Synplicity®

Simply Better Results

Synplicity® FPGA Synthesis

Synplify®, Synplify Pro®, Synplify® Premier, and
Synplify® Premier with Design Planner

User Guide
December 2005

Synplicity, Inc.
600 West California Avenue
Sunnyvale, CA 94086
(U.S.) +1 408 215-6000 direct
(U.S.) +1 408 222-0268 fax
www.synplicity.com

Preface

Disclaimer of Warranty

Synplicity, 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

While every precaution has been taken in the preparation of this book, Synplicity, Inc. assumes no responsibility for errors or omissions. This publication and the features described herein are subject to change without notice.

Trademarks

Synplicity, the Synplicity “S” logo, Amplify, Amplify ASIC, Amplify FPGA, Behavior Extracting Synthesis Technology, Certify, Embedded Synthesis, Fortify, HDL Analyst, PowerTime, RealPower, SCOPE, Simply Better Results, Simply Better Synthesis, Syndicated, Synplify, Synplify ASIC, Synplify Lite, Synplify Pro, and Synthesis Constraint Optimization Environment are registered trademarks of Synplicity Inc. BEST, DST, Direct Synthesis Technology, Identify, IICE, MultiPoint, Partition-Driven Synthesis, Physical Analyst, Physical Optimizer, PowerPlanner, PowerRoute, Synplify IP, TOPS, and Total Optimization Physical Synthesis are trademarks of Synplicity, Inc.

Verilog is a registered trademark of Cadence Design Systems, Inc. IBM and PC are registered trademarks of International Business Machines Corporation. Microsoft is a registered trademark of Microsoft Corporation. Sun, SPARC, Solaris, and SunOS are trademarks of Sun Microsystems, Inc. UNIX is a registered trademark of X/Open Corporation.

All other product names mentioned herein are the trademarks or registered trademarks of their respective owners.

Restricted Rights Legend

Printed in the U.S.A
December 2005

Synplicity Software License Agreement

Important! READ CAREFULLY BEFORE PROCEEDING

BY INDICATING YOUR ACCEPTANCE OF THE TERMS OF THIS AGREEMENT, YOU (“LICENSEE”) ARE REPRESENTING THAT YOU HAVE THE RIGHT AND AUTHORITY TO LEGALLY BIND YOURSELF OR YOUR COMPANY, AS APPLICABLE, AND CONSENTING TO BE LEGALLY BOUND BY ALL OF THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE TO THE LOCATION OF PURCHASE FOR A REFUND. This is a legal agreement governing use of the software program provided by Synplicity, Inc. (“Synplicity”) to you (the “SOFTWARE”). The term “SOFTWARE” also includes related documentation (whether in print or electronic form), any authorization keys, authorization codes, and license files, and any updates or upgrades of the SOFTWARE provided by Synplicity, but does not include certain “open source” software licensed by third party licensors and made available to you by Synplicity under the terms of such third party licensor’s license (such as software licensed under the General Public License (GPL)) (“Third Party Software”). If Licensee is a participant in the University Program or has been granted an Evaluation License or Subscription License, then some of the following terms and conditions may not apply (refer to the sections entitled, respectively, **Evaluation License** and **Subscription License**, below).

License. Synplicity grants to Licensee a non-exclusive right to install the SOFTWARE and to use or authorize use of the SOFTWARE by up to the number of nodes for which Licensee has a license and for which Licensee has the security key(s) or authorization code(s) provided by Synplicity or its agents for the purpose of creating and modifying Designs (as defined below). If Licensee has obtained the SOFTWARE under a node-locked license, then a “node” refers to a specific machine, and the SOFTWARE may be installed only on the number of “nodes” or machines authorized, must be used only on the machine(s) on which it is installed, and may be accessed only by users who are physically present at that node or machine. A node-locked license may only be used by one user at a time running one instance of the software at a time. If Licensee has obtained the SOFTWARE under a “floating” license, then a “node” refers to a concurrent user or session, and the SOFTWARE may be used concurrently by up to the number of users or sessions indicated. All SOFTWARE must be used within the country for which the systems were licensed and at Licensee's Site (contained within a one kilometer radius); however, if Licensee has a floating license then remote use is permitted by employees who work at the site but are temporarily telecommuting to that same site from less than 50 miles away (for example, an employee who works at a home office on occasion), but the maximum number of concurrent sessions or nodes still applies. In addition, Synplicity grants to Licensee a non-exclusive license to copy and distribute internally the documentation portion of the SOFTWARE in support of its license to use the program portion of the SOFTWARE. For purposes of this Agreement the “Licensee’s Site” means the location of the server on which the SOFTWARE resides, or when a server is not required, the location of the client computer for which the license was issued.

Evaluation License. If Licensee has obtained the SOFTWARE pursuant to an evaluation license, then, in addition to all other terms and conditions herein, the following restrictions apply: (a) the license to the SOFTWARE terminates after 20 days (unless otherwise agreed to in writing by Synplicity); and (b) Licensee may use the SOFTWARE only for the sole purpose of internal testing and evaluation to determine whether Licensee wishes to license the SOFTWARE on a commercial basis. Licensee shall not use the SOFTWARE to design any integrated circuits for production or pre-production purposes or any other commercial use including, but not limited to, for the benefit of Licensee's customers. If Licensee breaches any of the foregoing restrictions, then Licensee shall pay to Synplicity a license fee equal to Synplicity's perpetual list price plus maintenance for the commercial version of the SOFTWARE.

Subscription (Time-Based) License. If Licensee has obtained a Subscription License to the SOFTWARE, the, in addition to all other terms and conditions herein, the following restrictions apply: (a) Licensee is authorized to use the SOFTWARE only for a limited time (which time is indicated on the quotation or in the purchase confirmation documents); (b) Licensee's right to use the SOFTWARE terminates on the date the subscription term expires as set forth in the quotation or the purchase confirmation documents, unless Licensee has renewed the license by paying the applicable fees.

Project Based License. If Licensee has obtained a Project-Based License to the SOFTWARE, in addition to all other terms and conditions herein, the terms of Exhibit A will apply.

Copy Restrictions. This SOFTWARE is protected by United States copyright laws and international treaty provisions and Licensee may copy the SOFTWARE only as follows: (i) to directly support authorized use under the license, and (ii) in order to make a copy of the SOFTWARE for backup purposes. Copies must include all copyright and trademark notices.

Use Restrictions. This SOFTWARE is licensed to Licensee for internal use only. Licensee shall not (and shall not allow any third party to): (i) decompile, disassemble, reverse engineer or attempt to reconstruct, identify or discover any source code, underlying ideas, underlying user interface techniques or algorithms of the SOFTWARE by any means whatever, or disclose any of the foregoing; (ii) provide, lease, lend, or use the SOFTWARE for timesharing or service bureau purposes, on an application service provider basis, or otherwise circumvent the internal use restrictions; (iii) modify, incorporate into or with other software, or create a derivative work of any part of the SOFTWARE; (iv) disclose the results of any benchmarking of the SOFTWARE, or use such results for its own competing software development activities, without the prior written permission of Synplicity; or (v) attempt to circumvent any user limits, maximum gate count limits or other license, timing or use restrictions that are built into the SOFTWARE.

Transfer Restrictions/No Assignment. The SOFTWARE may only be used under this license at the designated locations and designated equipment as set forth in the license grant above, and may not be moved to other locations or equipment or otherwise transferred without the prior written consent of Synplicity. Any permitted transfer of the SOFTWARE will require that Licensee executes a "Software Authorization Transfer Agreement" provided by Synplicity. Further, Licensee shall not sublicense, or assign this Agreement or any of the rights or licenses granted under this Agreement, without the prior written consent of Synplicity.

Security. Licensee agrees to take all appropriate measures to safeguard the SOFTWARE and prevent unauthorized access or use thereof. Suggested ways to accomplish this include: (i) implementation of firewalls and other security applications, (ii) use of FLEXlm options file that restricts access to the SOFTWARE to identified users; (iii) maintaining and storing license information in paper format only; (iv) changing TCP port numbers every three (3) months; and (v) communicating to all authorized users that use of the SOFTWARE is subject to the restrictions set forth in this Agreement.

Ownership of the SOFTWARE. Synplicity retains all right, title, and interest in the SOFTWARE (including all copies), and all worldwide intellectual property rights therein. Synplicity reserves all rights not expressly granted to Licensee. This license is not a sale of the original SOFTWARE or of any copy.

Ownership of Design Techniques. “Design” means the representation of an electronic circuit or device(s), derived or created by Licensee through the use of the SOFTWARE in its various formats, including, but not limited to, equations, truth tables, schematic diagrams, textual descriptions, hardware description languages, and netlists. “Design Techniques” means the data, circuit and logic elements, libraries, algorithms, search strategies, rule bases, techniques and technical information incorporated in the SOFTWARE and employed in the process of creating Designs. Synplicity retains all right, title and interest in and to Design Techniques incorporated in the SOFTWARE, including all intellectual property rights embodied therein, provided that to the extent any Design Techniques are included as part of or embedded within Licensee’s Designs, Synplicity grants Licensee a personal, nonexclusive, nontransferable license to reproduce the Design Techniques and distribute such Design Techniques solely as incorporated into Licensee’s Designs and not on a standalone basis. Additionally, Licensee acknowledges that Synplicity has an unrestricted, royalty-free right to incorporate any Design Techniques disclosed by Licensee into its software, documentation and other products, and to sublicense third parties to use those incorporated design techniques.

Protection of Confidential Information. “Confidential Information” means (i) the SOFTWARE, in object and source code form, and any related technology, idea, algorithm or information contained therein, including without limitation Design Techniques, and any trade secrets related to any of the foregoing; (ii) either party's product plans, Designs, costs, prices and names; non-published financial information; marketing plans; business opportunities; personnel; research; development or know-how; (iii) any information designated by the disclosing party as confidential in writing or, if disclosed orally, designated as confidential at the time of disclosure and reduced to writing and designated as confidential in writing within thirty (30) days; and (iv) the terms and conditions of this Agreement; provided, however that “Confidential Information” will not include information that: (a) is or becomes generally known or available by publication, commercial use or otherwise through no fault of the receiving party; (b) is known and has been reduced to tangible form by the receiving party at the time of disclosure and is not subject to restriction; (c) is independently developed by the receiving party without use of the disclosing party's Confidential Information; (d) is lawfully obtained from a third party who has the right to make such disclosure; and (e) is released for publication by the disclosing party in writing.

Each party will protect the other's Confidential Information from unauthorized dissemination and use with the same degree of care that each such party uses to protect its own like information. Neither party will use the other's Confidential Information for purposes other than those necessary to directly further the purposes of this Agreement. Neither party will disclose to third parties the other's Confidential Information without the prior written consent of the other party.

Third Party Software. Licensee understands and agrees that, although provided to Licensee by Synplicity, Licensee’s use of each component library or module comprising the Third Party Software shall be governed by the relevant terms and conditions of the third party’s license agreements.

Termination. Synplicity may terminate this Agreement immediately if Licensee breaches any provision, including without limitation, failure by Licensee to implement adequate security measures as set forth above. Upon notice of termination by Synplicity, all rights granted to Licensee under this Agreement will immediately terminate, and Licensee shall cease using the SOFTWARE and return or destroy all copies (and partial copies) of the SOFTWARE and documentation.

Limited Warranty and Disclaimer. Synplicity warrants that the program portion of the SOFTWARE will perform substantially in accordance with the accompanying documentation for a period of 90 days from the date of receipt. Synplicity's entire liability and Licensee's exclusive remedy for a breach of the preceding limited warranty shall be, at Synplicity's option, either (a) return of the license fee, or (b) providing a fix, patch, work-around, or replacement of the SOFTWARE. In either case, Licensee must return the SOFTWARE to Synplicity with a copy of the purchase receipt or similar document. Replacements are warranted for the remainder of the original warranty period or 30 days, whichever is longer. Some states/jurisdictions do not allow limitations, so the above limitation may not apply. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES OR CONDITIONS, EITHER EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, ARE MADE BY SYNPLICITY OR ITS LICENSORS WITH RESPECT TO THE SOFTWARE AND THE ACCOMPANYING DOCUMENTATION, AND SYNPLICITY EXPRESSLY DISCLAIMS ALL WARRANTIES AND CONDITIONS NOT EXPRESSLY STATED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE. SYNPLICITY AND ITS LICENSORS DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS, BE UNINTERRUPTED OR ERROR FREE, OR THAT ALL DEFECTS IN THE PROGRAM WILL BE CORRECTED. Licensee assumes the entire risk as to the results and performance of the SOFTWARE. Some states/jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply.

Limitation of Liability. IN NO EVENT SHALL SYNPLICITY OR ITS LICENSORS OR THEIR AGENTS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTIONS, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SYNPLICITY AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. FURTHER, IN NO EVENT SHALL SYNPLICITY'S LICENSORS BE LIABLE FOR ANY DIRECT DAMAGES ARISING OUT OF LICENSEE'S USE OF THE SOFTWARE. IN NO EVENT WILL SYNPLICITY OR ITS LICENSORS BE LIABLE TO LICENSEE FOR DAMAGES IN AN AMOUNT GREATER THAN THE FEES PAID FOR THE USE OF THE SOFTWARE. Some states/jurisdictions do not allow the limitation or exclusion of incidental or consequential damages, so the above limitations or exclusions may not apply.

Intellectual Property Right Infringement. Synplicity will defend or, at its option, settle any claim or action brought against Licensee to the extent it is based on a third party claim that the SOFTWARE as used within the scope of this Agreement infringes or violates any US patent, copyright, trade secret or trademark of any third party, and Synplicity will indemnify and hold Licensee harmless from and against any damages, costs and fees reasonably incurred that are attributable to such claim or action; provided that Licensee provides Synplicity with (i) prompt written notification of the claim or action; (ii) sole control and authority over the defense or settlement thereof (including all negotiations); and (iii) at Synplicity's expense, all available information, assistance and authority to settle and/or defend any such claim or action. Synplicity's obligations under this subsection do not apply to the extent that (i) such claim or action would have been avoided but for modifications of the SOFTWARE, or portions thereof, other than modifications made by Synplicity after delivery to Licensee; (ii) such claim or action would have been avoided but for the combination or use of the SOFTWARE, or portions thereof, with other products, processes or materials not supplied or specified in writing by Synplicity; (iii) Licensee continues allegedly infringing activity after being notified thereof or after being informed of modifications that would have avoided the alleged infringement; or (iv) Licensee's use of the

SOFTWARE is not strictly in accordance with the terms of this Agreement. Licensee will be liable for all damages, costs, expenses, settlements and attorneys' fees related to any claim of infringement arising as a result of (i)-(iv) above.

If the SOFTWARE becomes or, in the reasonable opinion of Synplicity is likely to become, the subject of an infringement claim or action, Synplicity may, at Synplicity's option and at no charge to Licensee, (a) obtain a license so Licensee may continue use of the SOFTWARE; (b) modify the SOFTWARE to avoid the infringement; (c) replace the SOFTARE with a compatible, functionally equivalent, and non-infringing product, or (d) refund to Licensee the amount paid for the SOFTWARE, as depreciated on a straight-line 5-year basis, or such other shorter period applicable to Subscription Licenses.

THE FOREGOIN PROVISIONS OF THIS SECTION STATE THE ENTIRE AND SOLE LIABILITY AND OBLIGATIONS OF SYNPLICTY, AND THE EXCLUSIVE REMEDY OF LICENSEE, WITH RESPECT TO ANY ACTUAL OR ALLEGED INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS BY THE SOFTWARE (INCLUDING DESIGN TECHNIQUES) AND DOCUMENTATION.

Export. Licensee warrants that it is not prohibited from receiving the SOFTWARE under U.S. export laws; that it is not a national of a country subject to U.S. trade sanctions; that it will not use the SOFTWARE in a location that is the subject of U.S. trade sanctions that would cover the SOFTWARE; and that to its knowledge it is not on the U.S. Department of Commerce's table of deny orders or otherwise prohibited from obtaining goods of this sort from the United States.

Miscellaneous. This Agreement is the entire agreement between Licensee and Synplicity with respect to the license to the SOFTWARE, and supersedes any previous oral or written communications or documents (including, if you are obtaining an update, any agreement that may have been included with the initial version of the Software). This Agreement is governed by the laws of the State of California, USA excluding its conflicts of laws principals. This Agreement will not be governed by the U. N. Convention on Contracts for the International Sale of Goods and will not be governed by any statute based on or derived from the Uniform Computer Information Transactions Act (UCITA). If any provision, or portion thereof, of this Agreement is found to be invalid or unenforceable, it will be enforced to the extent permissible and the remainder of this Agreement will remain in full force and effect. Failure to prosecute a party's rights with respect to a default hereunder will not constitute a waiver of the right to enforce rights with respect to the same or any other breach.

Government Users. If the SOFTWARE is licensed to the United States government or any agency thereof, then the SOFTWARE and any accompanying documentation will be deemed to be "commercial computer software" and "commercial computer software documentation", respectively, pursuant to DFAR Section 227.7202 and FAR Section 12.212, as applicable. Any use, reproduction, release, performance, display or disclosure of the SOFTWARE and accompanying documentation by the U.S. Government will be governed solely by the terms of this Agreement and are prohibited except to the extent expressly permitted by the terms of this Agreement.

June 2005

Contents

Chapter 1: Introduction

Chapter 2: Project Setup

Chapter 3: Constraints, Attributes, and Options

Chapter 4: Result Analysis

Chapter 5: Physical Analyst

Chapter 6: Design Optimization

Chapter 7: Design Planning and Optimizations

Chapter 8: Vendor-Specific Optimizations

Chapter 9: Design Planning for Vendors

Chapter 10: Design Flows and Process Optimization

Chapter 11: Synplify Premier Design Flow

CHAPTER 1

Introduction

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

- The FPGA Synthesis Tools, on page 1-2
- The Generic FPGA Design Flow, on page 1-8
- Audience, on page 1-11
- Scope of the Document, on page 1-11
- Starting the Software, on page 1-12
- User Interface Overview, on page 1-14
- Design Flows, on page 1-16

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

The FPGA Synthesis Tools

This section briefly discusses the following topics:

- About the Synplify and Synplify Pro Software, next
- About the Synplify Premier Software, on page 1-3
- Synplicity FPGA Tool Features, on page 1-4
- Synplicity Product Family, on page 1-8

About the 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), developed by Synplicity® of Sunnyvale, California. 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. For a comparison of the features, see *Synplicity FPGA Tool Features*, on page 1-4. The software has the following built-in features:

- The HDL Analyst® tool, a graphical interface for analysis and crossprobing. This feature is an option with the Synplify software, but is standard with Synplify Pro.
- The Text Editor window, with a language-sensitive editor for writing and editing HDL code.
- The SCOPE® (Synthesis Constraint Optimization Environment®) interface, which uses a spreadsheet-like format to manage the timing constraints and attributes in the design.
- A symbolic FSM Compiler, which performs advanced state machine optimizations.

- The Timing Analyst window, which allows you to generate timing schematics and reports for specified paths for point-to-point timing analysis. This feature is not available in Synplify.
- The FSM Explorer, which tries different state machine optimizations before picking the best implementation. This feature is not available in Synplify.
- The FSM viewer, which lets you view the transitions in detail. This feature is not available in Synplify.
- A command line interface, from which you can run TCL scripts. This feature is not available in Synplify.

About the 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 the following flows:

- Graph-based physical synthesis—a fully automated flow for incremental performance improvement producing a design with detailed placement. Graph-based physical synthesis is currently available for Virtex-II Pro, Virtex-4, and Spartan-3.
- Graph-based physical synthesis with a design plan—a graph-based physical synthesis flow that is guided by a design plan. It also produces a design with detailed placement. Graph-based physical synthesis with a design plan is currently available for Virtex-II Pro, Virtex-4, and Spartan-3. The Design Planner is a separately licensed option to the Synplify Premier product.
- Design-plan based physical synthesis—an interactive flow that lets you perform physical synthesis optimization using the Design Planner. Design-plan based physical synthesis produces a design with coarse placement. Design-plan based physical synthesis is available for Altera Cyclone, Cyclone-II, Stratix, Stratix-GX, and Stratix-II and Xilinx Virtex, Virtex-II, and Virtex-E. The Design Planner is a separately licensed option to the Synplify Premier product.

- Prototyping— The Synplify Premier product supports a complete design and debugging environment featuring the Identify product along with automated HDL code translation for Synopsys® DesignWare® components. Using the Synplify Premier prototyping solution, you can quickly and automatically bring ASIC HDL code into high-capacity FPGAs, and then debug it with real-time feedback from the FPGA. Because the feedback from the debugger goes directly into the RTL code, the integration of logic synthesis lets you correct the code and retest it in the FPGA in record time.

In addition to the features described in *About the Synplify and Synplify Pro Software,* on page 1-2, the Synplify Premier tool features the following:

- Expanded HDL Analyst capabilities including support for physical regions.
- Design Planner - This licensed option for Synplify Premier lets you run interactive physical synthesis using a design plan. You assign physical constraints in Design Planner by interactively dragging and dropping RTL objects into regions of the device. This information along with the normal timing constraints, enables the Synplify Premier tool to estimate timing more accurately and use the estimates for additional optimizations. This produces a more highly optimized circuit in fewer iterations.
- Physical Optimizations - Physical optimization algorithms use physical design characteristics like placement and interconnect delay to affect the actual topology of the circuit. Design plans are used to control the optimizations. The Synplify Premier tool also helps reduce the time required to perform placement and routing, which is especially significant as FPGA designs continue towards the trend of one million plus gates.
- Physical Analysis Tools - These tools include the Physical Analyst and the Island Timing Analyst.

Synplicity FPGA Tool Features

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

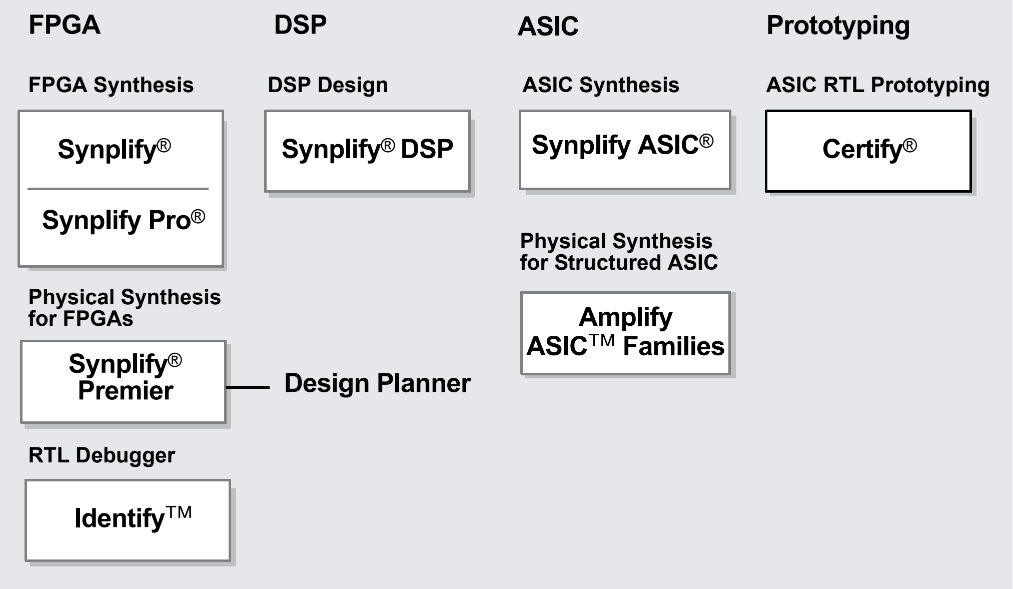
	Synplify	Synplify Pro	Synplify Premier	Synplify Premier with Design Planner
Performance				
Behavior Extracting Synthesis Technology® (BEST™)	x	x	x	x
Clearbox Support (selected Altera architectures)		x	x	x
Coregen Support (selected Xilinx architectures)		x	x	x
FSM Compiler	x	x	x	x
FSM Explorer		x	x	x
Gated Clock Conversion		x	x	x
Register Pipelining into Multipliers/ROMs		x	x	x
Register Retiming		x	x	x
Code Analysis				
Timing Analyzer – Point-to-point Report		x	x	x
HDL Analyst® Solution — Graphical Design Analysis	Option	x	x	x
FSM Viewer – View State Transitions		x	x	x
Crossprobing – Cross-tool Analysis		x	x	x
Probe Point Creation – Easy Insertion of Debug Pins		x	x	x
Team Design/Design Environment				
SCOPE® Spreadsheet	x	x	x	x
Advanced Project Management — Workspaces		x	x	x

	Synplify	**Synplify Pro**	**Synplify Premier**	**Synplify Premier with Design Planner**
Multiple Implementations		x	x	x
Log Watch Window — Fast Implementation Comparison		x	x	x
Tcl Entry and Viewing Window		x	x	x
Text Editor View	x	x	x	x
Mixed Language Design – VHDL and Verilog		x	x	x
Modular Flow Support for Xilinx Designs		x	x	x
MultiPoint™ Flow		x	x	x
Batch Mode (Floating licenses only)	x	x	x	x
Verplex Formal Verification Flow		x	x (Physical Synthesis Disbled)	
Physical Design				
Design Plan File				x
Drag-and-Drop Logic from HDL Analyst into Device Regions				x
Area Estimation and Region Capacity				x
Pin Assignments				x
Physical Synthesis Optimizations •Bit Slicing •Zippering •Tunneling and other boundary optimizations •Replication				x

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier with Design Planner
Graph-based Physical Synthesis			x	x
Island Timing Analyst			x	x
Physical Analyst			x	x
Place-and-Route Implementation Run From Project View		x	x	x
Prototyping				
RTL debugging with Identify			x	
Automatic translation of Synopsys® DesignWare® components			x	

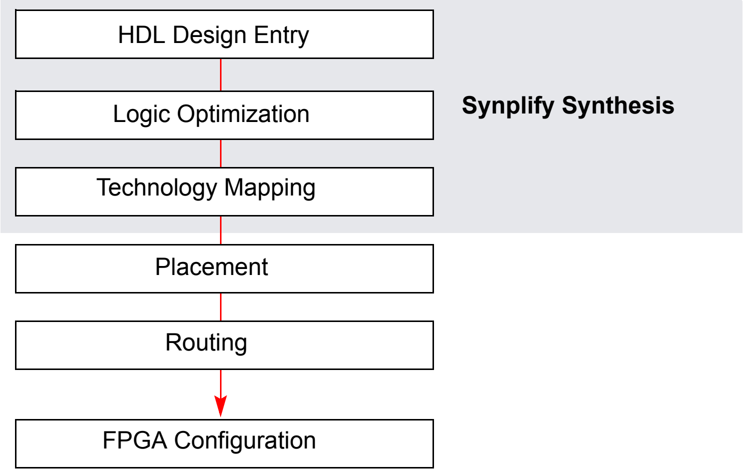
Synplicity Product Family

The Synplicity products are based on core synthesis technology, and share a common look and feel. The Fortify tools offer solutions to manage the power network in your design. The following figure shows the Synplicity products.



The Generic FPGA Design Flow

The following figure contains a generic design flow showing the typical steps a designer follows when implementing an FPGA. The shaded box shows the steps you can accomplish with Synplify synthesis. This generic design flow complements the specific design flow used for the tutorial.



The following sections describe each step more fully.

HDL Design Entry

The starting point for FPGA design is to specify the logic of the FPGA circuit to be implemented. You can do this by drawing a schematic, writing an HDL description, or specifying Boolean expressions.

