flagging by property 618  
  selecting/deselecting 620

objects (Physical Analyst)  
  finding 704  
  finding by location 708

  overlapping 697  
  select overlapping 697  
  selecting 696

OpenIP 148

optimization  
  for area 427  
  for timing 428  
  generated clock 477  
  logic preservation. *See* logic preservation.  
  mapper effort. *See* fast synthesis 428  
  preserving hierarchy 444  
  preserving objects 440  
  tips for 426

optimizing  
  enhanced logic optimizations 37

options file (place-and-route) 337, 340

OR 233

output constraints, setting 226

output files 836  
  specifying 296

output netlists  
  finding objects 643

## P

p\_nram primitive. *See* dual-port RAMs, multi-port RAMs, nram primitive

package library, adding 272

pad types  
  industry standards 228

PAR\_BELDLYRPT 333

parameters  
  extracting from Verilog source code 299

part selection options 289

partition flow (Xilinx)  
  diagram 865

partition flow, Xilinx 863

partitioning (Synplify Premier)  
  bit slicing 543

path constraints  
  false paths 235

pathnames  
  using wildcards for long names (Find) 639



- paths
  - crossprobing [650](#)
  - tracing between objects [664](#)
  - tracing from net [661](#)
  - tracing from pin [661](#)
- paths (Physical Analyst)
  - tracing between objects [727](#)
  - tracing from net [725](#)
  - tracing from pin [722](#)
- pattern matching
  - Find command (Tcl) [245](#)
- pattern searching [312](#)
- PDF
  - cutting from [86](#)
- Physical Analyst
  - analyzing netlists [721](#)
  - context window [687](#)
  - control panel [684](#)
  - crossprobing from text files [716](#)
  - crossprobing RTL view [717](#)
  - crossprobing to Technology view [719](#)
  - displaying instances [690](#)
  - displaying tooltips in Tcl window [703](#)
  - identifying encrypted IP objects [711](#)
  - opening [683](#)
  - overlapping objects [697](#)
  - properties [699](#)
- Physical Analyst view
  - adding markers [710](#)
  - critical paths [750](#)
  - crossprobing [713](#)
  - displaying net signal flow [696](#)
  - Expand commands [722](#)
  - filtering [721](#)
  - finding objects [704](#)
  - Go to Location command [708](#)
  - selecting objects [696](#)
  - tool tips (Physical Analyst) [702](#)
  - using markers [709](#)
  - zoom selected objects [685](#)
- physical constraints
  - design plan-based physical synthesis [49](#)
  - design-plan based logic synthesis [39](#)
  - translating UCF constraints [258](#)
  - using island timing report [516](#)
  - Xilinx output file [228](#)
- physical constraints (Design Planner - Altera)
  - Altera guidelines [519](#)
- physical constraints (Design Planner - Xilinx)
  - Xilinx guidelines [523](#)
- physical coordinates
  - marking [709](#)
- physical synthesis
  - Altera [60](#)
  - analyzing results [678](#)
  - improve performance (Actel) [763](#)
  - improve performance (Altera) [779](#), [827](#)
  - running place-and-route [849](#)
  - translating UCF constraints [258](#)
  - using design plan file [494](#)
  - with back annotation [353](#)
  - with design plan file [49](#)
  - Xilinx [69](#)
  - Xilinx detail placement [78](#)
  - Xilinx global placement [78](#)
  - Xilinx routing [79](#)
- PICs [788](#)
- pin assignment (Design Planner) [495](#)
  - assigning clock pins [498](#)
  - crossprobing [504](#)
  - temporary assigns [501](#)
- pin assignment tool (Design Planner) [491](#)
- pin assignments
  - converting to constraints [348](#)
- pin assignments (Design Planner)
  - temporary [501](#)
- pin loc constraint files
  - converting to SDC [348](#)
- pin locations
  - specifying (Xilinx) [812](#)
- pin names, displaying [659](#)
- pins
  - expanding logic from [661](#), [722](#)
  - properties [617](#)
- pipelining
  - adding attribute [431](#)
  - critical paths in regions [532](#)
  - definition [429](#)
  - multipliers [429](#)
  - prerequisites [429](#)
  - whole design [430](#)
- place-and-route
  - creating implementation [332](#)

---

- customizing option file [337](#), [340](#)
- placement constraint file [353](#)
- with back annotation [353](#)
- with physical synthesis [849](#)
- place-and-route implementations [332](#)
- PLLs
  - defining clocks [224](#)
- ports
  - false path constraint [235](#)
  - properties [617](#)
- POS interface
  - using [232](#)
- post-synthesis simulation, Xilinx [828](#)
- preferences
  - crossprobing to place-and-route file [623](#)
  - displaying Hierarchy Browser [623](#)
  - displaying labels [624](#)
  - RTL and Technology views [623](#)
  - SCOPE [217](#)
  - sheet size (UI) [624](#)
- preplace.srm file [682](#)
- preserving region resources
  - Design Plan Editor view [509](#)
- primitives
  - pin name display [659](#)
  - pushing into with mouse stroke [629](#)
  - viewing internal hierarchy [655](#)
- primitives (Synplify Premier)
  - breaking up large [543](#)
- probes
  - adding in source code [461](#)
  - definition [461](#)
  - retiming [438](#)
- process-level hierarchy [542](#)
- processors, embedding with EDK [199](#)
- Product of Sums interface. *See* POS interface
- project command
  - archiving projects [315](#)
  - copying projects [323](#)
  - unarchiving projects [320](#)
- project file hierarchy [279](#)
- project files
  - adding files [274](#)
  - adding source files [270](#)
  - batch mode [876](#)

- creating [270](#)
- definition [270](#)
- deleting files from [274](#)
- opening [273](#)
- replacing files in [274](#)
- updating include paths [277](#)
- VHDL file order [273](#)
- VHDL library [272](#)
- projects
  - archiving [315](#)
  - copying [323](#)
  - restoring archives [320](#)
- properties
  - copying and pasting (Physical Analyst) [703](#)
  - displaying with tooltip [617](#)
  - encrypted IP cells (Physical Analyst) [711](#)
  - finding objects with Tcl Find [245](#)
  - reporting for collections [248](#)
  - viewing for individual objects [617](#)
- Push/Pop mode
  - HDL Analyst [628](#)
  - keyboard shortcut [630](#)
  - using [628](#), [630](#)

## Q

- qsf2sdc
  - translating constraints [253](#)
- Quartus
  - batch mode [781](#)
  - integrated flow [779](#)
  - interactive flow [780](#)
  - using instantiated Clearbox netlist files [175](#)
- Quartus II
  - using synthesis results to run [779](#)
- Quartus II Incremental Compilation diagram [858](#)
- Quartus II Incremental Compilation flow [857](#)
- Quartus II Incremental Synthesis running [859](#)
- QUARTUS\_ROOTDIR variable
  - inferring Clearbox megafunctions [167](#)
  - instantiating Clearbox [170](#)
- question mark wildcard, Find command [640](#)

- QuickLogic
  - pad placement [856](#)
- QuickLogic netlist [836](#)
- R**
- RAM inference
  - multi-port RAMs, Altera [388](#)
  - Stratix dual-port [381](#)
- RAM resources
  - viewing in Design Planner [534](#)
- ram\_block primitive
  - Clearbox [165](#)
- RAMs
  - Altera Stratix [379](#)
  - compiling with SYNCORE [119](#)
  - dual-port, Stratix [381](#)
  - initializing [400](#)
  - initializing values (Xilinx) [404](#)
  - mapping LUTRAMs [389](#)
  - multi-port. *See* dual-port RAMs, multi-port RAMs
- RAMs, inferring [372](#)
  - advantages [372](#)
  - Altera EABs and ESBs [767](#)
  - Altera Flex details [386](#)
  - Xilinx block RAMs [391](#)
- ReadyIP
  - encryption-decryption flow [144](#)
- regions
  - assigning critical paths (Island Timing Analyst) [516](#)
  - assigning Xilinx block mults [539](#)
  - assigning Xilinx block RAM [534](#)
  - assigning Xilinx DSP [540](#)
  - creating Xilinx block mult regions [539](#)
  - creating Xilinx blockRAM [535](#)
  - retiming [439](#)
  - Xilinx critical paths [527](#)
- regions (Design Planner)
  - preserving logic and memory resources [509](#)
  - replicating logic manually
    - instances
      - replicating (Design Planner) 515**
- register balancing. *See* retiming
- register constraints, setting [221](#)
- register packing
  - See also* syn\_useioff attribute [807](#)
  - Altera [774](#)
  - Xilinx [807](#)
- registers
  - false path constraint [235](#)
  - INIT value [810](#)
- relative placement. *See* RLOCs
- replication
  - controlling [448](#)
- reports
  - gated clock conversion [472](#)
  - shift registers, Altera [412](#)
- resource sharing [787](#)
  - optimization technique [427](#)
  - overriding option with syn\_sharing [450](#)
  - results example [450](#)
  - using [450](#)
- resynthesis
  - compile points [571](#)
  - forcing with Resynthesize All [571](#)
  - forcing with Update Compile Point Timing Data [571](#)
- retiming
  - effect on attributes and constraints [437](#)
  - example [435](#)
  - overview [433](#)
  - probes [438](#)
  - regions [439](#)
  - report [436](#)
  - simulation behavior [438](#)
- return codes [876](#)
- RLOC\_ORIGINS
  - specifying [821](#)
- RLOCs [819](#), [821](#)
  - specifying with synthesis attribute [821](#)
  - specifying with xc attributes [819](#)
- ROM
  - block RAM mapping (Xilinx) [398](#)
- rom.info file [630](#)
- ROMs
  - compiling with SYNCORE [125](#)
  - inferencing in Altera designs [765](#)
  - pipelining [429](#)
  - viewing data table [630](#)
- route constraint
  - physical synthesis [230](#)