For the Synplify flow, design entry is the step where you generate the input for the tool. The input must be Verilog or VHDL descriptions. The software provides you with an environment where you can write or edit HDL descriptions.

Logic Optimization (Compilation)

This is the first stage of synthesis, in which the software restructures the original network into a set of combinational functions. In the Synplify flow, the combinational functions are represented as a Boolean network. At this point in the design process, you modify the initial logic design to optimize the area or speed of the final circuit, or both. The optimization is calculated from the netlist and is independent of the target technology. It includes operations like redundancy removal and common subexpression elimination.

Technology Mapping

Technology mapping is the second phase of optimization, in which the logic is optimized to a specific technology. During this phase, the compiled design is transformed into a circuit of optimized FPGA logic blocks. Depending on your design priorities, you might want to focus on area optimization (minimizing the total number of blocks), delay optimization (minimizing the number of logic block stages in time-critical paths), or both.

The Synplify tool uses architecture-specific mapping techniques to map the logic design. It has built-in tools to analyze critical paths, crossprobe, and check the RTL view. The software generates netlists in formats appropriate for the place-and-route tools that follow.

Placement

Placement is the first step of the physical design process. During placement, the logic blocks are placed in an FPGA array. At this point, considerations like the total interconnect length become important.

This is the point at which the Synplify software hands off control of the design to another tool. However if you have the Synplify Premier, you can use the results from an initial placement pass to further optimize your logic design.

Routing

Routing is the final step of the physical design process. At this stage, use the place-and-route tool to connect the placed logic blocks by assigning wire segments and choosing programmable switches.

FPGA Configuration

In this design phase, you configure the final FPGA chip and implement it.

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

Scope of the Document

This user guide is part of a document set that 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 Commands* chapter in the *Synplicity FPGA Synthesis Reference Manual*.
- The user guide contains task-based information. For a breakdown of how information is organized, see *Getting Help*, on page 1-13.

Starting the Software

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:

- Getting Started, on page 1-12
- Getting Help, on page 1-13

Getting Started

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

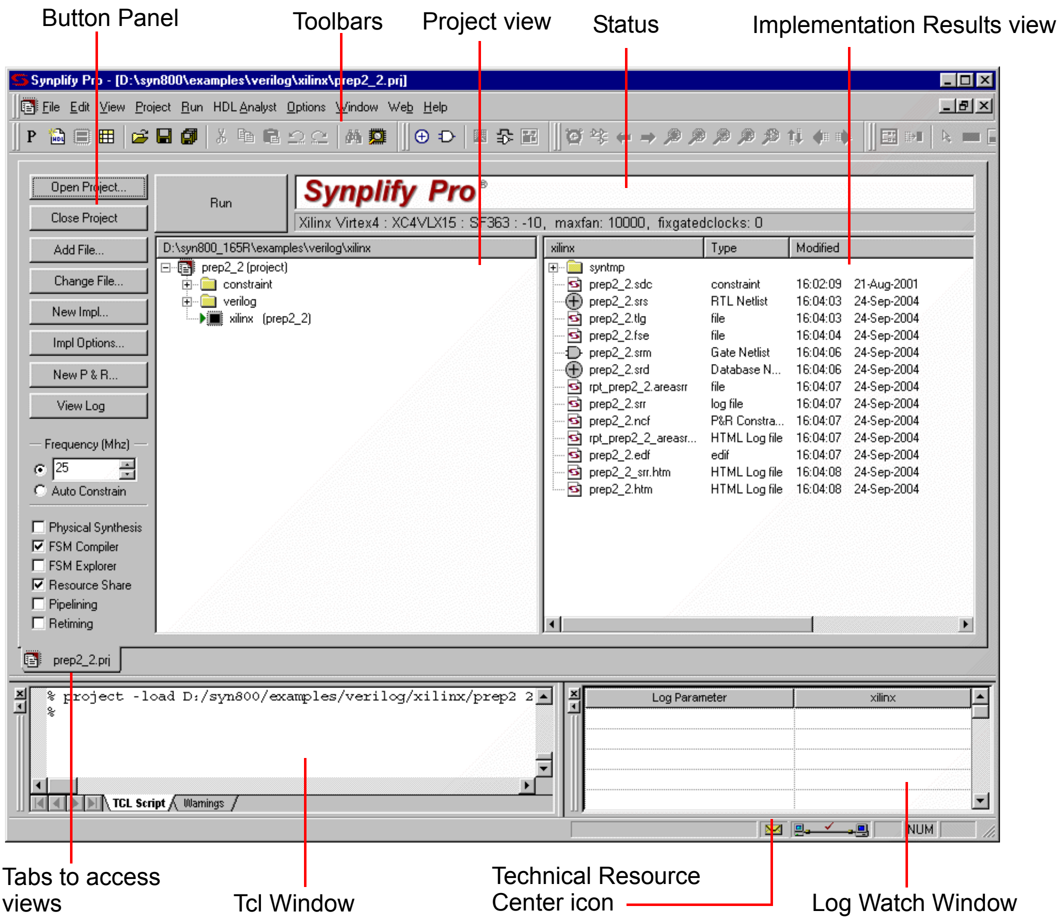
Before you call Synplicity Support, look through the documented information. You can access the information online from the Help menu, or refer to the PDF version. The following table shows you how the information is organized.

For help with...	Refer to the...
Using software features	*Synplicity FPGA Synthesis User Guide*
How to...	*Synplicity FPGA Synthesis User Guide*, application notes on the Synplicity support web site
Flow information	*Synplicity FPGA Synthesis User Guide*, application notes on the Synplicity support web site
Error messages	Online help (select Help->Error Messages)
Licensing	License configuration information for your platform
Attributes and directives	*Synplicity FPGA Synthesis Reference Manual*
Synthesis features	*Synplicity FPGA Synthesis Reference Manual*
Language and syntax	*Synplicity FPGA Synthesis Reference Manual*
Tcl syntax	Online help (select Help->Tcl Help)
Tcl synthesis commands	*Synplicity FPGA Synthesis Reference Manual*
Product updates	*Synplicity FPGA Synthesis Reference Manual* (Web menu commands)

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 the *User Interface Commands* chapter of the *Synplicity 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



Xilinx Virtex4 : XC4VLX15 : SF363 : -10, maxfan: 10000, fixgatedclocks: 0

Frequency (Mhz)
● 25
○ Auto Constrain

□ Physical Synthesis
☑ FSM Compiler
□ FSM Explorer
☑ Resource Share
□ Pipelining
□ Retiming

D:\syn800_165R\examples\verilog\xilinx

- prep2_2 (project)
  - constraint
  - verilog
  - xilinx (prep2_2)

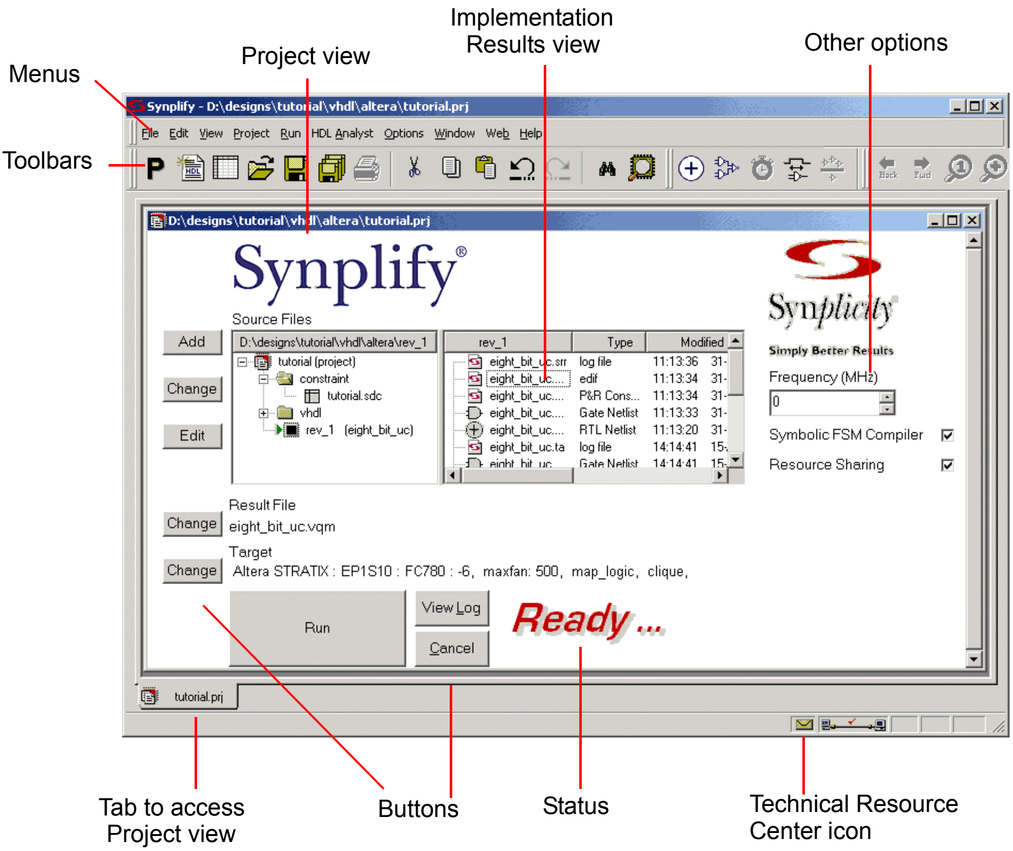
xilinx	Type	Modified	
syntmp			
prep2_2.sdc	constraint	16:02:09	21-Aug-2001
prep2_2.srs	RTL Netlist	16:04:03	24-Sep-2004
prep2_2.tlg	file	16:04:03	24-Sep-2004
prep2_2.fse	file	16:04:04	24-Sep-2004
prep2_2.srm	Gate Netlist	16:04:06	24-Sep-2004
prep2_2.srd	Database N...	16:04:06	24-Sep-2004
rpt_prep2_2.areasrr	file	16:04:07	24-Sep-2004
prep2_2.srr	log file	16:04:07	24-Sep-2004
prep2_2.ncf	P&R Constra...	16:04:07	24-Sep-2004
rpt_prep2_2_areasr...	HTML Log file	16:04:07	24-Sep-2004
prep2_2.edf	edif	16:04:07	24-Sep-2004
prep2_2_srr.htm	HTML Log file	16:04:08	24-Sep-2004
prep2_2.htm	HTML Log file	16:04:08	24-Sep-2004

```
% project -load D:/syn800/examples/verilog/xilinx/prep2 2
%
```

Log Parameter	xilinx

Synplify Interface

The following figure shows you the Synplify interface.


Menus
Toolbars
Project view
Implementation Results view
Other options
File Edit View Project Run HDL Analyst Options Window Web Help
Add
Change
Edit
Change
Change
Run
View Log
Cancel
Tab to access Project view
Buttons
Status
Technical Resource Center icon

Synplify - D:\designs\tutorial\vhdl\altera\tutorial.prj

D:\designs\tutorial\vhdl\altera\tutorial.prj

Synplify®

Source Files

D:\designs\tutorial\vhdl\altera\rev_1

- tutorial (project)
  - constraint
    - tutorial.sdc
  - vhdl
  - rev_1 (eight_bit_uc)

rev_1	Type	Modified
eight_bit_uc.srr	log file	11:13:36 31-
eight_bit_uc....	edif	11:13:34 31-
eight_bit_uc....	P&R Cons...	11:13:34 31-
eight_bit_uc....	Gate Netlist	11:13:33 31-
eight_bit_uc....	RTL Netlist	11:13:20 31-
eight_bit_uc.ta	log file	14:14:41 15-
eight_bit_uc	Gate Netlist	14:14:41 15-

Synplicity

Simply Better Results

Frequency (MHz) 0

Symbolic FSM Compiler ☑

Resource Sharing ☑

Result File

eight_bit_uc.vqm

Target

Altera STRATIX : EP1S10 : FC780 : -6, maxfan: 500, map_logic, clique,

Ready ...

tutorial.prj

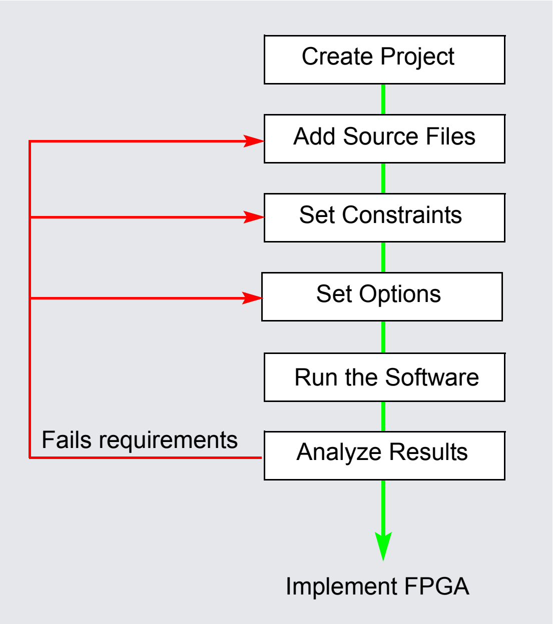
Design Flows

This section provides an overview of the following flows:

- Logic Synthesis Design Flow, on page 1-16
- Prototyping Design Flow, on page 1-17
- Physical Synthesis Design Flows, on page 1-17

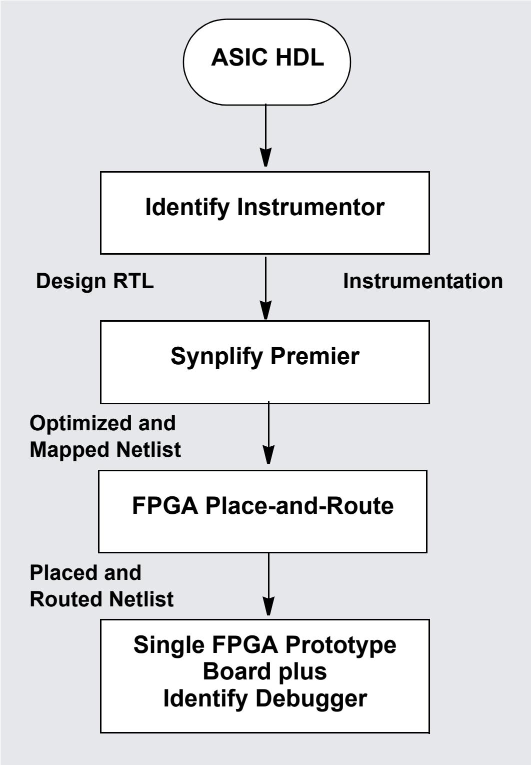
Logic Synthesis Design Flow

The following figure shows a logic synthesis design flow using Synplify, Synplify Pro, and Synplify Premier synthesis software. For a design flow with step-by-step instructions based on specific design data, download the tutorial from the Synplicity website: `http://trc.synplicity.com/tutorials/index.html`.



Prototyping Design Flow

The Synplify Premier software supports a complete design and verification environment featuring the Identify product along with automated HDL code translation.


ASIC HDL
Identify Instrumentor
Design RTL
Instrumentation
Synplify Premier
Optimized and Mapped Netlist
FPGA Place-and-Route
Placed and Routed Netlist
Single FPGA Prototype Board plus Identify Debugger

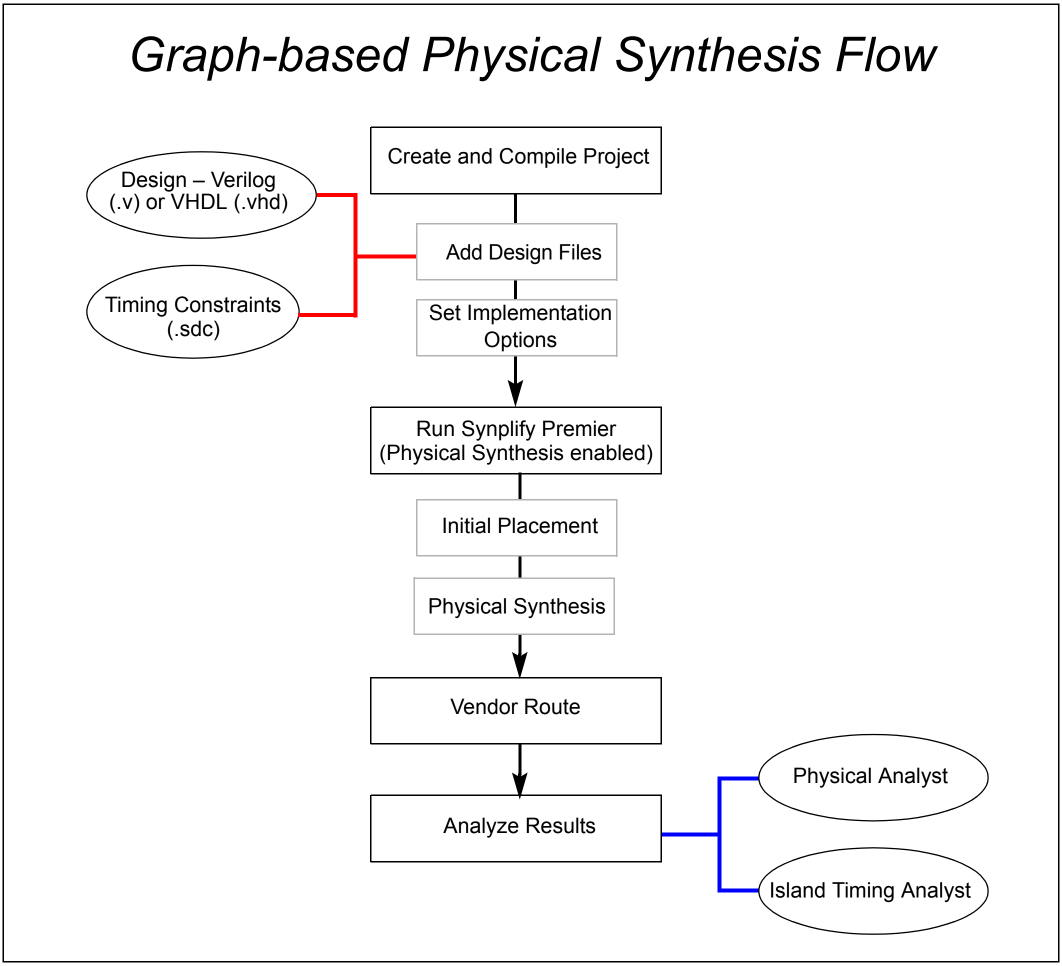
Physical Synthesis Design Flows

Physical synthesis is performed using the Synplify Premier tool. There are three physical synthesis flows:

- Graph-based Physical Synthesis, on page 1-18
- Graph-based Physical Synthesis with a Design Plan, on page 1-19
- Design Plan-based Physical Synthesis, on page 1-20

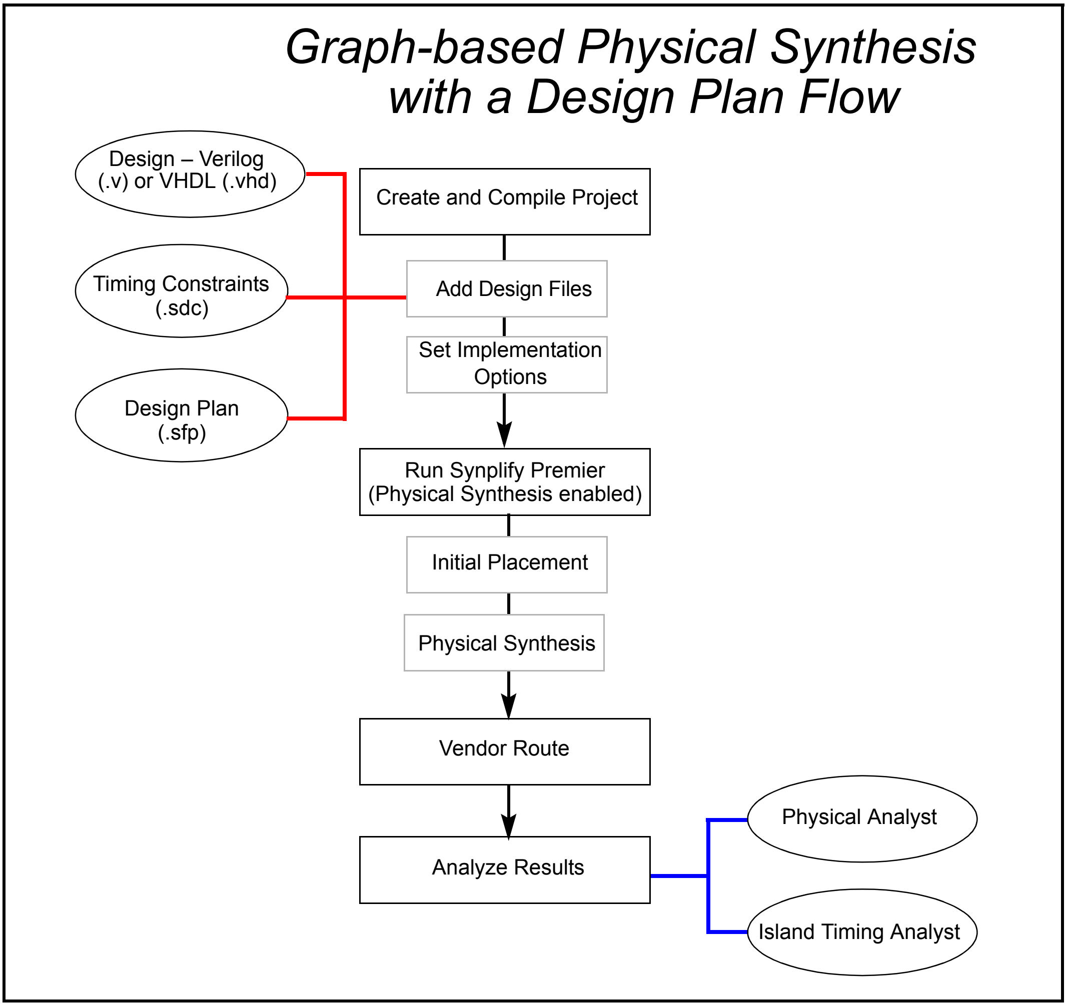
Graph-based Physical Synthesis

The following figure shows the Graph-based Physical Synthesis design flow, which includes the Synplify Premier features and tools to run physical synthesis. Note that this feature is currently applicable for Xilinx Virtex-II Pro, Virtex-4, and Spartan-3 technologies only.


Graph-based Physical Synthesis Flow
Create and Compile Project
Design – Verilog (.v) or VHDL (.vhd)
Add Design Files
Timing Constraints (.sdc)
Set Implementation Options
Run Synplify Premier (Physical Synthesis enabled)
Initial Placement
Physical Synthesis
Vendor Route
Physical Analyst
Analyze Results
Island Timing Analyst

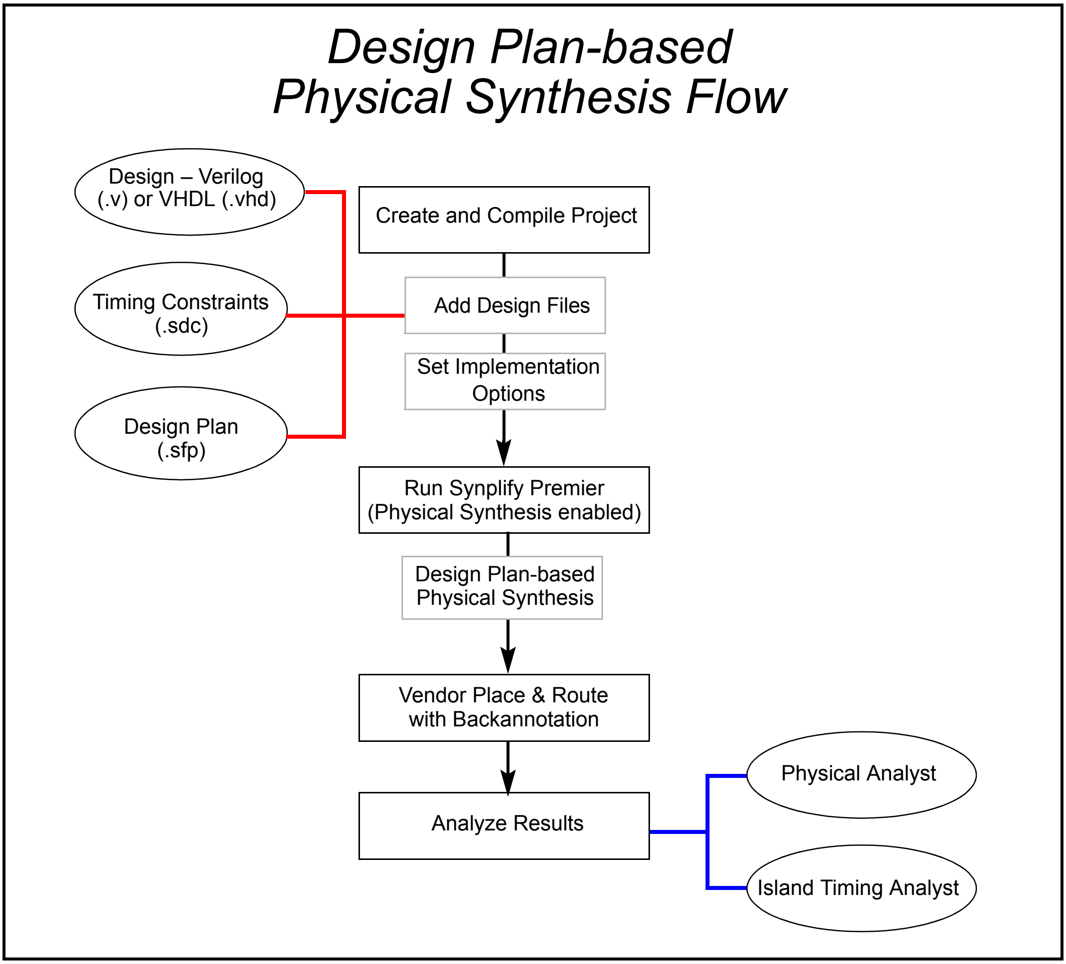
Graph-based Physical Synthesis with a Design Plan

The Graph-based Physical Synthesis with a Design Plan flow for Virtex-II Pro, Virtex-4, and Spartan-3 combines the graph-based push-button flow with detailed placement along with physical optimization using the Design Planner.


Graph-based Physical Synthesis
with a Design Plan Flow
Design – Verilog (.v) or VHDL (.vhd)
Timing Constraints (.sdc)
Design Plan (.sfp)
Create and Compile Project
Add Design Files
Set Implementation Options
Run Synplify Premier (Physical Synthesis enabled)
Initial Placement
Physical Synthesis
Vendor Route
Analyze Results
Physical Analyst
Island Timing Analyst

Design Plan-based Physical Synthesis

The following figure shows the Design-plan based physical synthesis flow using the Design Planner option, which includes the Synplify Premier features and tools to run physical synthesis. Note that this flow applies to Altera Cyclone, Cyclone-II, Stratix, Stratix-GX, Stratix-II and Xilinx Virtex, Virtex-II, and Virtex-E technologies.