---

- route option
  - Xilinx synthesis 230
- routing
  - Xilinx physical synthesis 79
- RTL view
  - See also* HDL Analyst
  - analyzing clock trees 733
  - crossprobing collection objects 239
  - crossprobing description 645
  - crossprobing from 647
  - crossprobing from Text Editor 649
  - defined 615
  - description 614
  - filtering 658
  - finding objects with Find 637
  - finding objects with Hierarchy Browser 635
  - flattening hierarchy 666
  - highlighting collections 247
  - opening 616
  - selecting/deselecting objects 620
  - sequential shift components 411
  - setting preferences 623
  - state machine implementation 456
  - traversing hierarchy 627

## S

- schematics
  - multisheet. *See* multisheet schematics
  - page size 624
  - selecting/deselecting objects 620
- SCOPE
  - adding attributes 308
  - adding probe insertion attribute 462
  - assigning Xilinx pin locations 813
  - case sensitivity for Verilog designs 245
  - collections compared to Tcl script window 237
  - creating compile-point constraint file 578
  - defining compile points 575
  - drag and drop 215
  - editing operations 216
  - I/O pad type 228
  - keyboard shortcuts 216
  - multicycle paths 234
  - pipelining attribute 431
  - setting compile point constraints 579
  - setting constraints 212

- setting display preferences 217
- specifying RLOCs 819, 821
- state machine attributes 369
- scope of the document 26
- sdc
  - converting from Xilinx ucf 261
- search
  - browsing objects with the Find command 636
  - browsing with the Hierarchy Browser 635
  - finding objects on current sheet 637
  - setting limit for results 639
  - setting scope 638
  - using the Find command in HDL Analyst views 637
- secure core flow
  - synthesis 196
- See also* search
- Select Net Instances command (Physical Analyst) 725
- selecting objects (Physical Analyst) 696
- Send Crossprobes when selecting command 713
- sequential shift components
  - Altshift\_tap 410
  - mapping 410
  - SRL16 primitives 410
  - Verilog 415
  - VHDL 414
- sequential shift components *See* shift registers
- set command
  - collections 249
- set\_option command 291
- sfp file
  - creating 494
  - logic synthesis 39
  - physical synthesis 49
- sheet connectors
  - navigating with 622
- sheet size
  - setting number of objects 624
- shift register lookup table. *See* sequential shift components
- shift registers