Design Plan-based
Physical Synthesis Flow
Design – Verilog (.v) or VHDL (.vhd)
Timing Constraints (.sdc)
Design Plan (.sfp)
Create and Compile Project
Add Design Files
Set Implementation Options
Run Synplify Premier (Physical Synthesis enabled)
Design Plan-based Physical Synthesis
Vendor Place & Route with Backannotation
Analyze Results
Physical Analyst
Island Timing Analyst

CHAPTER 2

Project Setup

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

This section describes how to set up your source files; project file setup is described in *Setting Up Project Files,* on page 2-11. 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:

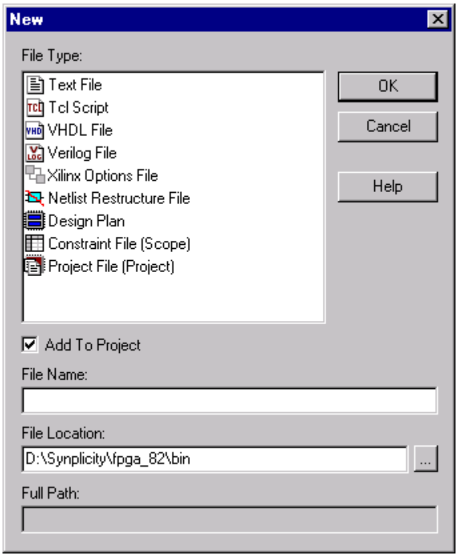
- Creating HDL Source Files, next
- Checking HDL Source Files, on page 2-4
- Editing HDL Source Files with the Built-in Text Editor, on page 2-5
- Using an External Text Editor, on page 2-8
- Setting Editing Window Preferences, on page 2-9

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 2-4 and *Editing HDL Source Files with the Built-in Text Editor,* on page 2-5).

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 2-16.

1. To create a new source file either click the HDL file icon (HDL) 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, make sure to enable the Use Verilog 2001 option before you run synthesis (Project->Implementation Options->Verilog tab).



File Type:

- Text File
- Tcl Script
- VHDL File
- Verilog File
- Xilinx Options File
- Netlist Restructure File
- Design Plan
- Constraint File (Scope)
- Project File (Project)

☑ Add To Project

File Name:

File Location: D:\Synplicity\fpga_82\bin

Full Path:

– Type a name and location for the file and Click OK.

A blank editing window opens with line numbers on the left. You can name it now by pressing Ctrl-s and naming the file.

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 2-5 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 (💾). Use the correct extension for the type of file you created (`.v` or `.vhd`).

   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 2-4.

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. 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:

   ```
   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.

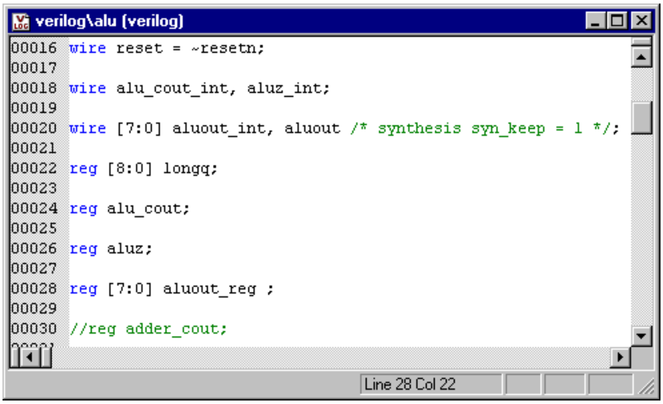
Messages can be categorized as errors, warnings, or notes. Review all messages and resolve any errors. Warnings are less serious than errors, but you must read through and understand them even if you do not resolve all of them. Notes are informative and do not need to be resolved.

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 2-8.

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 2-9.



```
wire reset = ~resetn;

wire alu_cout_int, aluz_int;

wire [7:0] aluout_int, aluout /* synthesis syn_keep = 1 */;

reg [8:0] longq;

reg alu_cout;

reg aluz;

reg [7:0] aluout_reg ;

//reg adder_cout;
```

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

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

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

3. To cut and paste a section of a PDF document, select the T-shaped Text Select icon, highlight the text you need and copy and paste it into your file. The Text Select icon lets you select parts of the document.

4. To create and work with bookmarks in your file, see the following table.

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

To...	Do...
Insert a bookmark	Click anywhere in the line you want to bookmark. Select Edit->Toggle Bookmarks, press Ctrl-F2, or select the first icon in the Edit toolbar. The line number is highlighted to indicate that there is a bookmark at the beginning of that line.
Delete a bookmark	Click anywhere in the line with the bookmark. Select Edit->Toggle Bookmarks, press Ctrl-F2, or select the first icon in the Edit toolbar. The line number is no longer highlighted after the bookmark is deleted.
Delete all bookmarks	Select Edit->Delete all Bookmarks, press Ctrl-Shift-F2, or select the last icon in the Edit toolbar. The line numbers are no longer highlighted after the bookmarks are deleted.
Navigate a file using bookmarks	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.

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 4-51 for details.

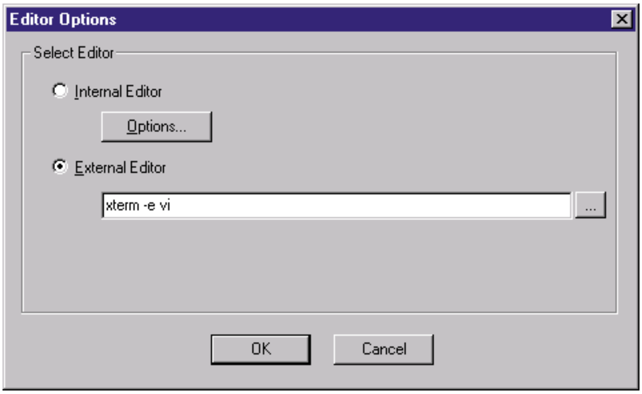
8. When you have fixed all the errors, select File->Save or click the Save icon to save the file.

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 2-5.

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 PC 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.



xterm -e vi

– 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.

Setting Editing Window Preferences

You can customize the fonts and colors used by the internal editor in the Text Editing window.

1. Select Options->Editor Options, and select Internal Editor. Click Options.
2. Select the kind of file for which you want to set the preferences.

   The Text Editing window can be used to set preferences for source files, log files, Tcl files, constraint files, or other default files. The Editor Options form opens.
3. This table shows you how to set some common syntax options from the Editor Options form:

To...	Do This on the Editor Options form...
Set syntax color defaults	Click Syntax coloring. On the Syntax Coloring form, check Use syntax coloring. Set the colors you want for keywords, comments, quotes, and default text by clicking Foreground and Background and selecting colors from the palette. Click OK.
Define comment characters	Click Syntax coloring. On the Syntax Coloring form, type the comment start character(s) in the lower part of the form. Type the comment end characters if necessary. Click OK.
Make the text editor case-sensitive	Click Syntax coloring On the Syntax Coloring form, check Case Sensitive. Click OK.
Set fonts	Click Fonts. On the Font form, set the font and the size. Click OK.
Set tabs	Specify tab size. Specify whether spaces or tabs are to be used to define tabs. Set the display of a tab character.

4. Click OK on the Editor Options form.

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, next
- Opening an Existing Project File, on page 2-14
- Making Changes to a Project, on page 2-15
- Using Mixed Language Source Files, on page 2-16
- Setting Project View Display Preferences, on page 2-18
- Updating Verilog Include Paths in Older Project Files, on page 2-21

Creating a Project File

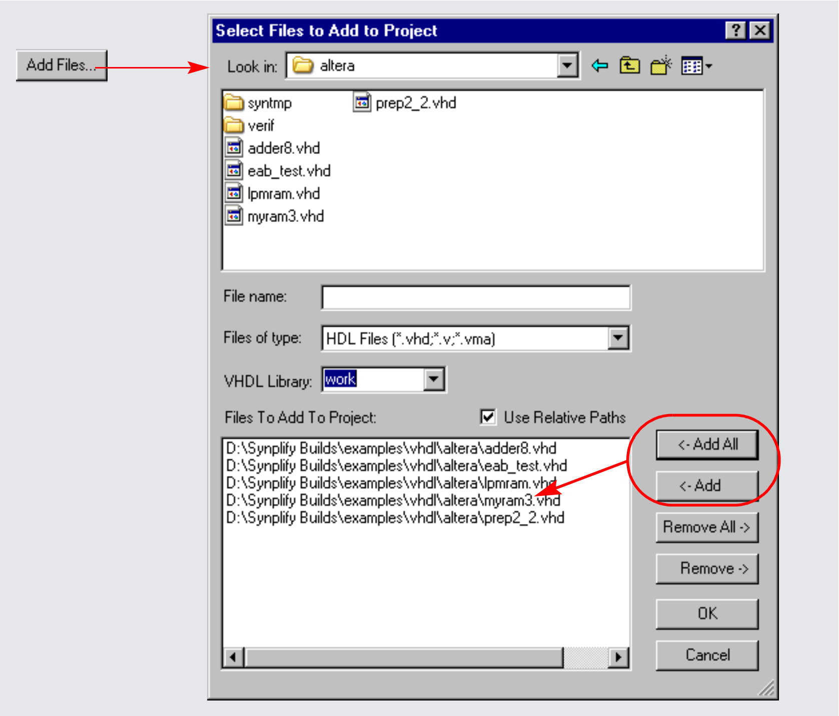
You must set up a project file for each project. A project contains the data needed for a particular design: the list of source files, the synthesis results file, and your device option settings. The following procedure shows you how to set up a project file using individual commands.

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 Select Files to Add 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 2-16.



Look in: altera

syntmp
verif
adder8.vhd
eab_test.vhd
lpmram.vhd
myram3.vhd
prep2_2.vhd

File name:

Files of type: HDL Files (*.vhd;*.v;*.vma)

VHDL Library: work

Files To Add To Project: ☑ Use Relative Paths

D:\Synplify Builds\examples\vhdl\altera\adder8.vhd
D:\Synplify Builds\examples\vhdl\altera\eab_test.vhd
D:\Synplify Builds\examples\vhdl\altera\lpmram.vhd
D:\Synplify Builds\examples\vhdl\altera\myram3.vhd
D:\Synplify Builds\examples\vhdl\altera\prep2_2.vhd

– 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.

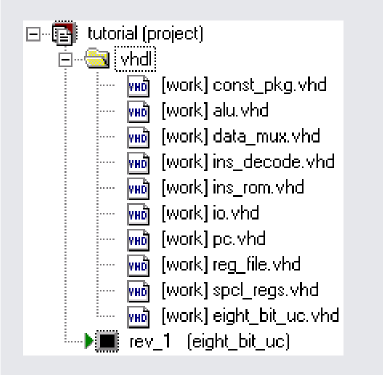
  You can add all the files in the directory and then remove the ones you do not need with the Remove button.

  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.



```
tutorial (project)
  vhdl
    [work] const_pkg.vhd
    [work] alu.vhd
    [work] data_mux.vhd
    [work] ins_decode.vhd
    [work] ins_rom.vhd
    [work] io.vhd
    [work] pc.vhd
    [work] reg_file.vhd
    [work] spcl_regs.vhd
    [work] eight_bit_uc.vhd
  rev_1  (eight_bit_uc)
```

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.

     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 3-11. 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 *Working with Actel Designs*, on page 8-8, *Working with Altera Designs*, on page 8-11, *Working with Lattice Designs*, on page 8-23, and *Working with Xilinx Designs*, on page 8-28.

   - For generic technology components, you can either add the Synplicity technology-independent Verilog library (<*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 **Command**	File->Open **Command**
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. To open a recent project, double-click it from the list of recent projects. Otherwise, click the Existing Project button to open the Open dialog box and select the project.	Select File->Open. Specify the correct directory in the Look In: field. Set File of Type to Project Files (*.prj). The box lists the project files. Double-click on the project you want to open.

The project opens in the Project window.

Making Changes to a Project

Typically, you might have to 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 2-11 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.

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.

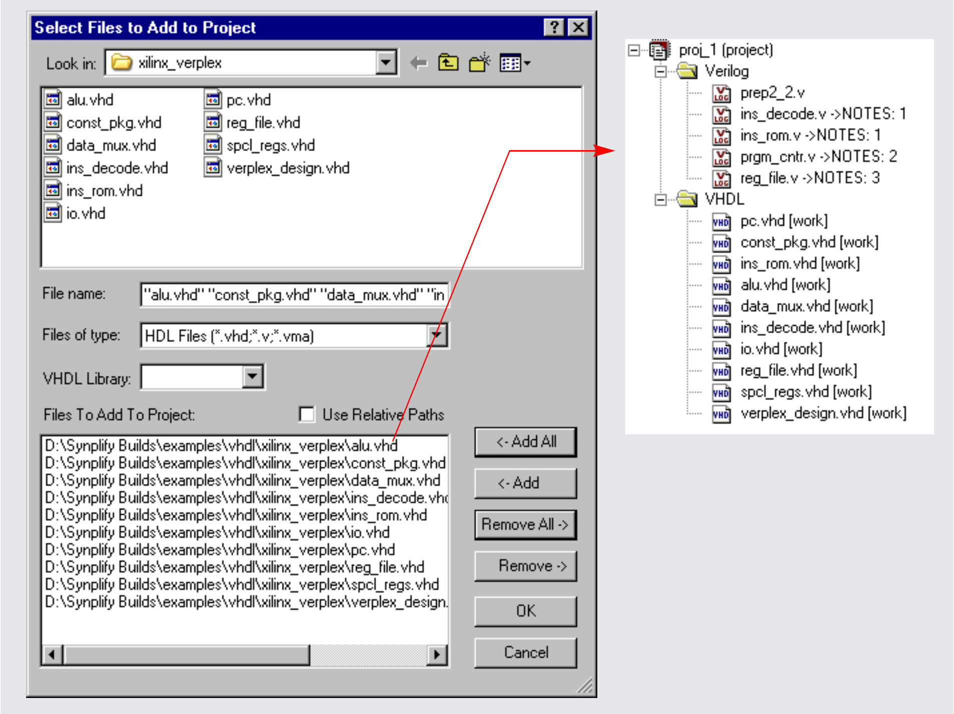
   When you add the files to the project, the Verilog and VHDL files are in separate folders in the Project view.

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 2-15.



Look in: xilinx_verplex

alu.vhd
const_pkg.vhd
data_mux.vhd
ins_decode.vhd
ins_rom.vhd
io.vhd
pc.vhd
reg_file.vhd
spcl_regs.vhd
verplex_design.vhd

File name: "alu.vhd" "const_pkg.vhd" "data_mux.vhd" "in

Files of type: HDL Files (*.vhd;*.v;*.vma)

VHDL Library:

Files To Add To Project: ☐ Use Relative Paths

```
D:\Synplify Builds\examples\vhdl\xilinx_verplex\alu.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\const_pkg.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\data_mux.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\ins_decode.vhc
D:\Synplify Builds\examples\vhdl\xilinx_verplex\ins_rom.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\io.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\pc.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\reg_file.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\spcl_regs.vhd
D:\Synplify Builds\examples\vhdl\xilinx_verplex\verplex_design.
```

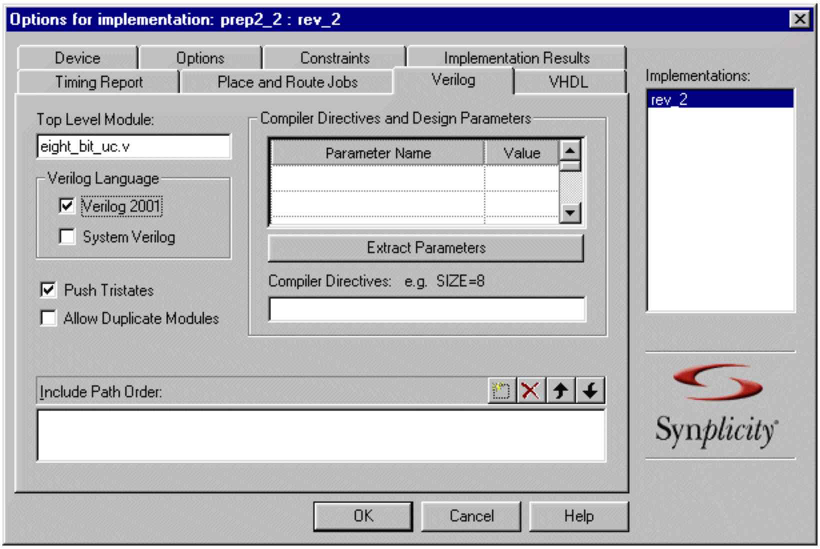
```
proj_1 (project)
  Verilog
    prep2_2.v
    ins_decode.v ->NOTES: 1
    ins_rom.v ->NOTES: 1
    prgm_cntr.v ->NOTES: 2
    reg_file.v ->NOTES: 3
  VHDL
    pc.vhd [work]
    const_pkg.vhd [work]
    ins_rom.vhd [work]
    alu.vhd [work]
    data_mux.vhd [work]
    ins_decode.vhd [work]
    io.vhd [work]
    reg_file.vhd [work]
    spcl_regs.vhd [work]
    verplex_design.vhd [work]
```

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 (Impl Options button), specify the top-level module. For more information about setting device options, see *Setting Implementation Options*, on page 3-2.

   – 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 3-47.



Top Level Module: eight_bit_uc.v

Verilog Language

☑ Verilog 2001

□ System Verilog

☑ Push Tristates

□ Allow Duplicate Modules

Compiler Directives: e.g. SIZE=8

Implementations: rev_2

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 Implementation Options*, on page 3-2.

   – 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.

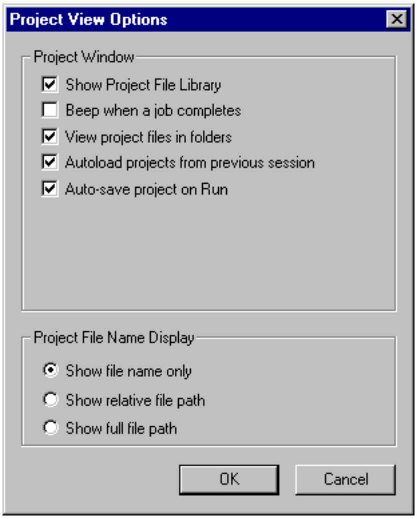
   You can now synthesize your design. The software reads in the mixed formats of the source files and generates a single `.srs` file that is used for synthesis.

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.



Project View Options

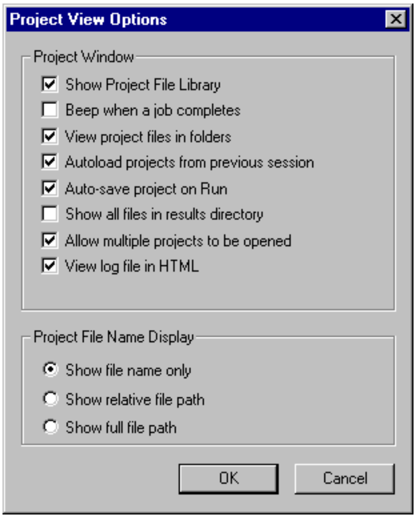
Project Window

- ☑ Show Project File Library
- □ Beep when a job completes
- ☑ View project files in folders
- ☑ Autoload projects from previous session
- ☑ Auto-save project on Run

Project File Name Display

- ● Show file name only
- ○ Show relative file path
- ○ Show full file path

Synplify Options



Project View Options

Project Window

- ☑ Show Project File Library
- □ Beep when a job completes
- ☑ View project files in folders
- ☑ Autoload projects from previous session
- ☑ Auto-save project on Run
- □ Show all files in results directory
- ☑ Allow multiple projects to be opened
- ☑ View log file in HTML

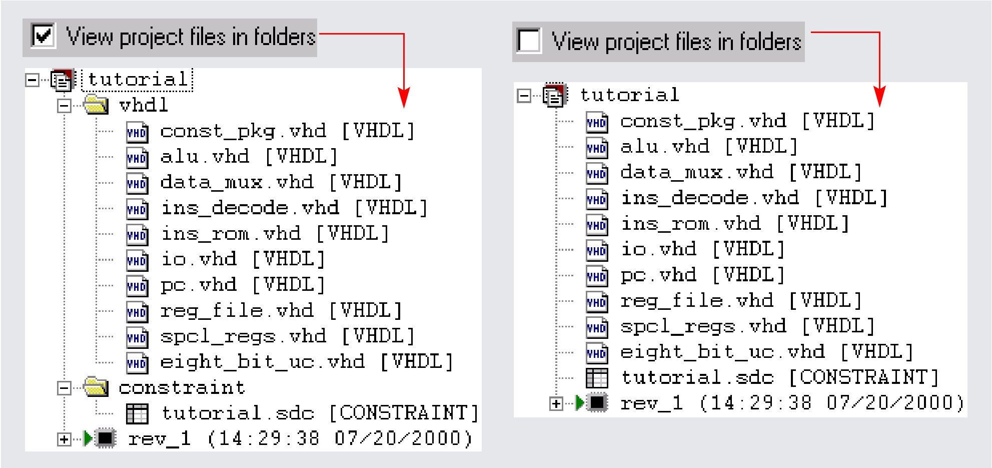
Project File Name Display

- ● Show file name only
- ○ Show relative file path
- ○ Show full file path

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.



☑ View project files in folders

```
tutorial
  vhdl
    const_pkg.vhd [VHDL]
    alu.vhd [VHDL]
    data_mux.vhd [VHDL]
    ins_decode.vhd [VHDL]
    ins_rom.vhd [VHDL]
    io.vhd [VHDL]
    pc.vhd [VHDL]
    reg_file.vhd [VHDL]
    spcl_regs.vhd [VHDL]
    eight_bit_uc.vhd [VHDL]
  constraint
    tutorial.sdc [CONSTRAINT]
  rev_1 (14:29:38 07/20/2000)
```

□ View project files in folders

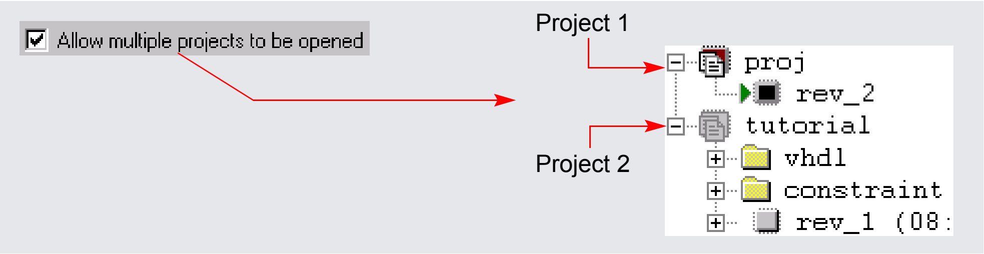
```
tutorial
  const_pkg.vhd [VHDL]
  alu.vhd [VHDL]
  data_mux.vhd [VHDL]
  ins_decode.vhd [VHDL]
  ins_rom.vhd [VHDL]
  io.vhd [VHDL]
  pc.vhd [VHDL]
  reg_file.vhd [VHDL]
  spcl_regs.vhd [VHDL]
  eight_bit_uc.vhd [VHDL]
  tutorial.sdc [CONSTRAINT]
  rev_1 (14:29:38 07/20/2000)
```

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 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.



☑ Allow multiple projects to be opened

5. 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.

6. 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 do 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.

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:

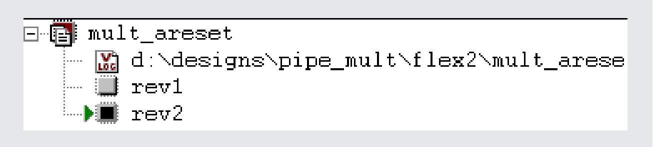
- Working with Multiple Implementations, next
- Creating Workspaces, on page 2-24
- Using Workspaces, on page 2-25

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 New Impl 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 implementation indicated by an arrow.

   ```
   mult_areset
       d:\designs\pipe_mult\flex2\mult_arese
       rev1
     ▶ rev2
   ```

   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.

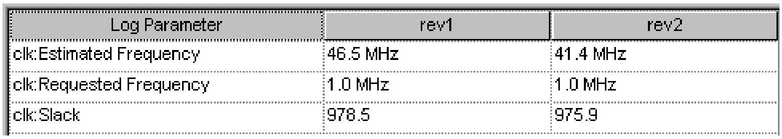
   – To run all the implementations in a project, select Run->Run All Implementations.

   You can use multiple implementations to try a different part or experiment with a different frequency. See *Setting Implementation Options,* on page 3-2 for information about setting options.

   The Project view shows the new implementation. A green arrow marks the current implementation in the Project view. The output files generated for this implementation are shown in the Implementation Results view on the right. The Log Watch Window monitors the current implementation. If you configured it to watch all implementations, it automatically adds the current implementation to 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 4-6 for details.



Log Parameter	rev1	rev2
clk:Estimated Frequency	46.5 MHz	41.4 MHz
clk:Requested Frequency	1.0 MHz	1.0 MHz
clk:Slack	978.5	975.9

   – 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.

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.

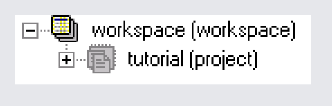
1. To create a new workspace, select File->New Workspace or right-click in the Project view and select Build Workspace.

2. In the dialog box,

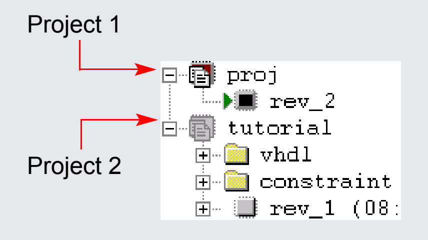
   – Select the project files (`.prj`) of the projects you want to add to the workspace.

   – 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.

Archiving Files and Projects

The archive utility provides a way to archive, extract, or copy your design projects. An archive file is in Synplicity proprietary format and is saved to a file name using the `.sar` extension.

The archive utility is available through the Project menu in the GUI, or by using the project Tcl command. This document provides a description of how to use the utility.

Topics include:

- Archive a Project, on page 2-26
- Extracting Design Files from an Archive, on page 2-31
- Copy a Project, on page 2-34
- Command Syntax, on page 2-37

Archive a Project

Use the archive utility to store the files for a design project into a single archive file in Synplicity Proprietary Format (`.sar`). You can archive an entire project or selected files from the project.

If you want to create a copy of a project without archiving the files, see *Copy a Project*, on page 2-34.

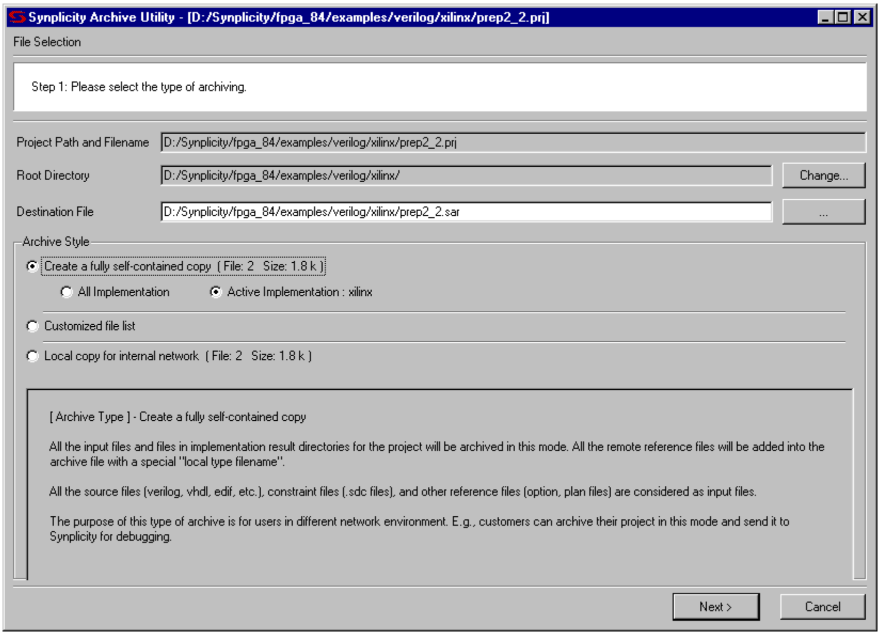
Here are the steps to create an archive:

1. In the GUI (Project view), select Project->Archive Project.

   This command does the following:

   – Automatically runs a syntax check on the active project (Run->Syntax Check command). This is done to ensure that a complete list of project files is generated. For example, in a case where you use Verilog include files in your project, a complete list of Verilog files will be included. Syntax check is automatically run for each implementation in the project to ensure the file list is complete for each implementation as well.

   – Displays the Synplicity Archive Utility wizard.


Synplicity Archive Utility - [D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj]
File Selection
Change...
...
Next >
Cancel

Step 1: Please select the type of archiving.

Project Path and Filename: D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj

Root Directory: D:/Synplicity/fpga_84/examples/verilog/xilinx/

Destination File: D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.sar

Archive Style

- ● Create a fully self-contained copy (File: 2 Size: 1.8 k)
  - ○ All Implementation ● Active Implementation : xilinx
- ○ Customized file list
- ○ Local copy for internal network (File: 2 Size: 1.8 k)

[Archive Type] - Create a fully self-contained copy

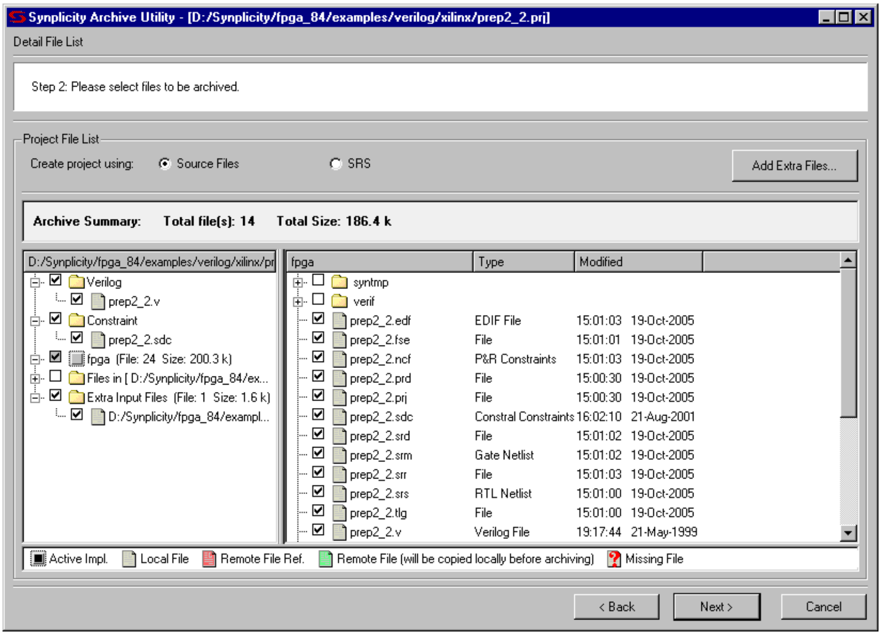
All the input files and files in implementation result directories for the project will be archived in this mode. All the remote reference files will be added into the archive file with a special "local type filename".

All the source files (verilog, vhdl, edif, etc.), constraint files (.sdc files), and other reference files (option, plan files) are considered as input files.

The purpose of this type of archive is for users in different network environment. E.g., customers can archive their project in this mode and send it to Synplicity for debugging.

2. The wizard displays the following information:
   - Name of the design project to archive.
   - Top-level directory where the project (`.prj`) is located. This is considered the root directory.
   - Pathname of the archive file. This is considered the destination file.
3. Choose the Archive Style:
   - Enable Create a fully self-contained copy to archive all project files including project input files and result files. If the project contains more than one implementation, choose to archive only the active implementation, or all implementations in the project.
   - Enable Customized file list to archive only the project files that you select.
   - Enable Local copy for internal network to archive project input files only, no result files and no remote reference files outside the root directory will be included.

4. If you enabled Customized file list, complete the next step, Otherwise, go to Step 7.



Synplicity Archive Utility - [D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj]

Detail File List

Step 2: Please select files to be archived.

Project File List

Create project using: ● Source Files ○ SRS

Archive Summary: Total file(s): 14 Total Size: 186.4 k

D:/Synplicity/fpga_84/examples/verilog/xilinx/pr

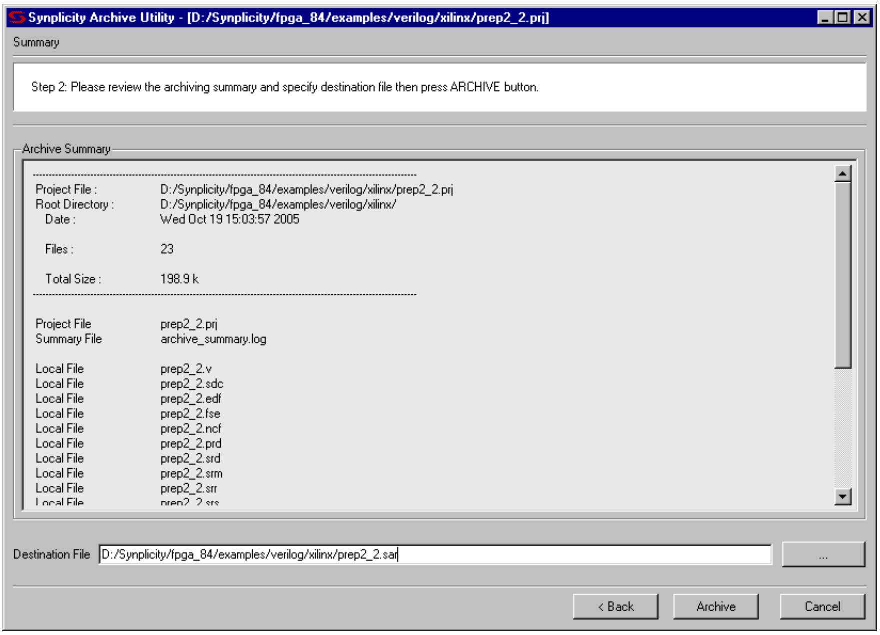
- ☑ Verilog
  - ☑ prep2_2.v
- ☑ Constraint
  - ☑ prep2_2.sdc
- ☑ fpga (File: 24 Size: 200.3 k)
- □ Files in [D:/Synplicity/fpga_84/ex...
- ☑ Extra Input Files (File: 1 Size: 1.6 k)
  - ☑ D:/Synplicity/fpga_84/exampl...

fpga	Type	Modified
□ syntmp		
□ verif		
☑ prep2_2.edf	EDIF File	15:01:03 19-Oct-2005
☑ prep2_2.fse	File	15:01:01 19-Oct-2005
☑ prep2_2.ncf	P&R Constraints	15:01:03 19-Oct-2005
☑ prep2_2.prd	File	15:00:30 19-Oct-2005
☑ prep2_2.prj	File	15:00:30 19-Oct-2005
☑ prep2_2.sdc	Constral Constraints	16:02:10 21-Aug-2001
☑ prep2_2.srd	File	15:01:02 19-Oct-2005
☑ prep2_2.srm	Gate Netlist	15:01:02 19-Oct-2005
☑ prep2_2.srr	File	15:01:03 19-Oct-2005
☑ prep2_2.srs	RTL Netlist	15:01:00 19-Oct-2005
☑ prep2_2.tlg	File	15:01:00 19-Oct-2005
☑ prep2_2.v	Verilog File	19:17:44 21-May-1999

5. Under Project File List, select the file list origin for your archive:
   - Source Files – Includes all HDL files in the archive.
   - SRS – Includes all `.srs` files (RTL schematics) in the archive. (Same as add_file -syn).
   - Use the Add Extra Files button to include additional files in the project.

6. Select the files to include in the archive by clicking on the check boxes next to the filenames.

7. Click Next.



Synplicity Archive Utility - [D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj]

Summary

Step 2: Please review the archiving summary and specify destination file then press ARCHIVE button.

Archive Summary

```
--------------------------------------------------------------------------------
Project File :        D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj
Root Directory :      D:/Synplicity/fpga_84/examples/verilog/xilinx/
  Date :              Wed Oct 19 15:03:57 2005

  Files :             23

  Total Size :        198.9 k
--------------------------------------------------------------------------------

Project File          prep2_2.prj
Summary File          archive_summary.log

Local File            prep2_2.v
Local File            prep2_2.sdc
Local File            prep2_2.edf
Local File            prep2_2.fse
Local File            prep2_2.ncf
Local File            prep2_2.prd
Local File            prep2_2.srd
Local File            prep2_2.srm
Local File            prep2_2.srr
Local File            prep2_2.srs
```

Destination File: D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.sar

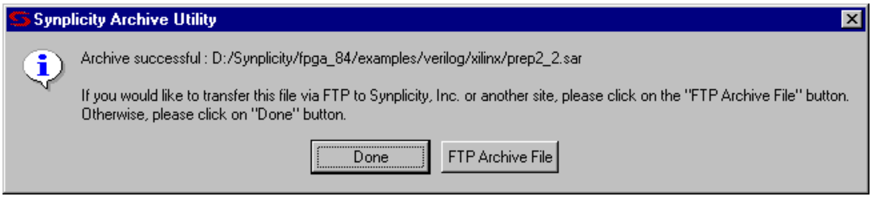
This summary displays all the files in the archive and also shows the full uncompressed file size. Actual size is smaller after the archive. There are no duplicate files.

Note: For Local copy for internal network archives, only the input files are listed.

8. Use the Back button to correct directory or file information and/or follow-up on any missing files, as appropriate.

9. Verify that the current archive contains the files that you want, then click Archive.

   This creates the project archive `.sar` file and displays the following prompt:



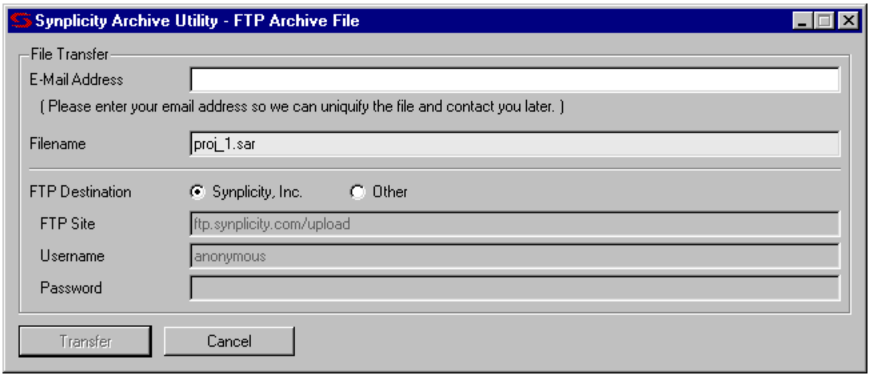
Archive successful : D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.sar

If you would like to transfer this file via FTP to Synplicity, Inc. or another site, please click on the "FTP Archive File" button. Otherwise, please click on "Done" button.

10. Click Done if you are finished.

    Otherwise, you can send the archived file to another site, for example, you can send the design project to the Synplicity FTP site. To do this:

11. Click FTP Archive File.



E-Mail Address

(Please enter your email address so we can uniquify the file and contact you later.)

Filename: proj_1.sar

FTP Destination: ● Synplicity, Inc. ○ Other

FTP Site: ftp.synplicity.com/upload

Username: anonymous

Password:

12. Fill in the following information:

    – Your email address. This email address, plus a date and time stamp are prepended to the `.sar` file name to uniquely identify your archive file.

    – Details on the FTP site destination, including username and password for sending to sites other than Synplicity.

13. Click Transfer.

    This completes the archive transfer. The Status field in the dialog box displays the transfer results.

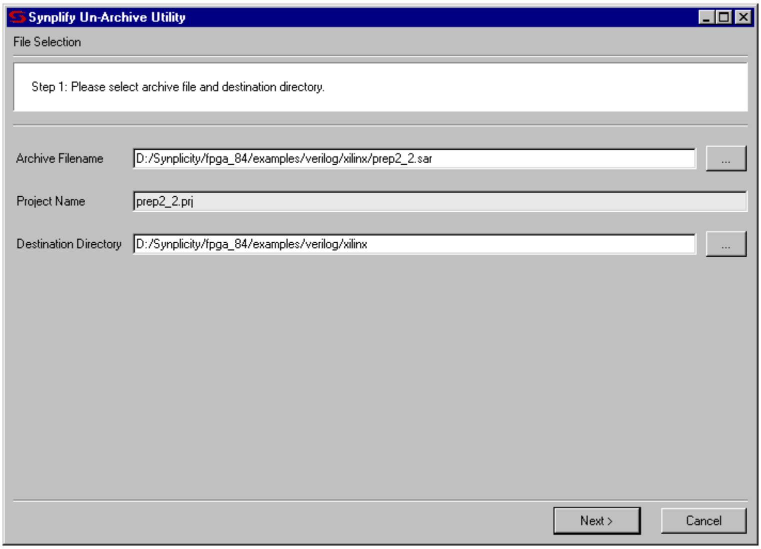
Extracting Design Files from an Archive

Use this utility to extract design project files from an archive file (`.sar`).

Here are the steps:

1. In the GUI (Project view), select Project->Un-Archive Project.

   This displays the Synplicity Un-Archive Utility wizard.



Synplify Un-Archive Utility

File Selection

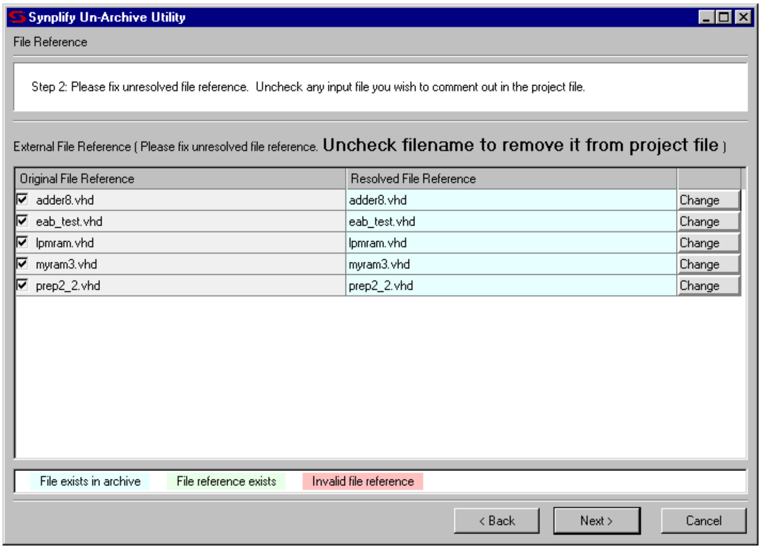
Step 1: Please select archive file and destination directory.

Archive Filename D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.sar

Project Name prep2_2.prj

Destination Directory D:/Synplicity/fpga_84/examples/verilog/xilinx

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).
3. Click Next.



Step 2: Please fix unresolved file reference. Uncheck any input file you wish to comment out in the project file.

External File Reference (Please fix unresolved file reference. Uncheck filename to remove it from project file)

Original File Reference	Resolved File Reference	
☑ adder8.vhd	adder8.vhd	Change
☑ eab_test.vhd	eab_test.vhd	Change
☑ lpmram.vhd	lpmram.vhd	Change
☑ myram3.vhd	myram3.vhd	Change
☑ prep2_2.vhd	prep2_2.vhd	Change

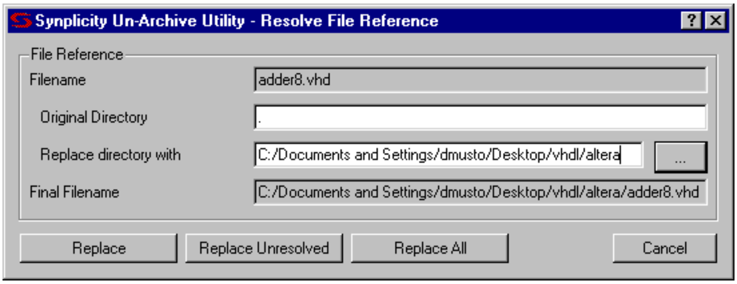
File exists in archive File reference exists Invalid file reference

4. 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 extracted, unchecked the box next to the file. The unchecked 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 extracting, 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.



Filename: adder8.vhd

Original Directory: .

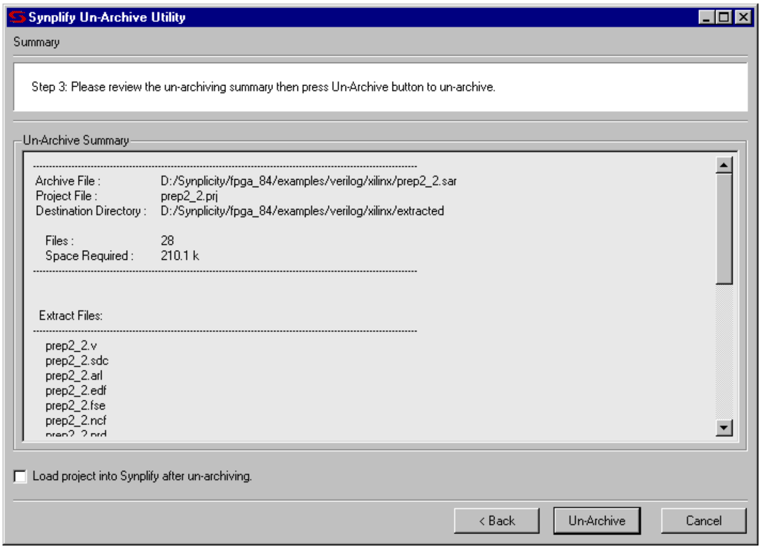
Replace directory with: C:/Documents and Settings/dmusto/Desktop/vhdl/altera

Final Filename: C:/Documents and Settings/dmusto/Desktop/vhdl/altera/adder8.vhd

The Replace directory with field specifies the new location of the project files you want to use:

- Replace – replaces only the specified file (Filename field) in the project.
- Replace Unresolved – replaces any unresolved files in the project, with files of the same name from the Replace directory.
- Replace All – replaces all files in the archived project with files of the same name from the Replace directory.
- To undo replace-file references, clear the Replace directory with field, then click Replace. This causes the utility to point back to the Original Directory.

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



Step 3: Please review the un-archiving summary then press Un-Archive button to un-archive.

```
------------------------------------------------------------------------------------------------
Archive File :            D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.sar
Project File :            prep2_2.prj
Destination Directory :   D:/Synplicity/fpga_84/examples/verilog/xilinx/extracted

  Files :                 28
  Space Required :        210.1 k
------------------------------------------------------------------------------------------------


 Extract Files:
------------------------------------------------------------------------------------------------
  prep2_2.v
  prep2_2.sdc
  prep2_2.arl
  prep2_2.edf
  prep2_2.fse
  prep2_2.ncf
  prep2_2.prd
```

☐ Load project into Synplify after un-archiving.

6. If you want to load this project in the UI after files are extracted, enable the Load project into Synplicity after un-archiving option.

7. Click Un-Archive.

   A message dialog box is displayed while the files are being extracted.

8. 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 a copy of a design project. You can copy an entire project, or selected files from the project.

If you want to create an archive of the project where the project is stored in a single archive file, see *Archive a Project*, on page 2-26.

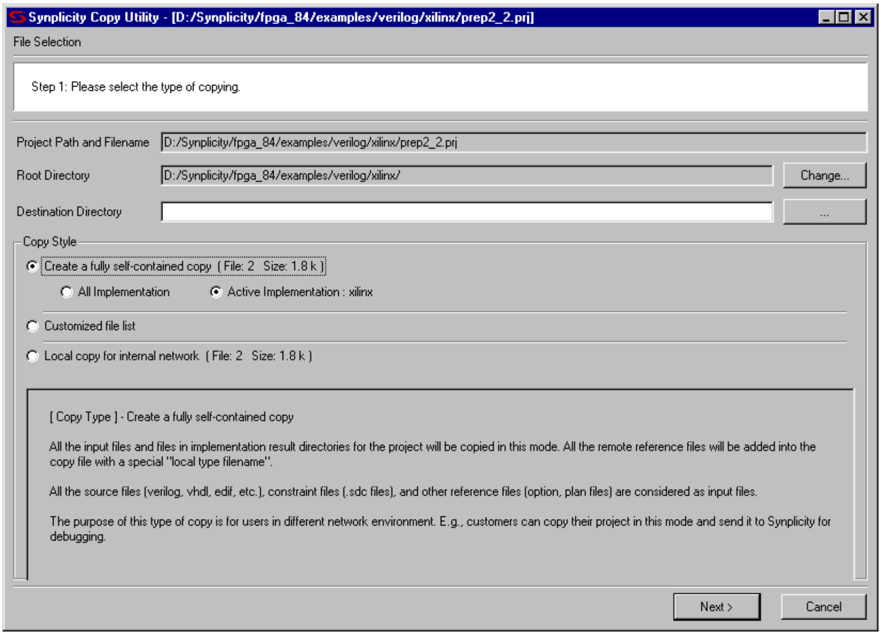
Here are the steps to create a copy of a design project:

1. From the GUI (Project view), select Project->Copy Project.

   This command does the following:

   – Automatically runs a syntax check on the active project (Run->Syntax Check command) to ensure that a complete list of project files is generated. This is done in case you use Verilog include files in your project. This syntax check is automatically run for each implementation in the project to ensure the file list is complete for each implementation as well.

   – Displays the Synplicity Copy Utility wizard.


Synplicity Copy Utility - [D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj]
File Selection
Change...
...
Next >
Cancel

Step 1: Please select the type of copying.

Project Path and Filename: D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj

Root Directory: D:/Synplicity/fpga_84/examples/verilog/xilinx/

Destination Directory:

Copy Style

● Create a fully self-contained copy (File: 2 Size: 1.8 k)

○ All Implementation ● Active Implementation : xilinx

○ Customized file list

○ Local copy for internal network (File: 2 Size: 1.8 k)

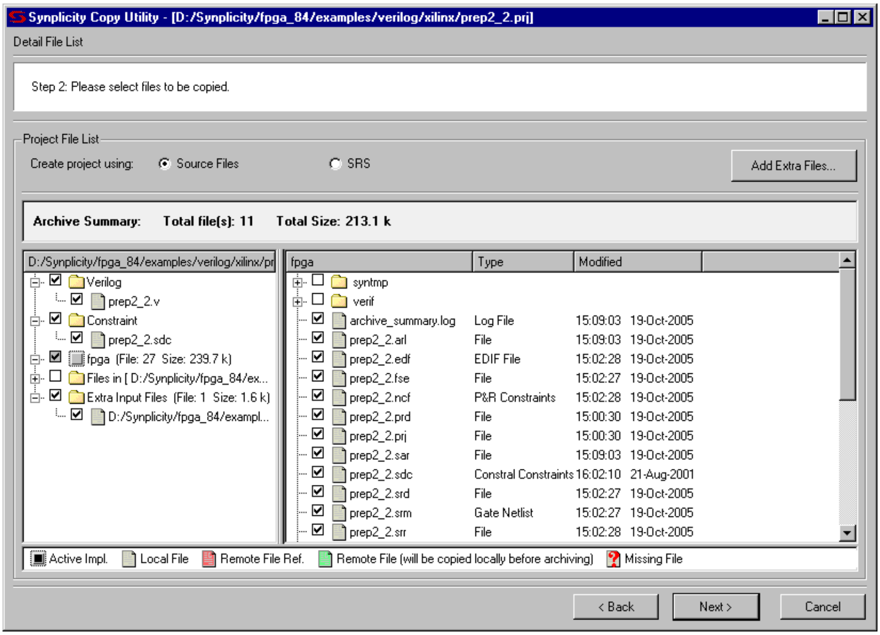
[Copy Type] - Create a fully self-contained copy

All the input files and files in implementation result directories for the project will be copied in this mode. All the remote reference files will be added into the copy file with a special "local type filename".

All the source files (verilog, vhdl, edif, etc.), constraint files (.sdc files), and other reference files (option, plan files) are considered as input files.

The purpose of this type of copy is for users in different network environment. E.g., customers can copy their project in this mode and send it to Synplicity for debugging.

2. The wizard displays:
   - Name of selected design project to copy.
   - Top-level directory where the project is located.
   - Destination directory in which to copy files.
3. Choose the Copy Style:
   - Enable Create a fully self-contained copy to copy all project files including project input files and result files. If the project contains more than one implementation, choose to copy only the active implementation or all implementations in the project.
   - Enable Local copy for internal network to copy project input files only, no result files will be included.
   - Enable Customized file list to copy only the project files that you select.
4. If you enable Customized file list, complete the next step, otherwise, go to Step 7.



Synplicity Copy Utility - [D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj]

Detail File List

Step 2: Please select files to be copied.

Project File List

Create project using: ● Source Files ○ SRS

Archive Summary: Total file(s): 11 Total Size: 213.1 k

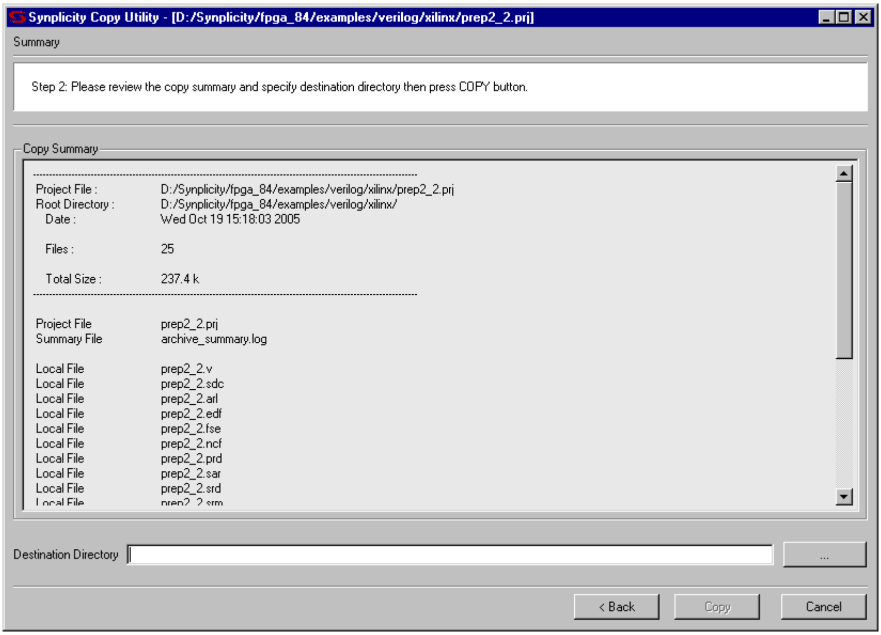
D:/Synplicity/fpga_84/examples/verilog/xilinx/pr
- ☑ Verilog
  - ☑ prep2_2.v
- ☑ Constraint
  - ☑ prep2_2.sdc
- ☑ fpga (File: 27 Size: 239.7 k)
- □ Files in [D:/Synplicity/fpga_84/ex...
- ☑ Extra Input Files (File: 1 Size: 1.6 k)
  - ☑ D:/Synplicity/fpga_84/exampl...

fpga	Type	Modified	
□ syntmp			
□ verif			
☑ archive_summary.log	Log File	15:09:03	19-Oct-2005
☑ prep2_2.arl	File	15:09:03	19-Oct-2005
☑ prep2_2.edf	EDIF File	15:02:28	19-Oct-2005
☑ prep2_2.fse	File	15:02:27	19-Oct-2005
☑ prep2_2.ncf	P&R Constraints	15:02:28	19-Oct-2005
☑ prep2_2.prd	File	15:00:30	19-Oct-2005
☑ prep2_2.prj	File	15:00:30	19-Oct-2005
☑ prep2_2.sar	File	15:09:03	19-Oct-2005
☑ prep2_2.sdc	Constral Constraints	16:02:10	21-Aug-2001
☑ prep2_2.srd	File	15:02:27	19-Oct-2005
☑ prep2_2.srm	Gate Netlist	15:02:27	19-Oct-2005
☑ prep2_2.srr	File	15:02:28	19-Oct-2005

5. Under Project File List, select the file list origin for your archive:
   - Source Files – Includes all HDL files in the archive.

- SRS – Includes all `.srs` files (RTL schematics) in the archive. (Same as add_file -syn).
- Use the Add Extra Files button to include additional files in the project.

6. Select the files to include in the archive by clicking on the check boxes next to the filenames.
7. Click Next.



Synplicity Copy Utility - [D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj]

Summary

Step 2: Please review the copy summary and specify destination directory then press COPY button.

Copy Summary