- inferring 410
- Shift-F3 key
  - Message Viewer 603
- Show Cell Interior option 655
- Show Context command
  - different from Expand 657
  - using 657
- signal flow (Physical Analyst) 696
  - displaying 696
- Signal Flow command 696
- signal pins (Physical Analyst)
  - displaying 693
- simulation, effect of retiming 438
- single-port RAMs
  - block RAM with registered output, Xilinx 396
- site columns
  - properties (Physical Analyst) 702
- sites (Physical Analyst)
  - properties 702
- slack
  - handling 756
  - setting margins 733
- slice\_primitive command 543
- Slow property 619
- SmartCompile (Xilinx) 862
- SmartGuide 863
- SOPC
  - specifying components as black boxes 193
  - specifying components as white boxes 193
- SOPC Builder
  - components 191
  - importing embedded systems 186
- sopc2syn
  - using 193
- source code
  - adding pipelining attribute 431
  - commenting with synthesis on/off 302
  - crossprobing from Tcl window 652
  - defining FSMs 367
  - fixing errors 87
  - opening automatically to crossprobe 648
  - optimizing 426
  - specifying RLOCs 819, 821
  - when to use for constraints 98
- source files
  - See also* Verilog, VHDL.
  - adding comments 86
  - adding files 270
  - checking 83
  - column editing 86
  - copying examples from PDF 86
  - creating 82
  - crossprobing 649
  - crossprobing to Physical Analyst 716
  - editing 85
  - editing operations 85
  - mixed language 95
  - specifying default encoding style 301
  - specifying top level file for mixed language projects 96
  - specifying top level in Project view 273
  - specifying top-level file 301
  - state machine attributes 369
  - using bookmarks 86
- special characters
  - Tcl collections 244
- specifying levels 668
- SRLs *See* shift registers
- startup block (Xilinx) 800
- state machines
  - See also* FSM Compiler, FSM Explorer, FSM viewer, FSMs.
  - attributes 369
  - descriptions in log file 455
  - implementation 456
  - parameter and 'define comparison 368
- statemachine.info file 674
- Stratix
  - dual-port rams 381
- subsystems
  - EDK cores 203
  - SOPC components 193
- subsystems, Xilinx
  - integrating as a top-level module 204
  - integrating as submodules 205
- subtractors
  - SYNCore 130
- syn\_allow\_retiming
  - using for retiming 434

---

syn\_allowed\_resources  
   compile points 568

syn\_black\_box  
   instantiating LPMs (Altera) 417

syn\_dspstyle attribute  
   inferring wide adders/subtractors 802

syn\_edif\_bit\_format attribute 804

syn\_edif\_scalar\_format attribute 804

syn\_encoding attribute 370

syn\_enum\_encoding directive  
   FSM encoding 371

syn\_force\_pad attribute  
   using 452, 794

syn\_forward\_io\_constraints attribute 105

syn\_hier attribute  
   Altera Quartus II Incremental  
     Compilation flow 860  
   controlling flattening 444  
   preserving hierarchy 444  
   using with compile points 579

syn\_insert\_buffer attribute  
   BUFGMUX 818

syn\_isclock  
   black box clock pins 366

syn\_keep  
   inferring Altera shift registers 411  
   inferring Lattice PICs 789  
   replicating redundant logic 441

syn\_keep attribute  
   preserving nets 440  
   preserving shared registers 440

syn\_keep directive  
   effect on buffering 448

syn\_loc attribute  
   Actel I/O placement for physical  
     synthesis 349

syn\_macro  
   specifying encrypted IP as white box 152  
   white-boxing non-secure cores 197

syn\_maxfan attribute  
   setting fanout limits 446

syn\_noarrayports attribute  
   use with als핀 833

syn\_noprune directive  
   inferring Altera shift registers 411

  preserving instances 440

syn\_pipeline attribute 431

syn\_preserve  
   effect on buffering 448  
   preserving power-on for retiming 435  
   preserving registers with INIT values  
     810

syn\_preserve directive  
   preserving FSMs from optimization 370  
   preserving logic 440

syn\_probe attribute 461  
   inserting probes 461  
   preserving nets 440

syn\_ramstyle attribute  
   glue logic for Altera Stratix RAMs 380  
   multi-port RAM inference 377  
   preventing glue logic (no\_rw\_check) 393

syn\_reference\_clock  
   defining non-clock signal frequencies  
     224

syn\_reference\_clock constraint 101

syn\_replicate attribute  
   using buffering 449

syn\_romstyle attribute  
   defining ROM style 765

syn\_sharing directive  
   overriding default 450

syn\_srlstyle attribute  
   altshift\_tap 410  
   mapping sequential shift components  
     to registers 410  
   setting shift register style 410