```
------------------------------------------------------------------------
Project File :        D:/Synplicity/fpga_84/examples/verilog/xilinx/prep2_2.prj
Root Directory :      D:/Synplicity/fpga_84/examples/verilog/xilinx/
  Date :              Wed Oct 19 15:18:03 2005

  Files :             25

  Total Size :        237.4 k
------------------------------------------------------------------------

Project File          prep2_2.prj
Summary File          archive_summary.log

Local File            prep2_2.v
Local File            prep2_2.sdc
Local File            prep2_2.arl
Local File            prep2_2.edf
Local File            prep2_2.fse
Local File            prep2_2.ncf
Local File            prep2_2.prd
Local File            prep2_2.sar
Local File            prep2_2.srd
Local File            prep2_2.srm
```

Destination Directory

8. Verify the copy information.
9. Enter a destination directory. If the directory does not exist it will be created.
10. Click Copy to create the project copy.

Command Syntax

The Tcl command syntax for archiving projects and files is as follows:

- project -archive
- project -copy
- project -unarchive

For a complete description of the project Tcl command options for archiving, see *project*, on page 5-14.

CHAPTER 3

Constraints, Attributes, and Options

This chapter describes the typical options you set when working through the synthesis design flow. It covers the following:

Setting 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 *Setting Constraints in the SCOPE Window*, on page 3-18. For information about overriding global settings with individual attributes or directives, see *Adding Attributes and Directives*, on page 3-66.

This section discusses the following topics:

- Setting Device Options, next
- Setting Optimization Options, on page 3-5
- Specifying Global Frequency and Constraint Files, on page 3-6
- Specifying Result Options, on page 3-9
- Specifying Timing Report Output, on page 3-10
- Setting Verilog and VHDL Options, on page 3-11
- Setting Synplify Premier Netlist Restructuring Optimizations, on page 3-16

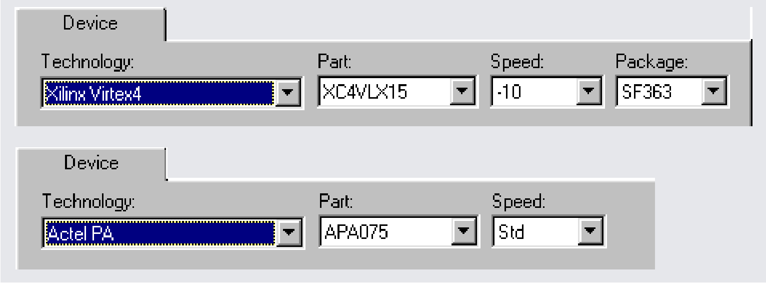
Setting Device Options

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 Options for Implementation form by clicking the Impl 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 the Synplify and Synplify Pro tools do.



Device

Technology: Xilinx Virtex4
Part: XC4VLX15
Speed: -10
Package: SF363

Device

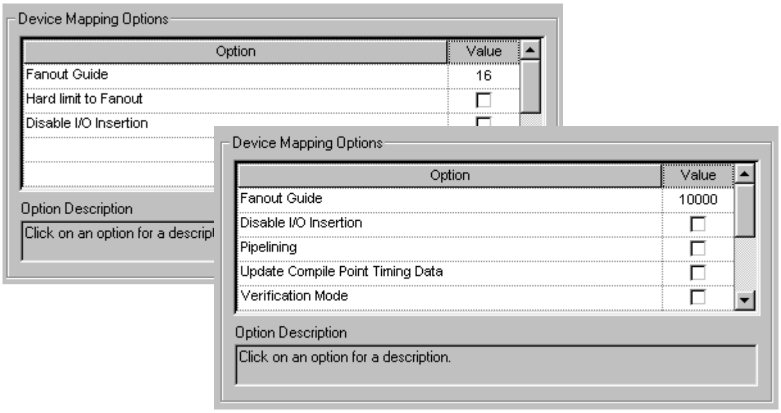
Technology: Actel PA
Part: APA075
Speed: Std

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 6-7, *Pipelining*, on page 6-40, and *Retiming*, on page 6-44, 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

Option	Value
Fanout Guide	16
Hard limit to Fanout	□
Disable I/O Insertion	□

Option Description

Click on an option for a descripl

Device Mapping Options

Option	Value
Fanout Guide	10000
Disable I/O Insertion	□
Pipelining	□
Update Compile Point Timing Data	□
Verification Mode	□

Option Description

Click on an option for a description.

4. Set other implementation options as needed (see *Setting Implementation Options, on page 3-2* 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 form mapped to the equivalent Tcl commands. Because the options are technology-based, all 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 (set_option...)**
Area delay percent (Altera, Lattice, Xilinx)	`-area_delay_percent` *`net_percentage`*
Cliquing (Altera)	`-cliquing {true|false}`

Disable I/O insertion	`-disable_io_insertion`
	`{true|false}`

Fan-in limit (Altera, Lattice)	`-fanin_limit` *`max_fanin`*
Fanout guide (Actel, Atmel, Xilinx)	`-fanout_guide` *`fanout_value`*
Fanout limit	`-fanout_limit` *`limit`*
Fanout limit (hard) (Actel)	`-maxfan_hard {true|false}`
Force GSR usage (Lattice, Xilinx)	`-force_gsr {true|false}`
Map logic (Altera, Lattice, Xilinx)	`-map_logic {true|false}`
Maximum terms/macrocell (Lattice, Xilinx)	`-max_terms_per_macrocell` *`max_terms`*
Package	`-package` *`pkg_name`*
Part	`-part` *`part_name`*
Soft buffers (Altera)	`-soft_buffers {true|false}`

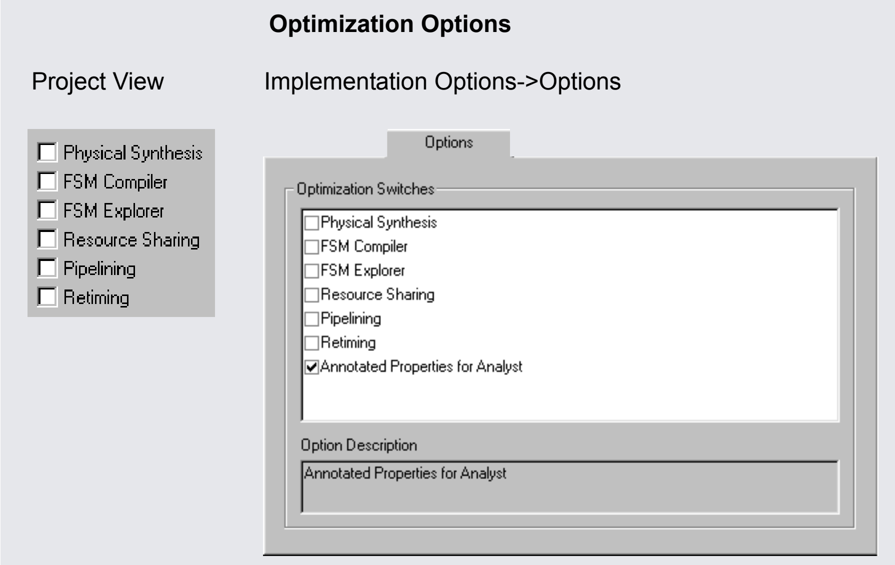
Option	**Tcl Command (set_option...)**
Speed	`-speed_grade` *`speed_grade`*
Technology	`-technology` *`keyword`*

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 Options for Implementation form by clicking the Impl 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

- □ Physical Synthesis
- □ FSM Compiler
- □ FSM Explorer
- □ Resource Sharing
- □ Pipelining
- □ Retiming

Implementation Options->Options

Optimization Switches

- □ Physical Synthesis
- □ FSM Compiler
- □ FSM Explorer
- □ Resource Sharing
- □ Pipelining
- □ Retiming
- ☑ Annotated Properties for Analyst

Option Description: Annotated Properties for Analyst

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

FSM Compiler	*Using the Symbolic FSM Compiler*, on page 6-17
FSM Explorer	*Using FSM Explorer*, on page 6-22
Resource Sharing	*Sharing Resources*, on page 6-5
Pipelining	*Pipelining*, on page 6-40
Retiming	*Retiming*, on page 6-44
Annotated Properties for Analyst	Annotates the design with generic non-timing instance properties (.sap) and timing properties (.tap). These properties are then viewable in the RTL View and Design Planner, as well as used to create collections using TCL Find. See *Object Properties*, on page 5-54 and *Annotated Timing Information*, on page 6-34 for more information.

The equivalent Tcl set_option command options are -frequency, -frequency auto, -resource_sharing, -use_fsm_explorer, -pipe, -retiming and -symbolic_fsm_compiler.

3. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 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.

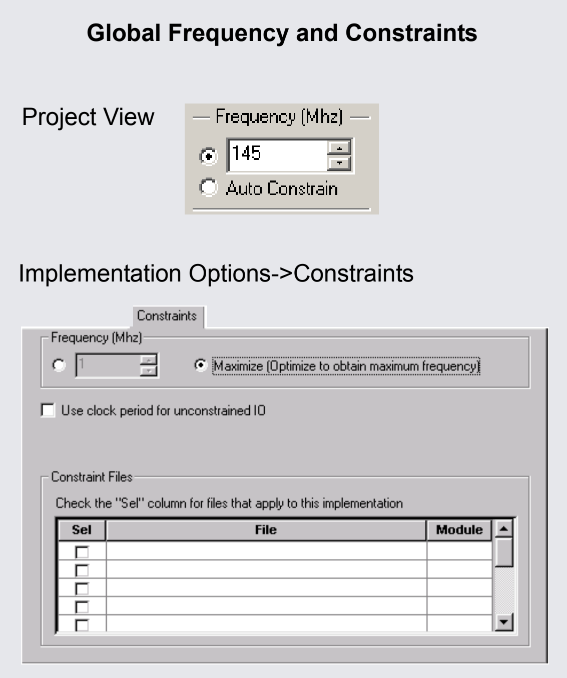
Specifying Global Frequency and Constraint Files

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 Options for Implementation form by clicking the Impl 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 *Setting Constraints in the SCOPE Window*, on page 3-18. 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 3-45 for details.



Global Frequency and Constraints

Project View

Frequency (Mhz)

● 145

○ Auto Constrain

Implementation Options->Constraints

Frequency (Mhz)

○ 1

● Maximize (Optimize to obtain maximum frequency)

□ Use clock period for unconstrained IO

Constraint Files

Check the "Sel" column for files that apply to this implementation

Sel	File	Module
□		
□		
□		
□		
□		

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 *Setting Constraints in the SCOPE Window*, on page 3-18.

3. To remove constraint files from an implementation, do one of the following:

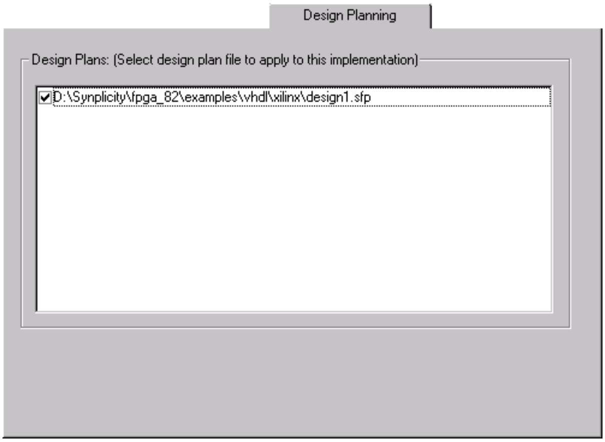
   – Select Project->Implementation Options->Constraints. Click off the checkbox next to the file name.

- 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->Synplify Premier. 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.


Design Planning
Design Plans: (Select design plan file to apply to this implementation)

☑ D:\Synplicity\fpga_82\examples\vhdl\xilinx\design1.sfp

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.

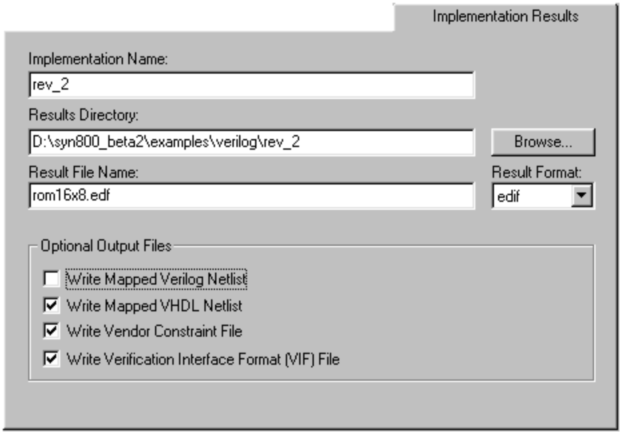
5. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 for a list of choices). Click OK.

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



Implementation Name: rev_2

Results Directory: D:\syn800_beta2\examples\verilog\rev_2

Result File Name: rom16x8.edf

Result Format: edif

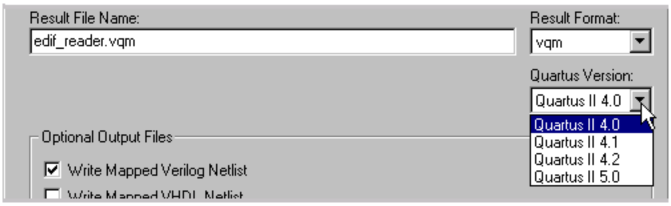
Optional Output Files

- □ 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 3-64 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 *Targeting Output to Your Vendor,* on page 8-6), the .vqm result format allows you to also select the version of Quartus II you are using from the pop-up menu.



Result File Name: edif_reader.vqm

Result Format: vqm

Quartus Version: Quartus II 4.0

Optional Output Files

☑ Write Mapped Verilog Netlist

□ Write Mapped VHDL Netlist

5. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 for a list of choices). Click OK.

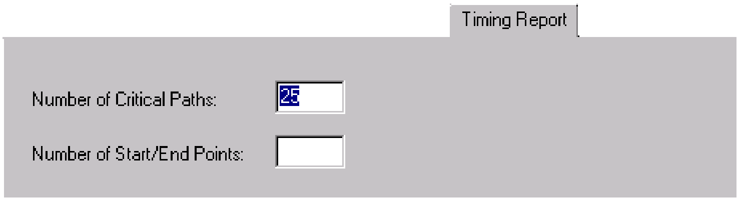
   When you synthesize the design, the software compiles and maps the design using the options you set.

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, Spartan-3, and Altera Stratix technologies. For more information about generating island timing reports, see *Generating the Island Timing Report*, on page 4-83.

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



Number of Critical Paths: 25

Number of Start/End Points:

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 Implementation Options*, on page 3-2 for a list of choices). Click OK.

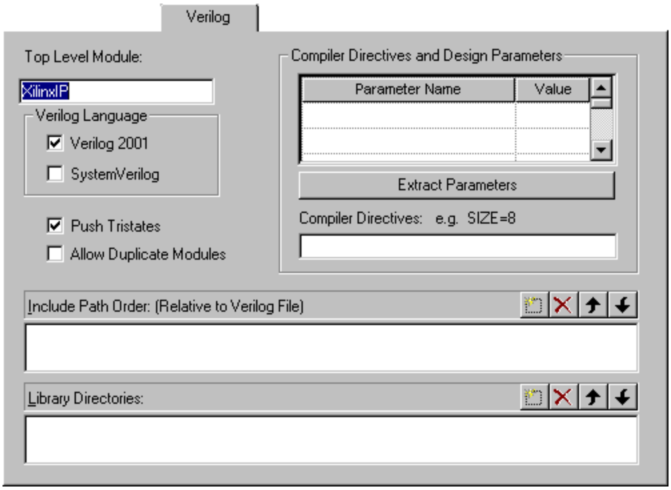
When you synthesize the design, the software compiles and maps the design using the options you set.

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 3-16.


Verilog
Compiler Directives and Design Parameters
Parameter Name
Value
Extract Parameters
Compiler Directives: e.g. SIZE=8
Include Path Order: (Relative to Verilog File)
Library Directories:

Top Level Module: XilinxIP

Verilog Language

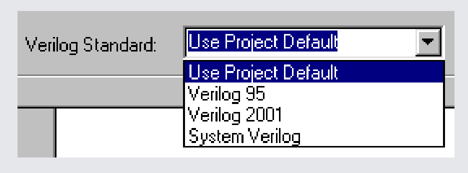
☑ Verilog 2001

☐ SystemVerilog

☑ Push Tristates

☐ Allow Duplicate Modules

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, 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.



Verilog Standard: Use Project Default

2. Specify the top-level module if you did not already do this in the Project view.

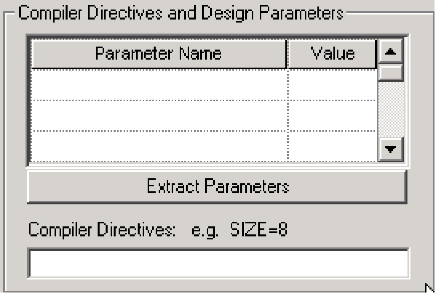
3. To extract parameters from the source code, do the following:

   – 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: Parameter extraction is not supported for mixed designs.



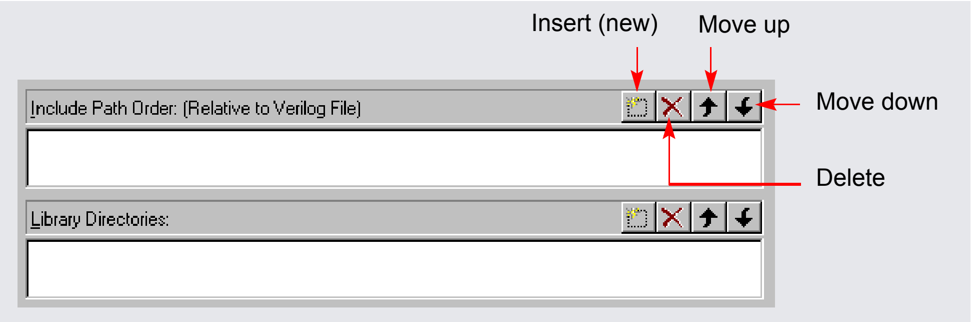
Compiler Directives and Design Parameters

Parameter Name	Value

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.

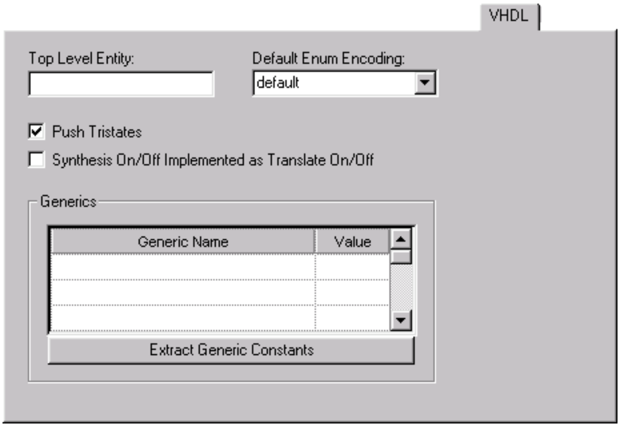
6. In the Library Directories, specify the path to the directory which contains the library files for your project. Use the buttons in the upper right corner of the box to add, delete, or reorder the paths.

7. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 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.



Top Level Entity:

Default Enum Encoding: default

☑ Push Tristates

□ Synthesis On/Off Implemented as Translate On/Off

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 3-16.

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 3-47.

   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 6-14.

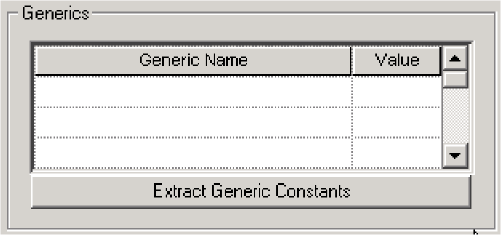
3. To extract generics from the source code, do this:

   – 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.

Note: Generic constraint extraction is not supported for mixed designs.



Generics

Generic Name	Value

4. To push tristates across process/block boundaries, check that Push Tristates is enabled. For details, see *Push Tristates Option,* on page 3-51 in the *Reference Manual*.

5. Determine the interpretation of the synthesis_on and synthesis_off directives:

   – To make the compiler to treat synthesis_on and synthesis_off directives like translate_on/translate_off, enable the Synthesis On/Off Implemented as Translate On/Off option.

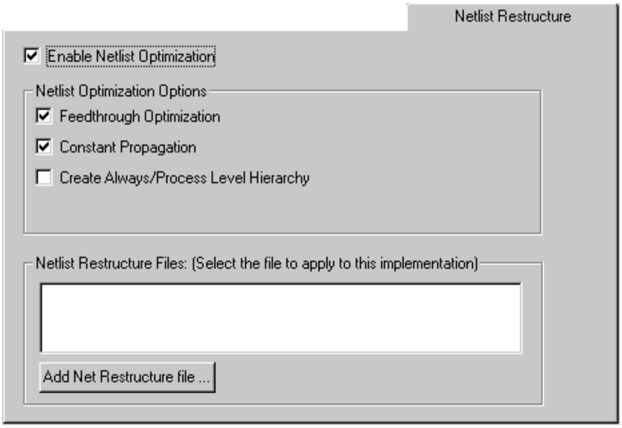
   – To ignore the synthesis_on and synthesis_off directives, make sure that this option is not checked. See *translate_off/translate_on Directive,* on page 8-218 in the *Reference Manual* for more information.

6. Set other implementation options as needed (see *Setting Implementation Options,* on page 3-2 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 Synplify Premier Netlist Restructuring Optimizations

You can only set netlist restructuring options in the Synplify Premier tool. To do so, select Project->Implementation Options, and click on the Netlist Restructure tab.



☑ Enable Netlist Optimization

Netlist Optimization Options

☑ Feedthrough Optimization

☑ Constant Propagation

☐ Create Always/Process Level Hierarchy

Netlist Restructure Files: (Select the file to apply to this implementation)

1. To reduce the number of ports, eliminate feedthrough ports by enabling Feedthrough Optimization. This can improve routability in the place-and-route tool.

2. 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.

3. 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.

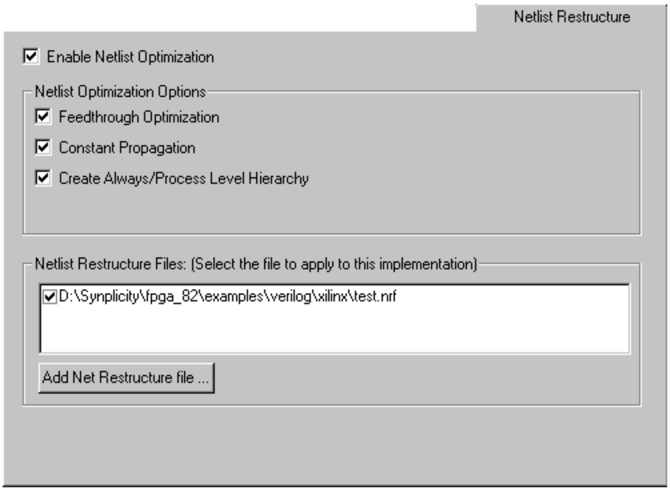
   For example, in Verilog, the always block becomes a module with the signals in the sensitivity list becoming inputs of the module and the signals that get their values set becoming outputs of the modules. Similarly, in VHDL, a process becomes a module. You might find that it is easier to apply a design plan to these always blocks/processes.

4. To group Altera Stratix MAC configurations together into one MAC block, enable Create MAC Hierarchy.

5. 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.



Netlist Restructure

☑ Enable Netlist Optimization

Netlist Optimization Options

☑ Feedthrough Optimization

☑ Constant Propagation

☑ Create Always/Process Level Hierarchy

Netlist Restructure Files: (Select the file to apply to this implementation)

☑ D:\Synplicity\fpga_82\examples\verilog\xilinx\test.nrf

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.

6. Set other implementation options as needed (see *Setting Implementation Options*, on page 3-2 for a list of choices). Click OK.

Setting Constraints in the SCOPE Window

You can use a text editor to create a constraint file as described in *Working with Constraint Files*, on page 3-60, 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.

This section describes the following:

- Using the SCOPE Window, next
- Entering and Editing Constraints in the SCOPE Window, on page 3-21
- Entering Default Constraints, on page 3-24
- Setting Clock and Path Constraints, on page 3-26
- Defining Clocks, on page 3-28
- Defining Input and Output Constraints, on page 3-33
- Defining Multicycle Paths, on page 3-34
- Defining From/To/Through for Timing Exceptions, on page 3-35
- Defining False Paths, on page 3-38
- Specifying Standard I/O Pad Types, on page 3-40
- Setting SCOPE Display Preferences, on page 3-41
- Translating Xilinx UCF Constraints, on page 3-42
- Converting Pin Location Constraint Files in the Synplify Premier Tool, on page 3-43

You can also use the SCOPE window to do the following:

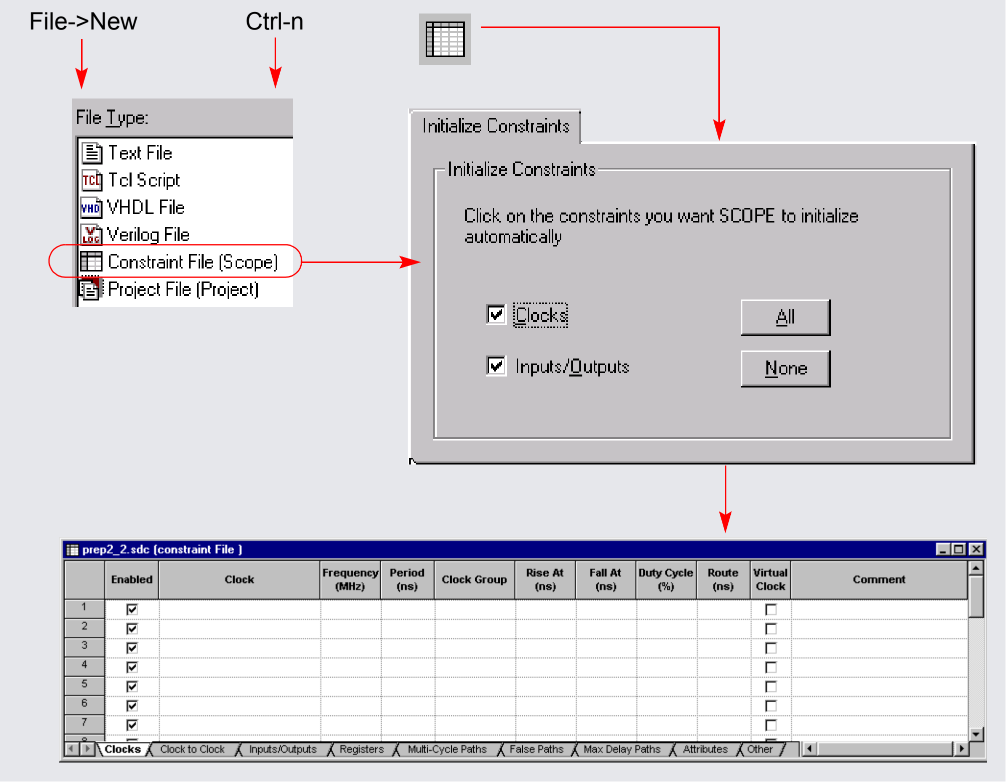
- Add attributes. See *Adding Attributes in the SCOPE Window*, on page 3-68.
- Create collections and apply constraints to them. See *Creating and Using Collections (SCOPE Window)*, on page 3-48.

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 *MultiPoint Synthesis*, on page 10-28.

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.



Click on the constraints you want SCOPE to initialize automatically

☑ Clocks

☑ Inputs/Outputs

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock	Comment
1	☑									□	
2	☑									□	
3	☑									□	
4	☑									□	
5	☑									□	
6	☑									□	
7	☑									□	

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.

You can now enter constraints directly or with the wizard. Refer to *Entering and Editing Constraints in the SCOPE Window*, on page 3-21 or *Entering Default Constraints*, on page 3-24.

2. 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 3-21. If you want to edit the Tcl file directly, see *Working with Constraint Files*, on page 3-60.

Entering and Editing Constraints in the SCOPE Window

For manual constraints, the direct method is best suited for editing and entering individual constraints. If you are setting many constraints or defaults, use the wizard, as described in *Entering Default Constraints*, on page 3-24. You can use the wizard 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 3-45.

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 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 *Creating and Using Collections (SCOPE Window)*, on page 3-48.	Collections
Input/output delays that model your FPGA input/output interface with the outside environment	Inputs/ Outputs
Delay constraints for paths feeding into/out of registers	Registers
Paths that require multiple clock cycles	Multicycle paths
Paths to ignore for timing analysis (false paths)	False Paths
Maximum delay for paths	Max Delay Paths
Attributes, like `syn_reference_clock`, 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 *MultiPoint Synthesis*, on page 10-28 for more information about compile points. (The Synplify tool does not support this flow.)	Compile Points
Place and route tool constraints 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 pulldown list of available choices.

– For object cells in the spreadsheet, click in the cell and select from the pulldown 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:

Prefix	Description
v:	view object (for a module)
i:	instance
p:	port
b:	bit slice of a port
n:	internal net

- For cells with values, type in the value or select from the pulldown 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 *Setting Constraints in the SCOPE Window*, on page 3-18.

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 3-60 for information about the commands in this file.

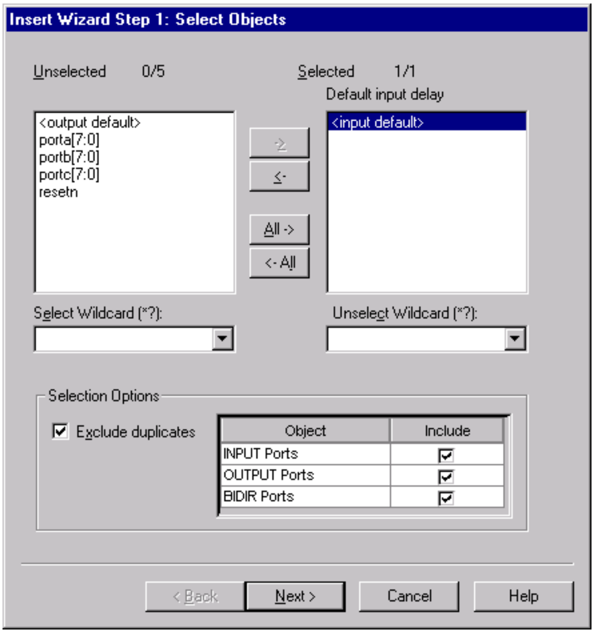
5. To apply the constraints to your design, you must add the file to the project now or later.
   - Add it immediately by clicking Yes in the prompt box that opens after you save the constraint file.
   - Add it later, following the procedure for adding a file described in *Making Changes to a Project*, on page 2-15.

Entering Default Constraints

The wizard is best for entering a number of constraints or for setting defaults manually. To edit or set individual constraints, or create constraints in the Other tab, work directly in the SCOPE window (*Setting Clock and Path Constraints,* on page 3-26). For auto constraints in Synplify Pro, see *Using Auto Constraints,* on page 3-45. The following procedure shows you two methods to enter defaults. The quick method, in step 1, is only appropriate for certain kinds of constraints. The rest of the steps show you how to use the wizard to enter other SCOPE constraints.

1. To quickly generate defaults in the Clocks or Inputs/Outputs tabs without the wizard, follow these steps. This method does not work for other constraints.
   - Click on the Clocks or Inputs/Outputs tabs, and select Edit->Insert Quick. A new row is inserted at the top of the spreadsheet.
   - Select the objects you want from the list.
   - Enter the values you want, and enable the constraint.
   - Save the constraint file and add it to the project.
2. To generate defaults with the wizard, follow the rest of these steps. Select a tab in the SCOPE window and then select Edit->Insert Wizard or press Ctrl-w to start the wizard.

The wizard guides you through two dialog boxes, which vary slightly depending on the kind of constraints you want to set.



Unselected 0/5

<output default>
porta[7:0]
portb[7:0]
portc[7:0]
resetn

Selected 1/1

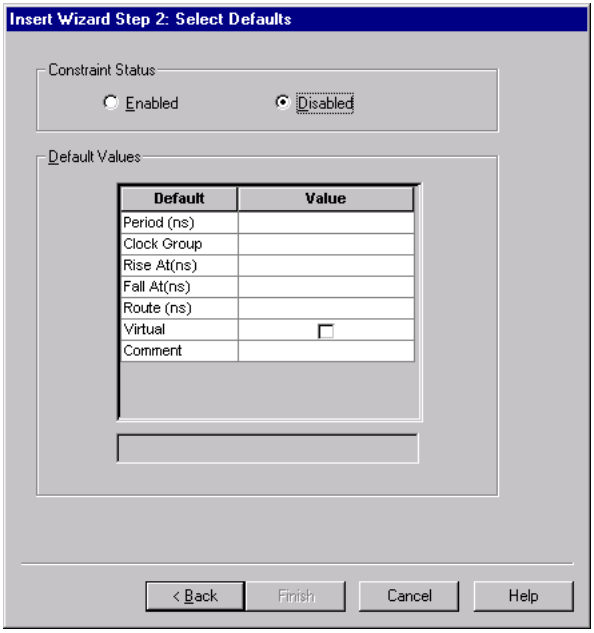
Default input delay

<input default>

☑ Exclude duplicates

Object	Include
INPUT Ports	☑
OUTPUT Ports	☑
BIDIR Ports	☑

3. In the first dialog box, select the design objects to which you want to attach the constraints.
   - Move objects to the selected list by either using wildcards, or highlighting objects in the unselected list and using the arrow buttons to move them. If there are no objects in the Unselected box, disable the Exclude Duplicates option.
   - Click Next.
4. In the second dialog box, set defaults for the selected objects.
   - Enable or disable the constraints.
   - Set the default value.
   - Click Finish.



Constraint Status

○ Enabled ● Disabled

Default Values

Default	Value
Period (ns)	
Clock Group	
Rise At(ns)	
Fall At(ns)	
Route (ns)	
Virtual	□
Comment	

When you are done, the constraints appear in the SCOPE window. To modify or add to them, do so directly in the SCOPE window (refer to *Entering and Editing Constraints in the SCOPE Window*, on page 3-21).

5. To apply the constraints, add the file to the project according to the procedure described in *Making Changes to a Project*, on page 2-15. The constraints file has a `.sdc` extension. See *Working with Constraint Files*, on page 3-60 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 *MultiPoint Synthesis*, on page 10-28 for more information about compile points and *Adding Attributes in the SCOPE Window*, on page 3-68. For information about setting default constraints, see *Entering Default Constraints*, on page 3-24.

To define...	Pane	Do this to set the constraint...
Clocks	Clock	Select the clock (Clock). Type a frequency value (Frequency) or a period (Period). Change the default Duty Cycle or set Rise/Fall At, if needed. Change the default clock group, if needed Check the Enabled box. See *Defining Clocks, on page 3-28* for information about clock attributes.
Virtual clocks	Clock	Set the clock constraints as described for clocks, above. Check the Virtual Clock box.
Route delay	Clock Inputs/ Outputs Registers	Specify the route delay in nanoseconds. Refer to *Defining Clocks, on page 3-28*, *Defining Input and Output Constraints, on page 3-33* and the Register Delays section of this table details.
Edge-to-edge clock delay	Clock to Clock	Select the starting edge for the delay constraint (From Clock Edge). Select the ending edge for the constraint (To Clock Edge). Enter a delay value. Mark the Enabled check box.
Input/output delays	Inputs/ Outputs	See *Defining Input and Output Constraints, on page 3-33* for information about setting I/O constraints.
Register delays	Registers	Select the register (Register). Select the type of delay, input or output (Type). Type a delay value (Value). Check the Enabled box. If you do not meet timing goals after place-and-route, adjust the clock constraint as follows: • 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. • Resynthesize your design.

To define...	Pane	Do this to set the constraint...
Maximum path delay	Max Path Delay	Select the port or register (From/Through). See *Defining From/To/Through for Timing Exceptions*, on page 3-35 for more information. Select another port or register if needed (To/Through). Set the delay value (Max Delay). Check the Enabled box.
Multicycle paths	Multicycle Paths	See *Defining Multicycle Paths*, on page 3-34.
False paths	False Paths Clock to Clock	See *Defining False Paths*, on page 3-38 for details.
Global attributes	Attributes	Set Object Type to <global>. Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.
Attributes	Attributes	Do either of the following: • Select the type of object (Object Type). Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box. • Set the attribute (Attribute) and its value (Value). Select the object (Object). Check the Enabled box.
Other	Other	Type the TCL command for the constraint (Command). Enter the arguments for the command (Arguments). Check the Enabled box.

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 3-45), do not define any clocks. The following procedures show you how to define clock frequency (*Defining Clock Frequency*, on page 3-29) and set other clock constraints that affect timing, like clock groups (*Defining Other Clock Requirements*, on page 3-32).

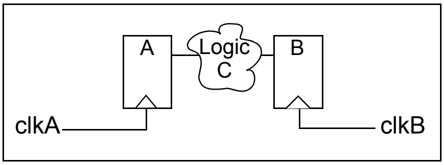
Defining Clock Frequency

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 3-32 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 3-32 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 constrained by a full cycle of clkB.
Defined	Undefined	The path is constrained by a full cycle of clkA.
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 global frequency value is used to constrain path. (Default is 1 MHz or period of 1000 ns.) All global frequency clocks are assigned to the same group.

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.

4. 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 -reg -enable en40}
   syn_reference_clock dummy
```

5. 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 Megawizard 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 3-29), you can also set other clock requirements, as follows:

1. 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.

2. 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. All inferred and other clocks that use the global frequency are in the same clock group.

   – 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.

   The software does not check design rules, so it is best to define the relationship between clocks as completely as possible.

3. 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.

4. Specify edge-to-edge clock delays on the Clock to Clock tab (define_clock_delay).

5. 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 3-26, you can also set the Use clock period for unconstrained IO option.

1. 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.

2. 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.

3. 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, enable 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, disable Use clock period for unconstrained IO.
- Synthesize the design. When you forward-annotate the constraints, the constraints used for synthesis are forward-annotated.

4. 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.

Defining Multicycle Paths

To define a multicycle path constraint, use the Tcl define_multicycle_path command, or select the SCOPE Multicycle tab and do the following;

1. 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 for Timing Exceptions,* on page 3-35 for more information.
2. Select another port or register if needed (From/To/Through).
3. Type the number of clock cycles (Cycles).
4. 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 this, the software uses the end clock period. The constraint is now calculated as follows, where the clock_distance is the shortest distance between the triggering edges of the start and end clocks. The reference_clock_period is either the start clock period or the end clock period, depending on what you specified.

   ```
   multicycle_distance = clock_distance + (m-1) * reference_clock_period
   ```

5. Check the Enabled box.

Defining From/To/Through for Timing Exceptions

For multicycle path, false path, and maximum path delay constraints, you must define paths with a combination of From/To/Through points. The following steps guide you through the details. You must specify at least one From, To, or Through point.

1. In the From field, identify the starting point for the path. Use the appropriate prefix for the object.

   The starting point can be a register, top-level input or bidirectional port, or black box. To specify multiple starting points, like all the bits of a bus, enclose them in square brackets: A[0:15].

   Where there are multiple constraints, the software uses the following priority constraints:

   - Bit constraints override bus constraints. Given a constraint From A[0:15] to B, and a second From A[8] to B, only the second constraint applies to paths starting from A[8]. The first constraint applies to paths starting from A[0:7, 9:15].
   - If the previous rule does not apply and there are multiple constraints, the tightest constraint prevails. If there is a 3-cycle constraint From A To B, and a 4-cycle constraint From A To B Through C, all paths starting at A and ending at B (including paths that cross C) get a 3-cycle constraint.
   - If you specify multiple start points and multiple end points such as From A[0:15] to B[0:15], the constraint applies from any start point to any end point. In this example, the exception applies to all 16 * 16 = 256 combinations of start/end points.

2. In the To field, identify the ending point for the path. Use the appropriate prefix for the object.

   The ending point can be a register, top-level output or bidirectional port, or black box. To specify multiple ending points, like all the bits of a bus, enclose them in square brackets: B[0:15]. In the case of multiple constraints, the priority rules described in the previous step apply.

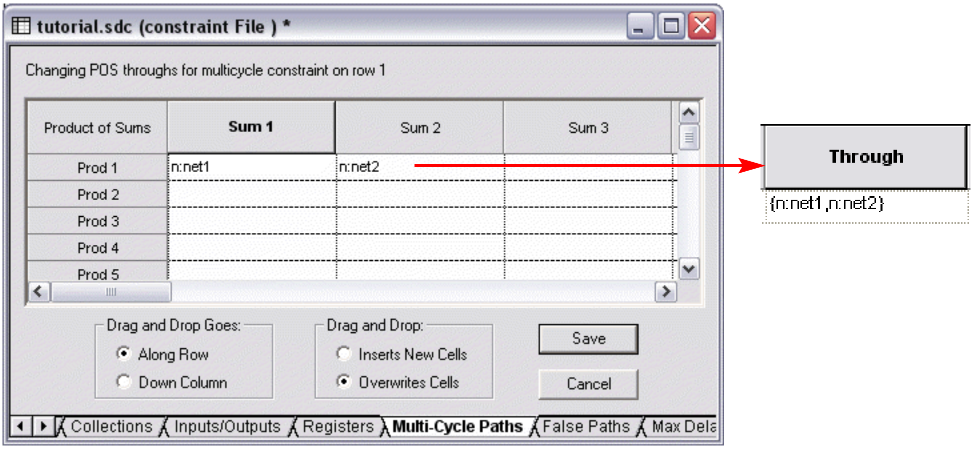
3. For a single through point, you can either type in the net name (prefixed by n:) or follow the steps below. The through point must be a net.

   - 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 a HDL Analyst view into the cell.
- Click Save.

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

4. To specify an OR when constraining a list of through points, you can type the net names in the Through field (see the following figure}. Alternatively, do the following
   - 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 a HDL Analyst view into the cell. Repeat this step along the same row, adding other nets in the Sum columns. The nets in each row form an OR list.



Changing POS throughs for multicycle constraint on row 1

Product of Sums	Sum 1	Sum 2	Sum 3
Prod 1	n:net1	n:net2	
Prod 2			
Prod 3			
Prod 4			
Prod 5			

Drag and Drop Goes:
● Along Row
○ Down Column

Drag and Drop:
○ Inserts New Cells
● Overwrites Cells

Through
{n:net1,n:net2}

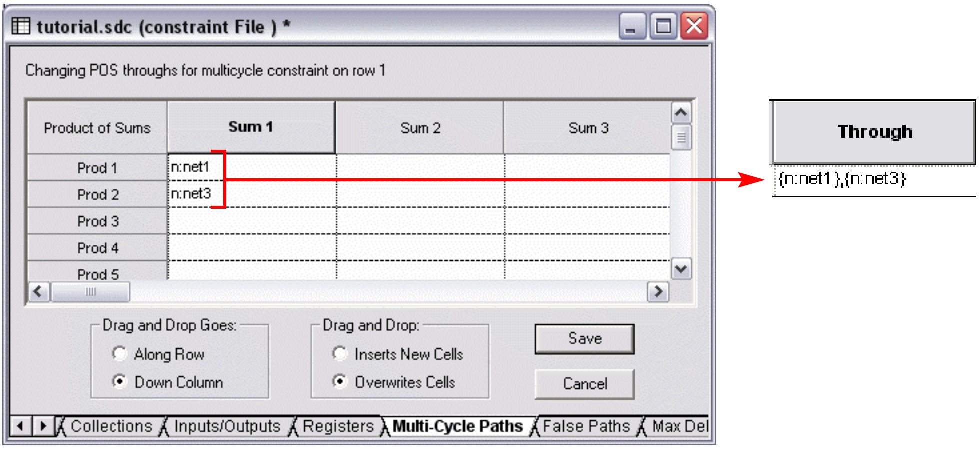
   - Alternatively, select Along Row in the SCOPE POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names along the row. The nets in each row form an OR list.
   - Click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

5. To specify an AND when constraining a list of through points, type the names in the Through field (see the following figure) or do the following:

   – Open the Product of Sums interface (see previous step).

   – Either type the first net name in the first cell in a Sum column or drag the net from a HDL Analyst view into the cell. Repeat this step down the same Sum column.



Changing POS throughs for multicycle constraint on row 1

Product of Sums	Sum 1	Sum 2	Sum 3
Prod 1	n:net1		
Prod 2	n:net3		
Prod 3			
Prod 4			
Prod 5			

Drag and Drop Goes:
○ Along Row
● Down Column

Drag and Drop:
○ Inserts New Cells
● Overwrites Cells

Through
{n:net1 },{n:net3}

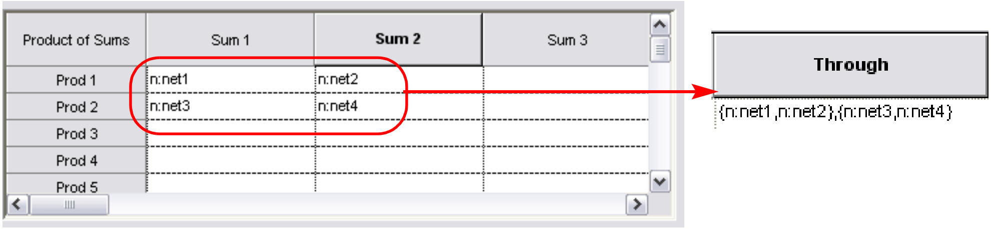
   – Alternatively, select Down Column in the SCOPE POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names down the column.

   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.

6. 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:

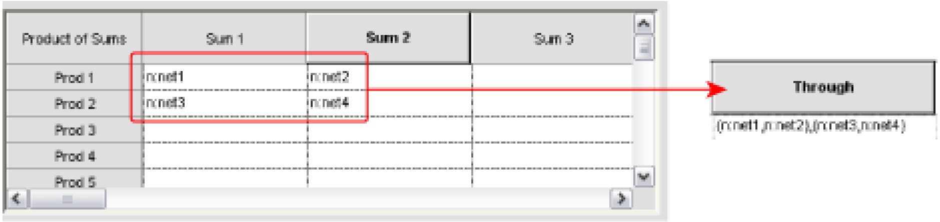
   – Create multiple lists as described in the previous 2 steps.

   – Click Save.



Product of Sums	Sum 1	Sum 2	Sum 3
Prod 1	n:net1	n:net2	
Prod 2	n:net3	n:net4	
Prod 3			
Prod 4			
Prod 5			

Through
{n:net1,n:net2},{n:net3,n:net4}



Product of Sums	Sum 1	Sum 2	Sum 3
Prod 1	n:net1	n:net2	
Prod 2	n:net3	n:net4	
Prod 3			
Prod 4			
Prod 5			

Through
{n:net1,n:net2},{n:net3,n:net4}

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 False Paths

You define false paths by setting constraints explicitly on the False 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.

1. To define a false path between ports or registers, select the SCOPE False Paths tab, and do the following:
   - Select the port or register (From/To/Through). See *Defining From/To/Through for Timing Exceptions, on page 3-35* for more information.
   - Select another port or register if needed (From/ To/Through).
   - Check the Enabled box.

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

2. 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).See *Defining From/To/Through for Timing Exceptions*, on page 3-35 for more information.

   – 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 whether their clock groups. This constraint can be overridden by a maximum delay constraint on the same path.

3. 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 as described in the next step.

4. To override an implicit false path between any two clocks (see the previous step), 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) as described in step 2.

   – Specify a value in the Delay column.

   – Mark the Enabled check box.

   The software treats this is 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.

5. To set an implicit false path on a path to/from an I/O port, select Project->Implementation Options->Constraints, and disable Use clock period for unconstrained IO.

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 7-40, *Altera I/O Standards,* on page 7-40, and *Xilinx I/O Standards,* on page 7-42.
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_padtype attribute. When you synthesize the design, the I/O specifications are mapped to the appropriate I/O pads within the technology.

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	With a SCOPE window open, select View-> Properties. Set the options you want on the Display Settings form. Check the Save settings to profile option if you want to settings to be the default.
Set fonts, colors, and borders for a row	Select a SCOPE row. Select Format -> Style. On the Styles form, check Save as Default if you want the new settings to be the default. Select the category you want to change (Row Header or Standard), and click Change. Set the display options you want and click OK on both forms.
Set fonts, colors, and borders for a column	Select a SCOPE row. Select Format -> Style. On the Styles form, check Save as Default if you want the new settings to be the default. Select the category you want to change (Column Header or Standard), and click Change. Set the display options you want and click OK on both forms.
Set fonts, colors, and borders for a single cell	Select a SCOPE cell. Select Format -> Cells. Set the display options you want and click OK.
Align text in columns and rows	Select a column or row in the SCOPE window. Select Format -> Align. Click the alignment you want and click OK.
Size columns/rows to text	Select a column or row in the SCOPE window. Select Format -> Resize Rows or Format -> Resize Columns.
Hide/show cells	Select a SCOPE cell. Select Format -> Cover Cells to hide a cell. Select Format -> Remove Covering to show a hidden cell.

Translating Xilinx UCF Constraints

If you have a Xilinx UCF file with timing constraints or pad constraints, you can translate these constraints to the sdc format and use them to drive synthesis.

1. Make sure the UCF file has a .ucf extension.
2. From the command line, run the translator on the UCF file.

   The translator is in the bin directory: *<install_dir>*/bin/edf2srs.exe. Use the following syntax:

   ```
   edf2srs -osyn <sdc_file> –ucf <ucf_file>
   ```

   The translator generates a constraint file in the .sdc format, which contains the timing-related constraints from the UCF file that are relevant to synthesis. It ignores the other backend constraints in the UCF file. The following table shows the UCF constraints that can be 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	

3. Use the generated .sdc file to drive synthesis.

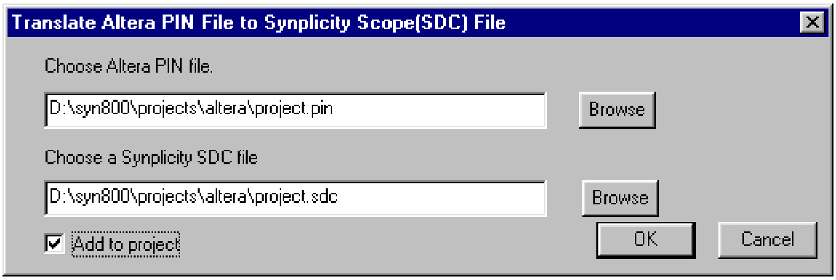
Converting Pin Location Constraint Files in the Synplify Premier Tool

You can automatically convert place-and-route pin loc constraint files to SCOPE constraint files (.sdc) using Run->Translate Constraints. The following subsections provide information on how to translate these files for Altera and Xilinx, respectively.

Altera PIN File

To translate a .pin file to an .sdc file:

1. Select Run ->Translate Constraints.
2. Enter the .pin file you want to translate and the name of the .sdc pin loc constraint file you wish to create.



Translate Altera PIN File to Synplicity Scope(SDC) File

Choose Altera PIN file.

D:\syn800\projects\altera\project.pin

Choose a Synplicity SDC file

D:\syn800\projects\altera\project.sdc

☑ Add to project

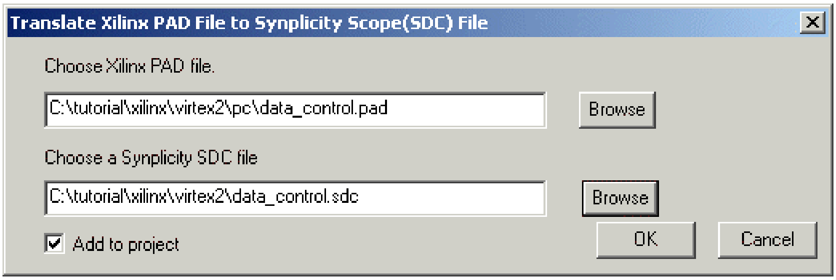
Figure 3-1: .pin File to .sdc File

3. Click on Add to Project, as appropriate, then click OK.

Xilinx PAD File

To translate a .pad file to a .sdc file:

1. Select Run ->Translate Constraints.
2. Enter the .pad file you want to translate and the name of the .sdc pin loc constraint file you wish to create.



Translate Xilinx PAD File to Synplicity Scope(SDC) File

Choose Xilinx PAD file.

C:\tutorial\xilinx\virtex2\pc\data_control.pad

Choose a Synplicity SDC file

C:\tutorial\xilinx\virtex2\data_control.sdc

☑ Add to project

Figure 3-2: .pad File to .sdc File

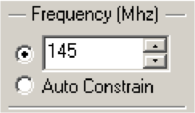
3. Click on Add to Project, as appropriate, then click OK.

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 7-4 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.

   Frequency (Mhz)

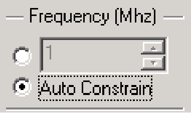
   ● 145

   ○ Auto Constrain

   – 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 multicycle or false path constraints are specified on registers.

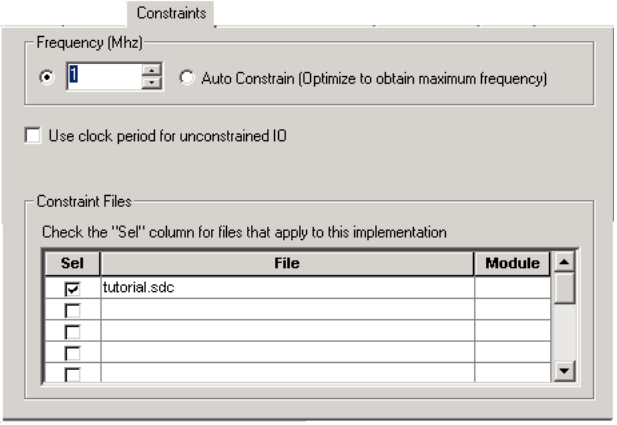
2. Enable the Auto Constrain button on the left side of the Project view. Alternatively, select Project->Implementation Options->Constraints, and enable the Auto Constrain option there.



Frequency (Mhz)

○ 1

● Auto Constrain



Frequency (Mhz)

● 1 ○ Auto Constrain (Optimize to obtain maximum frequency)

□ Use clock period for unconstrained IO

Constraint Files

Check the "Sel" column for files that apply to this implementation

Sel	File	Module
☑	tutorial.sdc	
□		
□		
□		
□		

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.