syn\_state\_machine directive  
   using with value=0 457

SYN\_TCL\_HOOKS environment variable  
   884

syn\_tco attribute  
   adding in SCOPE 365

syn\_tco directive 364  
   adding black box constraints 363

syn\_tpd attribute  
   adding in SCOPE 365

syn\_tpd directive 364  
   adding black box constraints 363

syn\_tsu attribute  
   adding in SCOPE 365

- syn\_tsu directive [364](#)
    - adding black box constraints [363](#)
  - syn\_use\_carry\_chain attribute
    - using [788](#)
  - syn\_useioff
    - preventing flops from moving during retiming [435](#)
  - syn\_useioff attribute
    - inferring Altera shift registers [411](#)
    - packing registers (Altera) [774](#)
    - packing registers (Xilinx) [807](#)
  - SYN\_XILINX\_GLOBAL\_PLACE\_OPT environment variable [346](#)
  - SYNCore
    - adder ports [136](#)
    - Adder/Subtractor [130](#)
    - adders [130](#)
    - counter compiler [137](#)
    - counters [137](#)
    - FIFO compiler [114](#)
    - RAM compiler [119](#)
    - ROM compiler [125](#)
    - subtractor ports [136](#)
    - subtractors [130](#)
  - synhooks
    - automating message filtering [607](#)
  - synhooks.tcl file [884](#)
  - Synplicity
    - product family [22](#)
  - synplicity.ucf file
    - non-secure cores [198](#)
    - relation to ncf file [228](#)
    - secure cores [198](#)
  - Synplify
    - features [23](#)
    - overview [23](#)
  - Synplify Premier
    - features [23](#)
    - list of design flows [35](#)
    - logic synthesis flows [35](#)
    - overview [23](#)
    - prototyping flow [80](#)
  - Synplify Premier physical constraints
    - using island timing report [516](#)
  - Synplify Pro
    - features [23](#)
    - overview [23](#)
  - prototyping flow [80](#)
  - synplify UNIX command [27](#)
  - synplify.ucf [262](#)
  - synplify.vhd [828](#)
  - synplify\_premier UNIX command [27](#)
  - synplify\_premier\_dp UNIX command [27](#)
  - synplify\_pro UNIX command [27](#)
  - syntax
    - checking source files [84](#)
  - syntax check [84](#)
  - synthesis
    - Xilinx non-secure cores [196](#)
    - Xilinx secure cores [196](#)
  - synthesis check [84](#)
  - synthesis\_on/off
    - using [302](#)
  - SystemDesigner
    - using with Xilinx IP [153](#)
- ## T
- ta file
    - generating [736](#)
  - tcl callbacks
    - customizing key assignments [885](#)
  - Tcl commands
    - batch script [877](#)
    - entering in SCOPE [221](#)
    - running [878](#)
  - Tcl expand command
    - crossprobing objects [239](#)
    - usage tips [246](#)
    - using in SCOPE [238](#)
  - Tcl files [878](#)
    - creating [879](#)
    - for bottom-up synthesis [883](#)
    - guidelines [99](#)
    - naming conventions [99](#)
    - recording from commands [879](#)
    - synhooks.tcl [884](#)
    - using variables [881](#)
    - wildcards [100](#)
  - Tcl find command
    - annotating properties [245](#)
    - case sensitivity [245](#)
    - crossprobing objects [239](#)



---

- database differences [239](#)
- examples of filtering [246](#)
- pattern matching [245](#)
- Tcl window vs SCOPE [237](#)
- usage tips [244](#)
- using in SCOPE [238](#)
- Tcl Script window
  - crossprobing [652](#)
  - message viewer [602](#)
- Tcl script window
  - collections compared to SCOPE [237](#)
- Tcl scripts
  - See Tcl files.
- Tcl window
  - displaying tooltips (Physical Analyst) [703](#)
- Technology view
  - See also HDL Analyst
  - critical paths [733](#)
  - crossprobing [645](#), [647](#)
  - crossprobing collection objects [239](#)
  - crossprobing from source file [649](#)
  - filtering [658](#)
  - finding objects [639](#)
  - finding objects with Find [637](#)
  - finding objects with Hierarchy Browser [635](#)
  - flattening hierarchy [666](#)
  - general description [614](#)
  - highlighting collections [247](#)
  - opening [616](#)
  - selecting/deselecting objects [620](#)
  - setting preferences [623](#)
  - state machine implementation in [456](#)
  - traversing hierarchy [627](#)
- temporary assigns (Design Planner) [501](#)
  - drag and drop [501](#)
  - empty [501](#)
  - return assignment [501](#)
- text editor
  - built-in [85](#)
  - external [90](#)
  - using [85](#)
- Text Editor view
  - crossprobing [647](#)
- Text Editor window
  - colors [88](#)
  - crossprobing [88](#)
- fonts [88](#)
- text files
  - crossprobing [649](#)
- The Synplicity Product Family [22](#)
- third-party vendor tools
  - invoking [838](#)
- through constraints [232](#)
  - AND lists [233](#)
  - OR lists [232](#)
- time stamp, checking on files [275](#)
- time stamps
  - Xilinx partition flow [866](#)
- timing
  - after logic synthesis [682](#)
- timing analysis [730](#)
  - using stand-alone ta [736](#)
- timing analyst
  - modifying constraints using .adc file [737](#)
  - using the stand-alone ta [736](#)
- timing constraints [101](#)
  - define\_compile\_point [569](#)
  - define\_current\_design [569](#)
  - Xilinx output file [228](#)
- timing failures, handling [756](#)
- timing information
  - critical paths [734](#)
- timing optimization [428](#)
- timing report
  - connectivity-based [743](#)
  - viewing [749](#)
  - Island Timing Analyst [743](#)
  - stand-alone (.ta) [736](#)
- timing reports
  - specifying format options [297](#)
- tips
  - memory usage [670](#)
- to constraints
  - specifying [231](#)
- tool tags
  - creating [838](#)
  - definition [838](#)
- tool tips
  - Physical Analyst [702](#)
- tooltips (Physical Analyst)



---

- copying information from [703](#)
- displaying in Tcl window [703](#)
- top level
  - specifying [301](#)
- transparent instances
  - flattening [666](#)
  - lower-level logic on multiple sheets [622](#)

## U

- UCF constraints [255](#)
  - input files [261](#)
  - supported [263](#), [264](#)
  - translating for logic synthesis [255](#)
  - translating for physical synthesis [258](#)
- ucf file
  - translating with ucf2sdc\_old [264](#)
  - using as input for logic design [256](#)
- ucf file. *See also* [synplicity.ucf](#)
- ucf2sdc.log file [261](#)
- ucf2sdc\_old
  - translating Xilinx constraints [264](#)
- UNISIM library
  - simulation [828](#)
- UNISIM library [798](#)
- UNIX commands
  - synplify [27](#)
  - synplify\_premier [27](#)
  - synplify\_premier\_dp [27](#)
  - synplify\_pro [27](#)
- unsupported.ucf [262](#)

## V

- vendor-specific netlists [836](#)
- verification
  - using VIF file [844](#)
- Verification Interface Format (VIF) file.  
*See* [VIF file](#).
- Verilog
  - 'define statements [300](#)
  - Actel ACTgen macros [761](#)
  - adding attributes and directives [307](#)
  - adding probes [461](#)
  - Altera LPM library [423](#)
  - Altera LPM megafunction example [417](#)
  - Altera PLLs [769](#)

- always block hierarchy [331](#)
- black boxes [358](#)
- black boxes, instantiating [358](#)
- case sensitivity for Tcl Find command [245](#)
- checking source files [83](#)
- choosing a compiler [299](#)
- clock DLLs [823](#)
- creating source files [82](#)
- crossprobing from HDL Analyst view [647](#)
- crossprobing to Physical Analyst [716](#)
- defining FSMs [368](#)
- defining state machines with parameter and 'define [368](#)
- editing operations [85](#)
- extracting parameters [299](#)
- include paths, updating [277](#)
- inferring DesignWare functions [108](#)
- initializing RAMs [400](#)
- instantiating LPMs as black boxes (Altera) [417](#)
- macro library (Xilinx) [798](#)
- mixed language files [95](#)
- RAM structures for inference [373](#)
- RLOCs [820](#)
- sequential shift components [415](#)
- specifying compiler directives [300](#)
- specifying top-level module [301](#)
- structural, for instantiated Clearbox [173](#)

- Verilog 2001
  - setting global option from the Project view [299](#)
  - setting option per file [299](#)

- Verilog macro libraries
  - Actel [760](#)
  - Lattice [785](#)

- Verilog model (.vmd) [570](#)

## VHDL

- Actel ACTgen macros [761](#)
- adding attributes and directives [305](#)
- adding probes [461](#)
- Altera LPM megafunction example [419](#)
- Altera PLLs [769](#)
- black boxes [360](#)
- black boxes, instantiating [360](#)
- case sensitivity for Tcl Find comand [245](#)
- checking source file [83](#)
- clock DLLs [823](#)