Using Collections

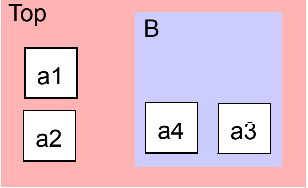
A collection is a group of objects. It can consist of just one object, or of other collections. You can set the same constraint for multiple objects if you group them together in a collection. You can either define collections in the SCOPE window or type the commands in the Tcl script window. The Synplify tool does not support collections.

- Comparing Methods for Defining Collections, next
- Creating and Using Collections (SCOPE Window), on page 3-48
- Creating Collections (Tcl Commands), on page 3-51
- Using the Tcl Find Command to Define Collections, on page 3-53
- Using the Expand Tcl Command to Define Collections, on page 3-55
- Viewing and Manipulating Collections (Tcl Commands), on page 3-56

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.



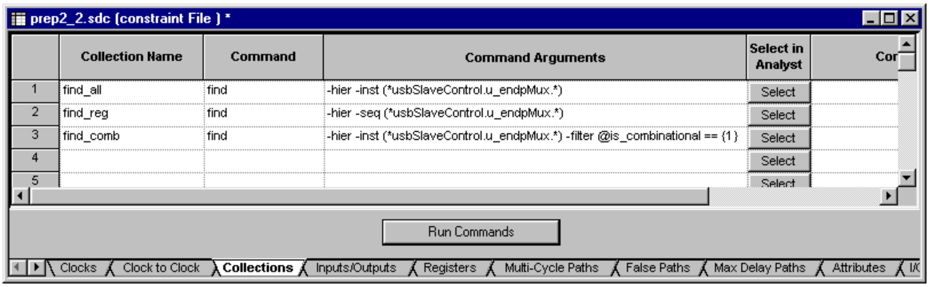
Similarly, the current Analyst view could be a lower-level view. In the design shown above, if you push down into B, and then type `find -hier a*` in the Tcl window, the command finds a3 and a4. However if you cut and paste the same command into the SCOPE Collections tab, your results would include a1, 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 3-51). See *Comparing Methods for Defining Collections*, on page 3-47 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 3-51.



	Collection Name	Command	Command Arguments	Select in Analyst	Cor
1	find_all	find	-hier -inst (*usbSlaveControl.u_endpMux.*)	Select	
2	find_reg	find	-hier -seq (*usbSlaveControl.u_endpMux.*)	Select	
3	find_comb	find	-hier -inst (*usbSlaveControl.u_endpMux.*) -filter @is_combinational == {1}	Select	
4				Select	
5				Select	

   - 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 3-53 and *Using the Expand Tcl Command to Define Collections*, on page 3-55. 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 3-47.

- 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.

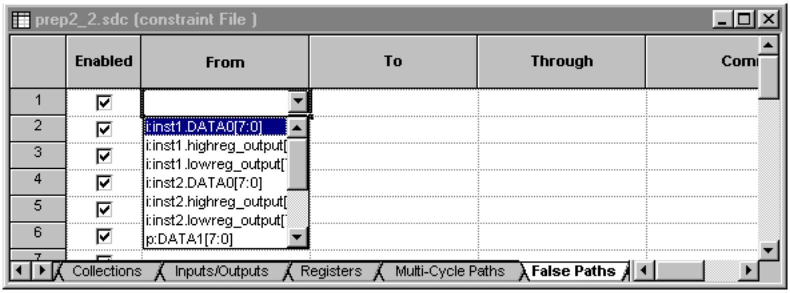
- To crossprobe the objects selected by the find and expand commands, click Select in the Select in Analyst column. The schematic views highlight the objects located by these commands. For other viewing operations, see *Viewing and Manipulating Collections (Tcl Commands)*, on page 3-56.

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 3-51 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.



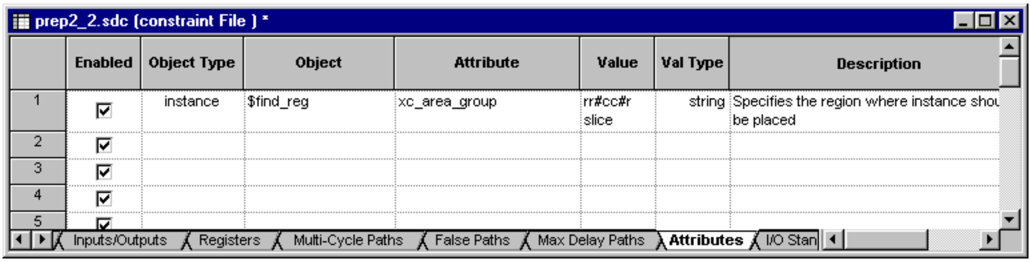
prep2_2.sdc (constraint File)

	Enabled	From	To	Through	Com
1	☑				
2	☑				
3	☑				
4	☑				
5	☑				
6	☑				

   - 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 3-50.

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 .srr file, the netlist, and if you are using Synplify Premier, the Design Planner view to see that the attribute is honored.



prep2_2.sdc (constraint File) *

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	instance	$find_reg	xc_area_group	rr#cc#r slice	string	Specifies the region where instance shou be placed
2	☑						
3	☑						
4	☑						
5	☑						

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 3-48). 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 3-47.

For details of the syntax for the commands described here, refer to *Tcl Collection Commands*, on page 5-39 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 3-53 and *Using the Expand Tcl Command to Define Collections*, on page 3-55). 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 *Examples: c_union Command*, on page 3-52
Concatenate collections	c_union. See *Examples: c_union Command*, on page 3-52.
Create a collection from the differences between collections	c_diff. See *Examples: c_diff Command*, on page 3-52.
Create a collection from common objects in collections	c_intersect. See *Examples: c_intersect Command*, on page 3-52.
Find objects that belong to just one collection	c_symdiff. See *Examples: c_symdiff Command*, on page 3-53.

You can now do various operations on the objects in the collection (see *Viewing and Manipulating Collections (Tcl Commands)*, on page 3-56), 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 collection1and 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_symdiff 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_symdiff collection1 {i:reg1 i:reg3}]
c_list $diffCollection
   {"i:reg2" "i:reg3"}
```

You can also use the command to compare two collections:

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

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 3-47.

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 3-55). This section lists some tips for using the Tcl find command.

1. Tcl find always searches at the top-level of your design, irregardless 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 3-55.

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 5-48 of the *Reference Manual*.

Case rules	Use the case rules for the language from which the object was generated: • VHDL: case-insensitive • Verilog: case-sensitive. Make sure that the object name you type in the SCOPE window matches the Verilog name. 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: `find {a*} -nocase`
Pattern matching	You have two choices: • Specify the -regexp argument, and then use regular expressions for pattern matching. • Do not specify -regexp, and use only the * and ? wildcards for pattern matching.
Restricting search by type of object	Use the -*object_type* argument. The following command finds all nets that contain syn. `find -net {*syn*}`
Restricting search to hierarchical levels below the current view	Use the -hier argument. The following example finds all objects below the current view that begin with a: `find {a*} -hier`

Restricting search by object property	• Select Project->Implementation Options. On the Options tab, enable Annotated Properties for Analyst.
	• 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.
	• Use the -filter argument to the find command. The following example finds any register in the current view that is clocked by myclk.
	`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 3-56:
   - 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 3-56.

Examples: Useful Find Commands

To find...	Use a command like this example...
Instances by slack value	set result [find –hier –inst {*} –filter @slack <= {-1.000}]
Instance within a slack range	set result [find –hier –inst {*} –filter @slack <= {-1.000} && @slack >= {+1.000}]
Pins by fanin/fanout value	set result [find –hier –inst {*.D} –filter @fanin <= {50}]
Sequential components by primitive type	set result [find –hier –seq {*} –filter @view=={ FDRSE}

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 5-45 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 3-51).

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 5-39 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 3-59.

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 Options 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 5-39 in the *Reference Manual*.

To...	**Do this...**
Copy a collection	Create a new variable for the copy and copy the original collection to it with the set command. When you make changes to the original, it does not affect the copy, and vice versa. **set** my_collection_copy $my_collection
List the objects in a collection	Use the c_print command to view the objects in a collection, and optionally their properties, in column format: “v:top” “v:block_a” “v:block_b” Alternatively, you can use the -print option to an operation command to list the objects.
Generate a Tcl list of the objects in a collection	Use the c_list 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. This is an example of c_list results: {“v:top” “v:block_a” “v:block_b”} Alternatively, you can use the -print option to an operation command to list the objects.
Iterate through a collection	Use the c_foreach command. This example iterates through all the objects in the collection: **c_foreach** port [**find -port** *] { **define_false_path -from** $port

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]
```

Working with Constraint Files

Constraint files are text files that are automatically generated by the SCOPE interface (see *Setting Constraints in the SCOPE Window*, on page 3-18), 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, next
- Tcl Syntax Guidelines for Constraint Files, on page 3-61
- Using a Text Editor for Constraint Files, on page 3-62
- Generating Constraint Files for Forward Annotation, on page 3-64

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 *Adding Attributes and Directives*, on page 3-66 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:

- Pay attention to case, because Tcl is case-sensitive.
- Remember these rules when naming objects:
  - Make sure that object names match the names in the HDL code.
  - Enclose all instance and port names with curly braces {}.
  - Do not use spaces in names.
  - Use periods as separators in hierarchical names
  - In Verilog modules, use the following syntax for instance, port, and net names, where *cell* is the name of the design entity, *prefix* is a prefix to identify objects with the same name, and *object_name* is an instance path with periods as separators:

    v:*cell[prefix***:***]object_name*

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

  - Use the following syntax for instance, port, and net names in VHDL modules, 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 periods as separators. You only need view if there is more than one architecture for the design. See the preceding table for the prefixes for different objects.

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

- Use the * and ? wildcards to match names. The asterisk matches any number of characters, and the question mark matches a single character. These characters do not match periods that are used as hierarchy separators. For example, you can use the following to identify all bits of the statereg instance in the statemod module:

```
statemod | i: 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 3-60.

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 3-61.
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 3-28 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 3-28 for additional information
Clock domains with asymmetric duty cycles	`define_clock`. See *Defining Clocks,* on page 3-28 for additional information
Edge-to-edge clock delays	`define_clock_delay`. See *Defining Clocks,* on page 3-28 for additional information

To define...	Use...
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 3-33 for additional information
Output delays from your FPGA	`define_output_delay`. See *Defining Input and Output Constraints*, on page 3-33 for additional information
Paths with multiple clock cycles	`define_multicycle_path`. See *Defining From/To/Through for Timing Exceptions*, on page 3-35 for additional information
False paths (certain technologies)	`define_false_path`. See *Defining False Paths*, on page 3-38 for additional information
Path delays	`define_path_delay`. See *Defining From/To/Through for Timing Exceptions*, on page 3-35 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

# Get better results on the critical path going to register
# "inst3.q[0]" (in the memory) by adding 3 ns with -improve
   define_reg_input_delay {inst3.q[0]} -improve 3.0
```

4. You can also add vendor-specific attributes in the constraint file using define_attribute. See *Adding Attributes to a Tcl Constraint File*, on page 3-73 for more information.

5. Save the file.

6. Add the file to the project as described in *Making Changes to a Project*, on page 2-15, and run synthesis.

Generating Constraint Files for Forward Annotation

You can create certain vendor-specific constraint files, where the synthesis constraints are mapped to the appropriate vendor constraints.

1. Set attributes to control forward annotation.

   – To forward-annotate timing constraints for Actel eX, 54sxa, Axcelerator, 500K, APA, and ProASIC3/3E 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 `.ncf` 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 3-32 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 3-32 for details. The synthesis software forward-annotates the PLL inputs.

   – For some Lattice designs, set the `-from` and `-to` false path and multicycle 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, define_multicycle_path, define_false_path, define_max_delay, and global frequency constraints into corresponding commands in the *.acf file for Altera, the *.lp file for Lattice, the *filename*_sdc.sdc file for Actel, and the *.ncf file for Xilinx. See the *Reference Manual* for details about forward annotation.

4. For Lattice Orca designs, you must copy the constraints into the .prf file.
   - Open the ispLEVER place-and-route tool, and run the Map stage. This creates a .prf file.
   - Copy the .lp file created by the Synplicity software and paste it at the end of the .prf file. Do not overwrite the .prf file.

Adding 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. Attributes control mapping optimizations and directives control compiler optimizations. Because of this difference, you must specify directives in the source code. Attributes can be added from the SCOPE and HDL Analyst windows as well as in the source code. For further details, refer to these subtopics:

- Adding Attributes and Directives in VHDL, next
- Adding Attributes and Directives in Verilog, on page 3-68
- Adding Attributes in the SCOPE Window, on page 3-68
- Adding Attributes with the SCOPE Wizard, on page 3-71
- Adding Attributes to a Tcl Constraint File, on page 3-73
- Adding Attributes From the RTL and Technology Views, on page 3-74

Adding Attributes and Directives in VHDL

You can use other methods to add attributes to objects, as listed in *Adding Attributes and Directives*, on page 3-66. However, you can specify directives only in the Verilog or VHDL 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

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 <att_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;
   ```

   Fordetails of the syntax conventions, see *VHDL Attribute and Directive Syntax*, on page 10-91 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 as follows:

   ```
      <design_unit_declaration>;
      attribute <att_name> : <data_type>;
      attribute <att_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.

Adding Attributes and Directives in Verilog

You can use other methods to add attributes to objects, as described in *Adding Attributes and Directives, on page 3-66*. However, you can specify directives only in the Verilog or VHDL source code.

Verilog does not have predefined synthesis attributes and directives, so you must add them as comments. Note that the attribute or directive name is preceded by the keyword synthesis.

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	Verilog Line Comment Syntax
/* synthesis *<att_name>* = *<value>* */	// synthesis *<att_name>* = *<value>*
/* synthesis *<dir_name>* = *<value>* */	// synthesis *<dir_name>* = *<value>*

For details of the synatx rules, see *Verilog Attribute and Directive Syntax, on page 9-78* 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 */;
```

Adding Attributes in the SCOPE Window

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 *Adding Attributes and Directives in VHDL, on page 3-66* or *Adding Attributes and Directives in Verilog, on page 3-68*).

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.

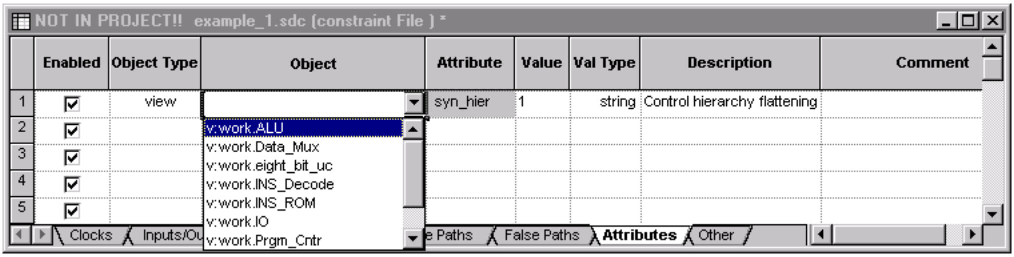
   – Drag the object to which you want to attach the attribute from the RTL or Technology views to the Object column in the SCOPE window.

   – Select the type of object in the Object Filter column, and then select an object from the list of choices in the Object column.

   – 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.

   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.

4. Specify the attribute by holding down the mouse button in the Attribute column and selecting an attribute from the list. To specify a group of attributes, use the wizard, as described in *Adding Attributes with the SCOPE Wizard*, on page 3-71.



	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	☑	view		syn_hier	1	string	Control hierarchy flattening	
2	☑							
3	☑							
4	☑							
5	☑							

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.

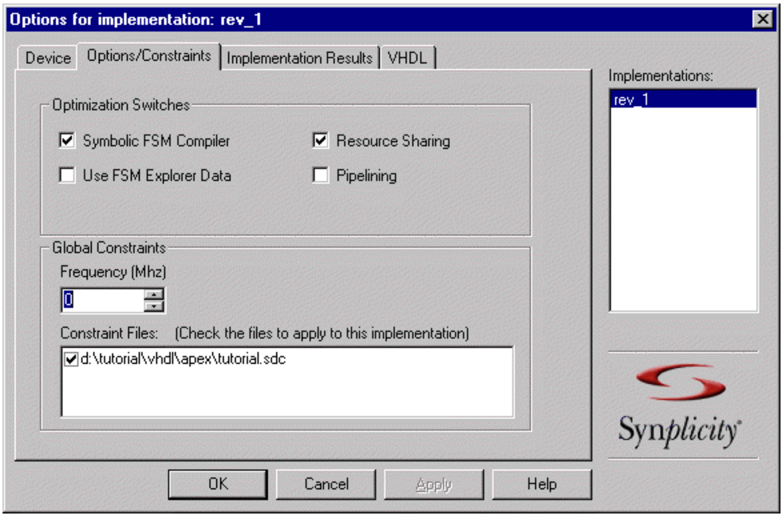
   If you manually type an attribute the software does not recognize, or select an incompatible attribute/object combination, the attribute cell is shaded in red.

6. Save the file.

7. Add it to the project, if it is not already in the project.

   – Choose Project -> Implementation Options.

   – Go to the Options/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.



Options for implementation: rev_1

☑ Symbolic FSM Compiler ☑ Resource Sharing
□ Use FSM Explorer Data □ Pipelining

Frequency (Mhz) 0

Constraint Files: (Check the files to apply to this implementation)

☑ d:\tutorial\vhdl\apex\tutorial.sdc

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.

Adding Attributes with the SCOPE Wizard

The SCOPE attribute wizard provides an easy interface for assigning attributes to design objects. It is most useful for assigning (or unassigning) the same attribute value to *several* objects at once. It provides a convenient way to enable, disable, and re-enable multiple identical attribute assignments. One common use is to assigning default attribute values. Use the following procedure.

1. Right-click in the SCOPE spreadsheet Attributes panel, then choose Insert Wizard from the popup menu (or from the Edit menubar menu).

   A dialog box opens.

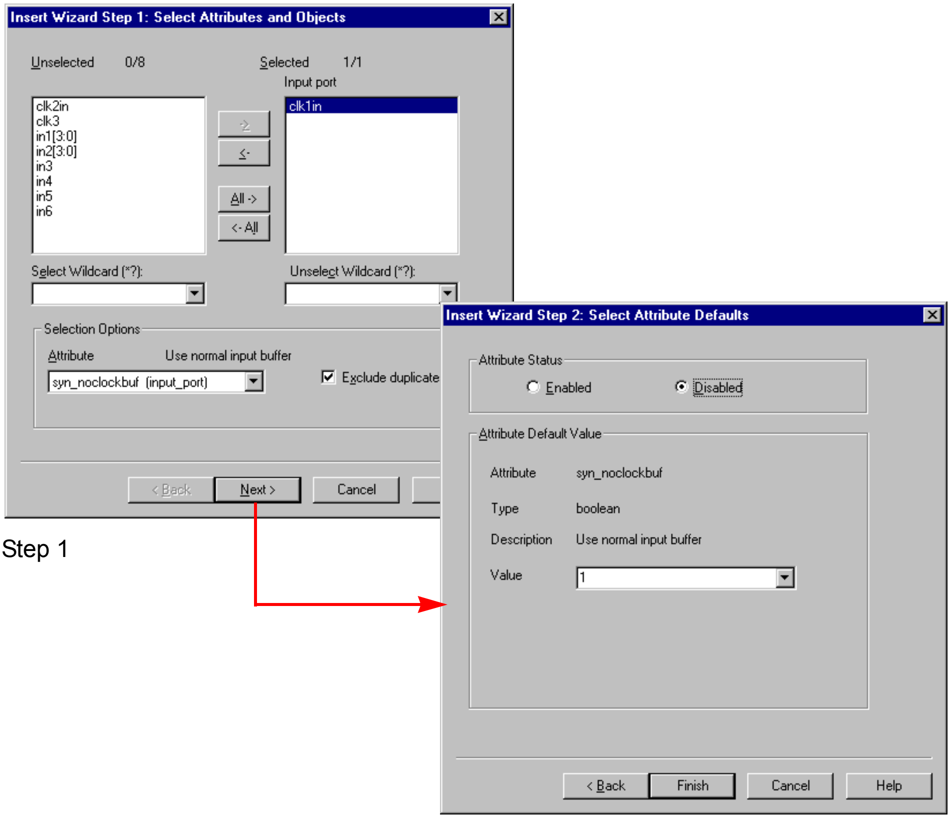
2. Choose an attribute from the Selection Options pulldown list. The selected attribute determines the object choices available in the Unselected list. (If you do not see a list, disable Exclude Duplicates.)

3. Select the objects you want.

   – Select the objects, either with the mouse in the Unselected box, or by using wildcards in the Select Wildcards (*?) field. The objects are highlighted in the Unselected box.

   – Move them to the Selected list by clicking the right arrow (->). Alternatively, select an object and double-click to move it to the Selected list.

   – Click Next to go to the second dialog box.



Insert Wizard Step 1: Select Attributes and Objects

Unselected 0/8

clk2in
clk3
in1[3:0]
in2[3:0]
in3
in4
in5
in6

Selected 1/1

Input port

clk1in

Select Wildcard (*?):

Unselect Wildcard (*?):

Selection Options

Attribute: syn_noclockbuf (input_port) — Use normal input buffer

☑ Exclude duplicate

Insert Wizard Step 2: Select Attribute Defaults

Attribute Status: ○ Enabled ● Disabled

Attribute Default Value

Attribute	syn_noclockbuf
Type	boolean
Description	Use normal input buffer
Value	1

Step 1

Step 2

4. In the second dialog box, do the following:

   – Set the value of the attribute, which applies to *all* the selected objects.

   – Enable the attribute to apply the value; disable it to remove the attribute.

   – Click Finish. The attribute is set in the SCOPE spreadsheet and saved in the constraint file. The SCOPE spreadsheet reflects your choices.

Adding Attributes to a Tcl Constraint File

When you add attributes through the SCOPE window (*Adding Attributes in the SCOPE Window*, on page 3-68 and Adding Attributes with the SCOPE Wizard, on page 3-71), the attributes are automatically added to the constraint file using the Tcl define_attribute syntax. The following procedure explains how to add attributes manually to a Tcl constraint file. For information about editing the constraints in a constraint file, see *Using a Text Editor for Constraint Files*, on page 3-62.

1. In the constraint file, add the attribute and value you want, using the following define_attribute syntax.

   define_attribute {*object_name*} *attribute_name value*

   Check the descriptions of individual attributes in the *Reference Manual* for the exact values and syntax of the attribute.

The following code excerpt shows some attributes set in a constraint file. Some of the attributes are specific to Xilinx designs:

```
# 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
```

Adding Attributes From the RTL and Technology Views

You can add attributes to instances, nets, or ports in the RTL or Technology windows.

1. If you already have a constraint file but you want to use a new one for the attributes, create a file first, and add it to the project. If you are using an existing constraint file, go to the next step.

2. Select an instance, net, or port in an RTL or Technology view.

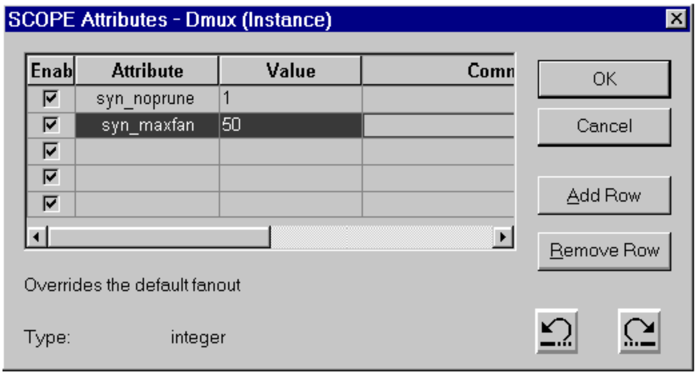
   You can only select a single object. The instance must be a primitive or a module.

3. Right-click and select SCOPE->Edit Attributes from the popup menu.

   If the command is grayed out, you have selected an invalid object.

   If you do not have a constraint file, the software asks you if you want to create one. If you select OK, the software automatically creates a constraint file and adds it to the project file. If you have a constraint file, the software opens and minimizes it. If there are multiple constraint files, you are prompted to choose one from a list.

   Then, an attribute editing dialog box opens.



Enab	Attribute	Value	Comn
☑	syn_noprune	1	
☑	syn_maxfan	50	
☑			
☑			
☑			

Overrides the default fanout

Type: integer

4. Specify the attribute and the value in the box. The bottom left of the form shows a short description of the selected attribute and lists the type of value required.

5. Click OK.

   The software writes the attribute to the constraint file.

CHAPTER 4

Result Analysis

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:

For information about using the Synplify Premier Physical Analyst, see Chapter 5, *Physical Analyst*.

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, next
- Analyzing Results Using the Log File Reports, on page 4-5
- Using the Log Watch Window, on page 4-6

Viewing the Log File

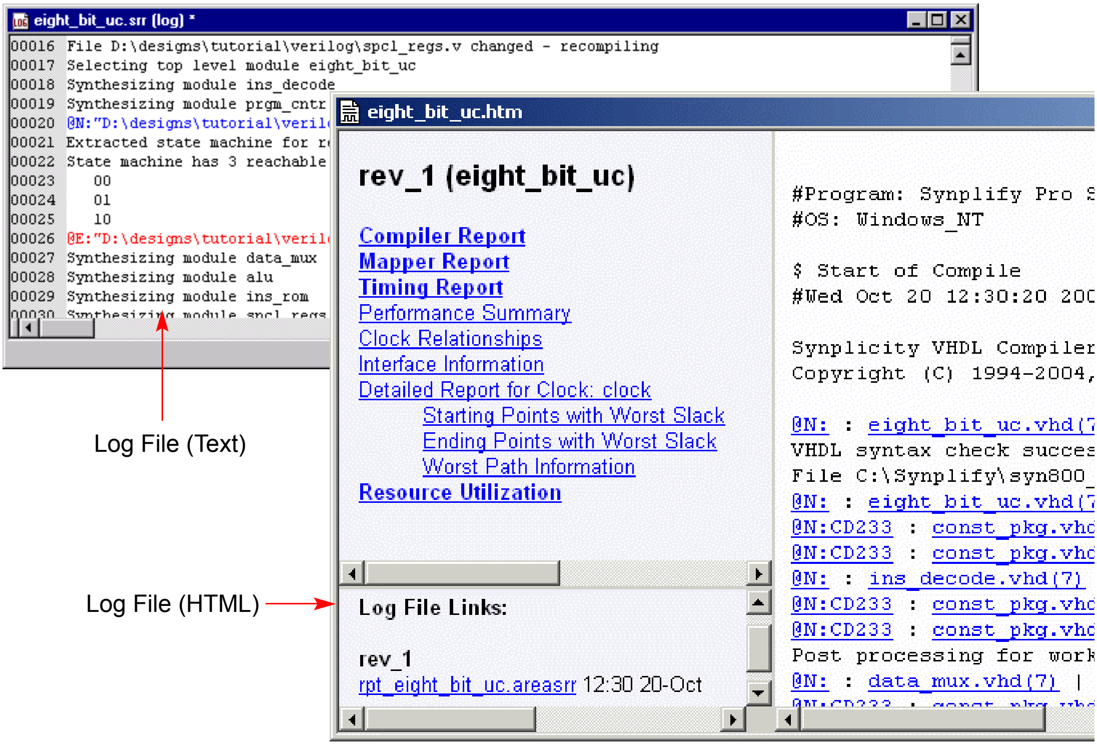
The log file contains the most comprehensive results and information about a synthesis run. The default log file is in HTML format, but there is a text version available too.

For Synplify Pro or Synplify Premier users who just 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 4-6) 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 4-5*.