- constants 302
  - creating source files 82
  - crossprobing from HDL Analyst view 647
  - crossprobing to Physical Analyst 716
  - defining FSMs 369
  - DesignWare component instantiations 109
  - editing operations 85
  - extracting generics 302
  - initializing RAMs with variable declarations 403
  - initializing with signal declarations 401
  - instantiating LPMs as black boxes (Altera) 417
  - LPM instantiation example 421
  - macro libraries, Actel 760
  - macro library (Xilinx) 798
  - mixed language files 95
  - prepared components method of instantiation 422
  - process hierarchy 331
  - RAM structures for inference 373
  - RLOCs 820
  - sequential shift components 414
  - specifying top-level entity 301
  - structural, for instantiated Clearbox 173
  - VHDL files
    - adding library 272
    - adding third-party package library 272
    - order in project file 273
    - ordering automatically 273
  - VHDL macro libraries
    - Lattice 786
  - vi text editor 90
  - VIF file
    - using 844
  - vif2conformal.tcl script 847
  - Virtex
    - block RAM. *See also* block RAM.
    - clock buffers 822
    - I/O buffers 825
    - netlist 837
    - PCI core 804
  - virtual clock, setting 220
  - vqm
    - inferred Clearbox 167
    - instantiated Clearbox 170
    - instantiated Clearbox with netlist 175
- ## W
- warning messages
    - definition 85
  - warnings
    - feedback muxes 428
    - filtering 603
    - handling 609
    - sorting 603
  - white boxes
    - defined for EDK flow 206
    - EDK cores 206
    - encrypted Xilinx cores 207
    - SOPC components 193
    - using `syn_macro` on non-secure cores 197
  - wide adders/subtractors
    - example 803
    - inferring 801
    - prerequisites for inference 802
  - wildcards
    - effect of search scope 640
    - Find command (Tcl) 245
    - message filter 605
  - wildcards (Find)
    - examples 642
    - how they work 640
  - wildcards (Physical Analyst)
    - in Find command 706
  - workspaces
    - creating 287
    - using 288
  - write modes, Virtex-II 394
- ## X
- xc\_clockbuftype attribute
    - specifying 822
  - xc\_fast attribute
    - for critical paths 797
  - xc\_loc attribute
    - assigning locations in SCOPE 813
  - xc\_map attribute
    - relative location 819
  - xc\_padtype attribute

---

- specifying I/Os [825](#)
- xc\_rloc attribute
  - specifying relative location [820](#)
- xc\_uset attribute
  - grouping instances for relative placement [820](#)
  - using to group instances [820](#)
- xcf files
  - translating to sdc [261](#)
- XFLOW for cores [210](#)
- xflow script [340](#)
- Xilinx
  - block RAMs [391](#)
  - clock buffers [822](#)
  - converting PAD files [348](#)
  - converting pin assignments to SDC [348](#)
  - CoreGen [804](#)
  - defining DCMs and DLLs [225](#)
  - design guidelines [797](#)
  - detail placement for physical synthesis [78](#)
  - EDK cores [199](#)
  - EDK standalone hardware flow [209](#)
  - EDK. *See* EDK
  - EDK-ISE hardware flow [208](#)
  - encrypted EDK cores [207](#)
  - forward-annotation [106](#)
  - generating EDK cores with ISE [208](#)
  - global placement for physical synthesis [78](#)
  - GSR [800](#)
  - I/O buffers [825](#)
  - I/O insertion, manual [817](#)
  - I/O locations [812](#)
  - I/O pad type [228](#)
  - including cores for synthesis [196](#)
  - incremental flow [862](#)
  - INIT property [405](#)
  - INIT property, VHDL [407](#)
  - IP cores [195](#), [804](#)
  - macro libraries [798](#)
  - macros [798](#)
  - netlist [837](#)
  - non-secure core flow [196](#)
  - packing registers [807](#)
  - partition flow [863](#)
  - physical synthesis flows [69](#)
  - place-and-route option file [340](#)
  - post-synthesis simulation [828](#)

- reoptimizing EDIF [826](#)
- routing for physical synthesis [79](#)
- secure core flow [196](#)
- shift registers [410](#)
- specifying pin location [812](#)
- startup blocks [800](#)
- synthesis constraint files [228](#)
- tips for optimizing [797](#)
- Virtex-II write modes [394](#)
- Xilinx block multipliers (Synplify Premier) [539](#)
- Xilinx DSP blocks (Synplify Premier) [540](#)
- xtclsh flow [340](#)

## Z

- zippering
  - guidelines [550](#), [551](#)
  - partitioning (Synplify Premier) [550](#)
- zoom selected objects (Physical Analyst) [685](#)