```
File D:\designs\tutorial\verilog\spcl_regs.v changed - recompiling
Selecting top level module eight_bit_uc
Synthesizing module ins_decode
Synthesizing module prgm_cntr
@N:"D:\designs\tutorial\verilo
Extracted state machine for r
State machine has 3 reachable
   00
   01
   10
@E:"D:\designs\tutorial\verilo
Synthesizing module data_mux
Synthesizing module alu
Synthesizing module ins_rom
Synthesizing module spcl_regs
```

rev_1 (eight_bit_uc)

Compiler Report
Mapper Report
Timing Report
Performance Summary
Clock Relationships
Interface Information
Detailed Report for Clock: clock
    Starting Points with Worst Slack
    Ending Points with Worst Slack
    Worst Path Information
Resource Utilization

Log File Links:

rev_1
rpt_eight_bit_uc.areasrr 12:30 20-Oct

```
#Program: Synplify Pro S
#OS: Windows_NT

$ Start of Compile
#Wed Oct 20 12:30:20 200

Synplicity VHDL Compiler
Copyright (C) 1994-2004,

@N: : eight_bit_uc.vhd(7
VHDL syntax check succes
File C:\Synplify\syn800_
@N: : eight_bit_uc.vhd(7
@N:CD233 : const_pkg.vhd
@N:CD233 : const_pkg.vhd
@N: : ins_decode.vhd(7)
@N:CD233 : const_pkg.vhd
@N:CD233 : const_pkg.vhd
Post processing for work
@N: : data_mux.vhd(7) |
@N:CD233 : const_pkg.vhd
```

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 2-5*.

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 *Analyzing Paths with the Timing Analyst,* on page 4-76. 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 4-8 for information. Notes are informational and usually can be ignored. For details about crossprobing and fixing errors, see *Handling Warnings,* on page 4-14, *Editing HDL Source Files with the Built-in Text Editor,* on page 2-5, and *Crossprobing from the Text Editor Window,* on page 4-51.

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 2-5.

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,
   - 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 4-81 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 *Analyzing Paths with the Timing Analyst*, on page 4-76. For information about the Synplify Premier island-based timing report, see *Basic Operations in the Schematic Views*, on page 4-16.

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 4-2.

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

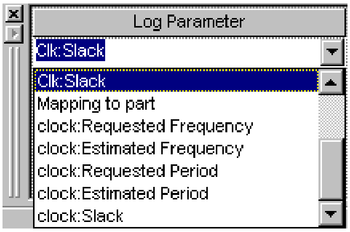
   If you open an existing project, the Log Watch window shows the parameters set the last time you opened the window.

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 2-6 in the *Reference Manual* for information about the popup menu commands.

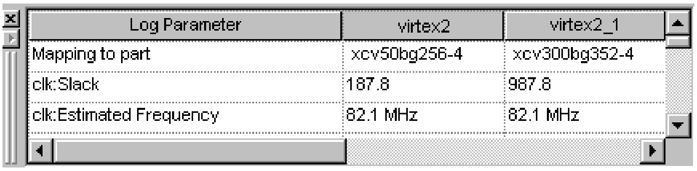
3. Select the log parameter you want to monitor by clicking on a line and selecting a parameter from the resulting popup menu.



   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 *Design Guidelines*, on page 6-2.



Log Parameter	virtex2	virtex2_1
Mapping to part	xcv50bg256-4	xcv300bg352-4
clk:Slack	187.8	987.8
clk:Estimated Frequency	82.1 MHz	82.1 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:

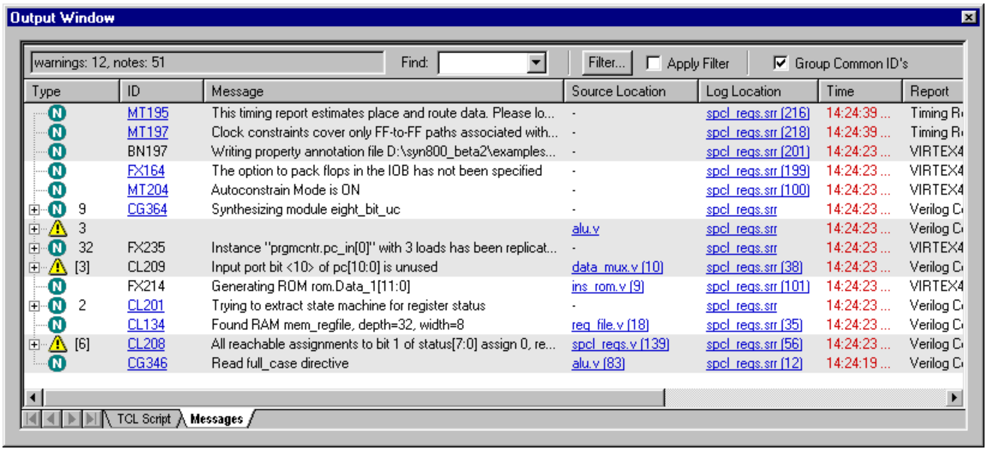
- Checking Results in the Message Viewer, next
- Filtering Messages in the Message Viewer, on page 4-10
- Filtering Messages from the Command Line, on page 4-13
- Handling Warnings, on page 4-14
- Automating Message Filtering with a synhooks Script, on page 4-14

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 2-11 in the *Reference Manual* for a full description of the window.



Output Window

warnings: 12, notes: 51 Find: Filter... □ Apply Filter ☑ Group Common ID's

Type	ID	Message	Source Location	Log Location	Time	Report
N	MT195	This timing report estimates place and route data. Please lo...	-	spcl_regs.srr (216)	14:24:39 ...	Timing R
N	MT197	Clock constraints cover only FF-to-FF paths associated with...	-	spcl_regs.srr (218)	14:24:39 ...	Timing R
N	BN197	Writing property annotation file D:\syn800_beta2\examples...	-	spcl_regs.srr (201)	14:24:23 ...	VIRTEX4
N	FX164	The option to pack flops in the IOB has not been specified	-	spcl_regs.srr (199)	14:24:23 ...	VIRTEX4
N	MT204	Autoconstrain Mode is ON	-	spcl_regs.srr (100)	14:24:23 ...	VIRTEX4
⊞ N 9	CG364	Synthesizing module eight_bit_uc	-	spcl_regs.srr	14:24:23 ...	Verilog C
⊞ ⚠ 3			alu.v	spcl_regs.srr	14:24:23 ...	Verilog C
⊞ N 32	FX235	Instance "prgmcntr.pc_in[0]" with 3 loads has been replicat...	-	spcl_regs.srr	14:24:23 ...	VIRTEX4
⊞ ⚠ [3]	CL209	Input port bit <10> of pc[10:0] is unused	data_mux.v (10)	spcl_regs.srr (38)	14:24:23 ...	Verilog C
N	FX214	Generating ROM rom.Data_1[11:0]	ins_rom.v (9)	spcl_regs.srr (101)	14:24:23 ...	VIRTEX4
⊞ N 2	CL201	Trying to extract state machine for register status	-	spcl_regs.srr	14:24:23 ...	Verilog C
N	CL134	Found RAM mem_regfile, depth=32, width=8	reg_file.v (18)	spcl_regs.srr (35)	14:24:23 ...	Verilog C
⊞ ⚠ [6]	CL208	All reachable assignments to bit 1 of status[7:0] assign 0, re...	spcl_regs.v (139)	spcl_regs.srr (56)	14:24:23 ...	Verilog C
N	CG346	Read full_case directive	alu.v (83)	spcl_regs.srr (12)	14:24:19 ...	Verilog C

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 4-10. 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.

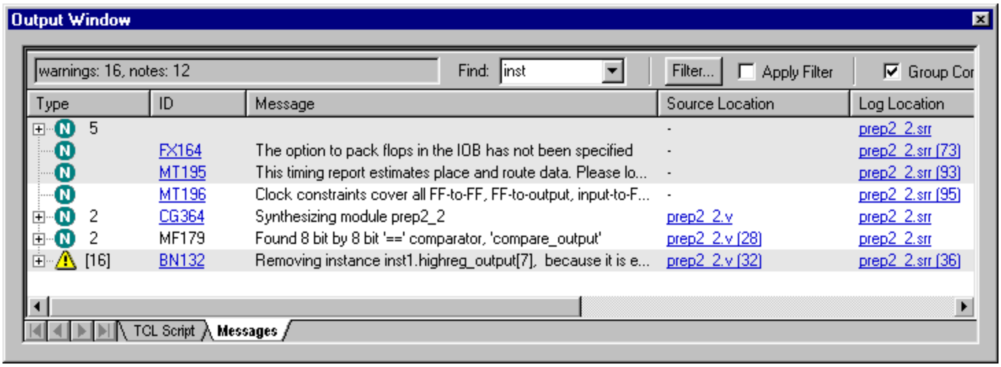
   – To open the corresponding source code file, click the link in the Source Location column. Correct any errors and rerun synthesis. For warnings, see *Handling Warnings*, on page 4-14.

   – 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 4-8. For the command line equivalent of this procedure, see *Filtering Messages from the Command Line*, on page 4-13.

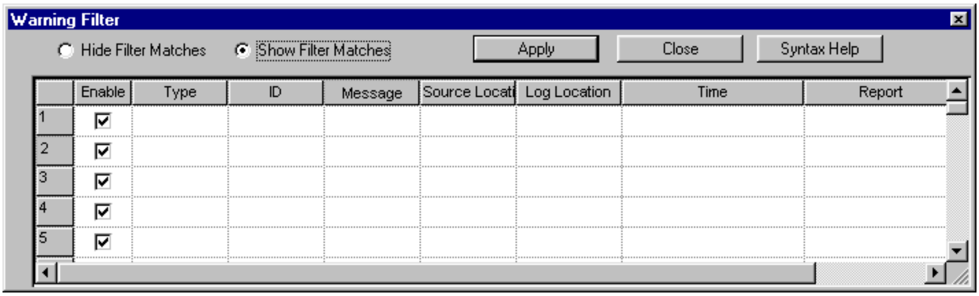
1. Open the message viewer by clicking the Messages tab in the Tcl window.



Type	ID	Message	Source Location	Log Location
⊞ N 5			-	prep2_2.srr
N	FX164	The option to pack flops in the IOB has not been specified	-	prep2_2.srr (73)
N	MT195	This timing report estimates place and route data. Please lo...	-	prep2_2.srr (93)
N	MT196	Clock constraints cover all FF-to-FF, FF-to-output, input-to-F...	-	prep2_2.srr (95)
⊞ N 2	CG364	Synthesizing module prep2_2	prep2_2.v	prep2_2.srr
⊞ N 2	MF179	Found 8 bit by 8 bit '==' comparator, 'compare_output'	prep2_2.v (28)	prep2_2.srr
⊞ ⚠ [16]	BN132	Removing instance inst1.highreg_output[7], because it is e...	prep2_2.v (32)	prep2_2.srr (36)

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.



	Enable	Type	ID	Message	Source Locati	Log Location	Time	Report
1	☑							
2	☑							
3	☑							
4	☑							
5	☑							

3. Set your display preferences.

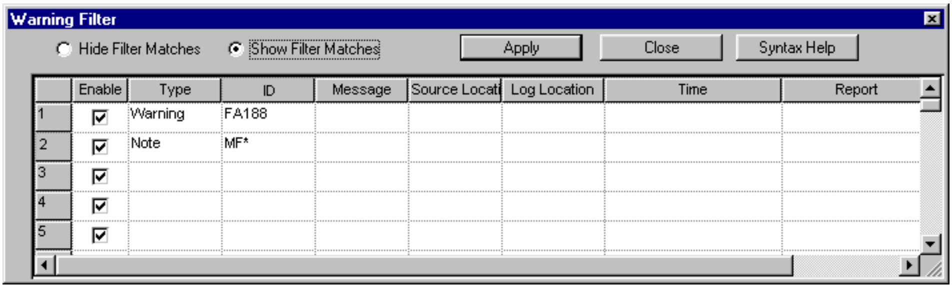
   – To hide your filtered choices from the list of messages, click Hide Filter Matches in the Warning Filter window.

   – 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.



	Enable	Type	ID	Message	Source Locati	Log Location	Time	Report
1	☑	Warning	FA188					
2	☑	Note	MF*					
3	☑							
4	☑							
5	☑							

   – 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 4-10. 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 5-60 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
   ```

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.

Automating Message Filtering with a synhooks 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 4-10 or *Filtering Messages from the Command Line*, on page 4-13 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 10-10.

3. Edit the synhooks.tcl file so that it reads like the following example. For syntax details, see *Tcl synhooks File Syntax*, on page 5-58 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.

Basic Operations in the Schematic Views

The RTL and Technology views 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 designed 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, next
- Opening the Views, on page 4-17
- Analyzing Your Design Graphically, on page 4-19
- Viewing Object Properties, on page 4-20
- Selecting Objects in the RTL/Technology Views, on page 4-23
- Working with Multisheet Schematics, on page 4-24
- Moving Between Views in a Schematic Window, on page 4-26
- Setting Schematic View Preferences, on page 4-26
- Managing Windows, on page 4-28

For information on specific tasks like analyzing critical paths, see the following sections:

- Exploring Object Hierarchy by Pushing/Popping, on page 4-31
- Exploring Object Hierarchy of Transparent Instances, on page 4-36
- Browsing to Find Objects, on page 4-37
- Crossprobing, on page 4-48
- Analyzing With the HDL Analyst Tool, on page 4-56
- Analyzing Timing, on page 4-73

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 Synplicity 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 Synplicity 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...

Start with a compiled design.

To open a hierarchical RTL view, do one of the following:

- Select HDL Analyst->RTL->Hierarchical View.
- Click the RTL View icon (⊕) (a plus sign inside a circle).
- Double-click the `.srs` file in the Implementation Results view.

To open a flattened RTL view, select HDL Analyst->RTL->Flattened View.

To open a Technology view...

Start with a mapped (synthesized) design.

To open a hierarchical Technology view, do one of the following:

- Select HDL Analyst ->Technology->Hierarchical View.
- Click the Technology View icon (NAND gate icon).
- Double-click the `.srm` file in the Implementation Results view.

To open a flattened Technology view, select HDL Analyst-> Technology->Flattened View.

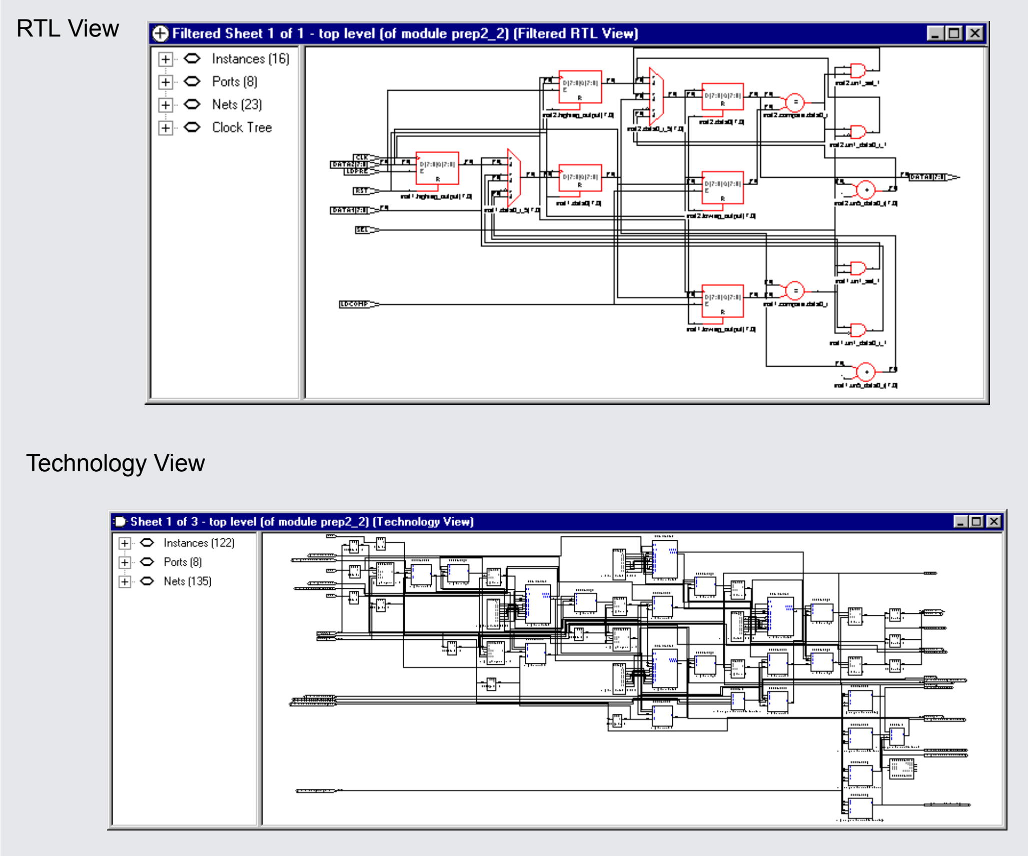
To open a Floorplan view

Start with a synthesized design that has been floorplanned with physical constraint regions.

To open a RTL Floorplan view:

- Select HDL Analyst->RTL->Floorplanned View.
- Double-click the partitioned netlist (`.srp`) file from the Implementation Results view.

All RTL and Technology views have the schematic on the right and a pane on the left that contains a hierarchical list of the objects in the design. This pane is called the Hierarchy Browser. The bar at the top of 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 2-18 in the *Reference Manual* for a description of the Hierarchy Browser.


RTL View
Filtered Sheet 1 of 1 - top level (of module prep2_2) (Filtered RTL View)
Instances (16)
Ports (8)
Nets (23)
Clock Tree
Technology View
Sheet 1 of 3 - top level (of module prep2_2) (Technology View)
Instances (122)
Ports (8)
Nets (135)

Analyzing Your Design Graphically

By 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.

In the Technology view, the software uses module generators to implement the high-level structures from the RTL view using 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. Synplify users do not have access to the FSM view. You can use techniques like crossprobing, flattening, and filtering to isolate and examine the components. The following table points you to where you can find more information about some analysis techniques.

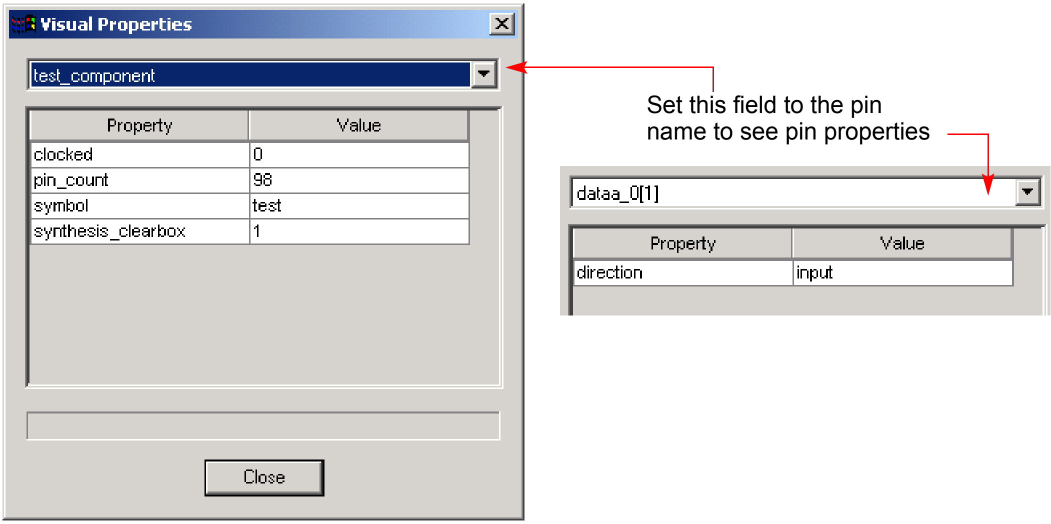
For Information About	See...
Crossprobing	Crossprobing, on page 4-48
Analyzing logic	Analyzing With the HDL Analyst Tool, on page 4-56
Isolating or filtering logic	Filtering Schematics, on page 4-60
Expanding filtered logic	Expanding Pin and Net Logic, on page 4-62 and Expanding and Viewing Connections, on page 4-66
Flattening	Flattening Schematic Hierarchy, on page 4-67
Analyzing timing	Analyzing Timing, on page 4-73

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.



test_component

Property	Value
clocked	0
pin_count	98
symbol	test
synthesis_clearbox	1

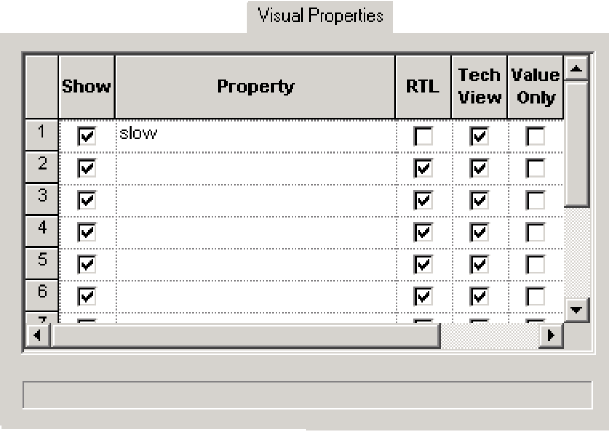
Set this field to the pin name to see pin properties

dataa_0[1]

Property	Value
direction	input

3. 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 pulldown list. Some properties are only available in certain views.



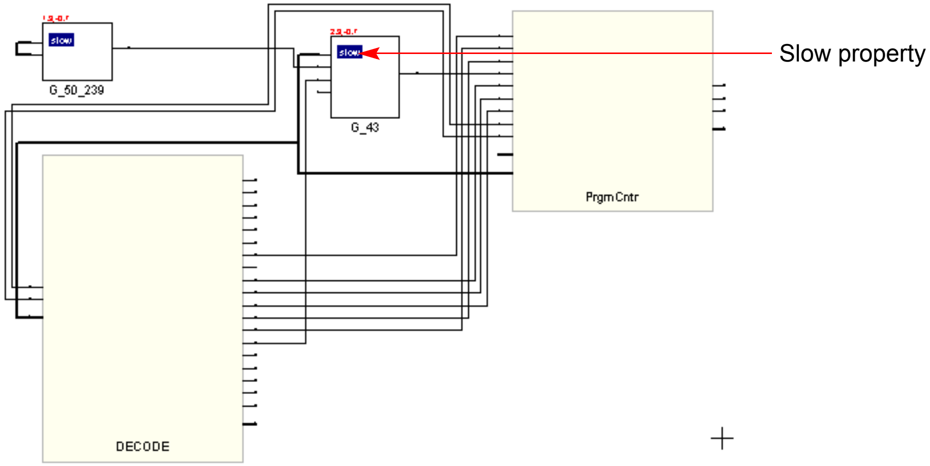
	Show	Property	RTL	Tech View	Value Only
1	☑	slow	□	☑	□
2	☑		☑	☑	□
3	☑		☑	☑	□
4	☑		☑	☑	□
5	☑		☑	☑	□
6	☑		☑	☑	□

   – 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

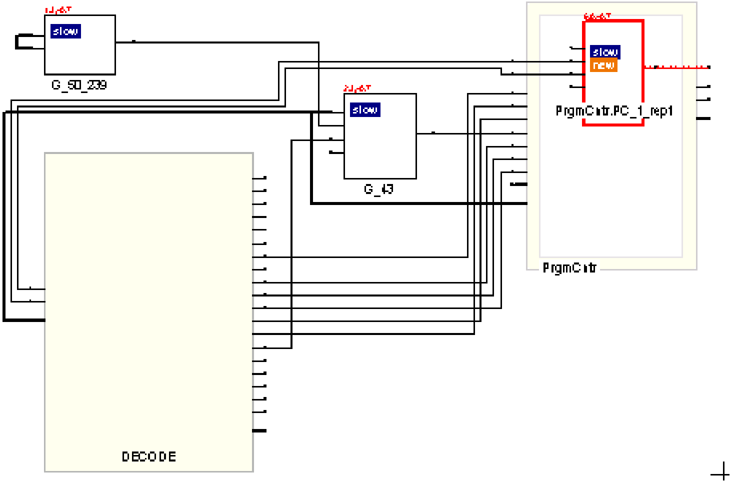
You can view objects with the slow property when you are analyzing your critical path. All objects with this property do not meet the timing criteria. The following figure shows a filtered view of a critical path, with slow instances flagged in blue.



Slow property

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: • Draw a rectangle around the objects. • Select an object, press Ctrl, and click other objects you want to select. • Select multiple objects in the Hierarchy Browser. See *Browsing With the Hierarchy Browser*, on page 4-37. • Use Find to select the objects you want. See *Using Find for Hierarchical and Restricted Searches*, on page 4-39.
Objects by type (instances, ports, nets)	Use Edit->Find to select the objects (see *Browsing With the Find Command*, on page 4-38), 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: • Right-click and choose the appropriate command from the Select All Schematic/Current Sheet popup menus. • Select the objects in the Hierarchy Browser.
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 4-48 for more information about crossprobing.

Some commands affect selection by adding to the selected set of objects: the Expand commands, the Select All commands, and the Select Net Driver and Select Net Instances commands.

Working with Multisheet Schematics

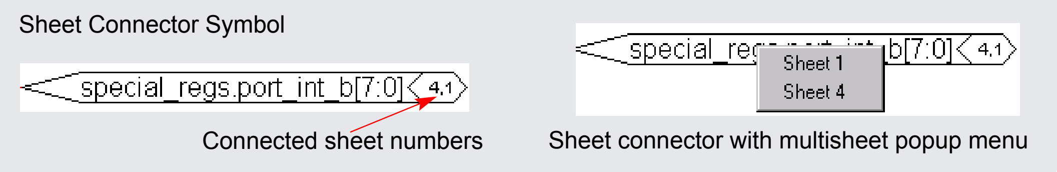
The title bar of the RTL or Technology view indicates the number of sheets in that schematic. In a multisheet schematic, nets that span multiple sheets are indicated by sheet connector symbols, which you can use for navigation.

1. To reduce the number of sheets in a schematic, select Options->HDL Analyst Options and increase the values set for Sheet Size Options - Instances and Sheet Size Options - Filtered Instances. To display fewer objects per sheet (increase the number of sheets), increase the values.

   These options set a limit on the number of objects displayed on an unfiltered and filtered schematic sheet, respectively. A low Filtered Instances value can cause lower-level logic inside a transparent instance to be displayed on a separate sheet. The sheet numbers are indicated inside the empty transparent instance.

2. To navigate through a multisheet schematic, refer to this table. It summarizes common operations and ways to navigate.

To view...	Use one of these methods...
Next sheet or previous sheet	Select View->Next/Previous Sheet. Press the right mouse button and draw a horizontal mouse stroke (left to right for next sheet, right to left for previous sheet). Click the icons: Next Sheet (➡) or Previous Sheet (⬅) Press Shift-right arrow (Next Sheet) or Shift-left arrow (Previous sheet). Navigate with View->Back and View ->Forward if the next/previous sheets are part of the display history.
A specific sheet number	Select View->View Sheets and select the sheet. Click the right mouse button, select View Sheets from the popup menu, and then select the sheet you want. Press Ctrl-g and select the sheet you want.
Lower-level logic of a transparent instance on separate sheets	Check the sheet numbers indicated inside the empty transparent instance. Use the sheet navigation commands like Next Sheet or View Sheets to move to the sheet you need.
All objects of a certain type	To highlight all the objects of the same type in the schematic, right-click and select the appropriate command from the Select All Schematic popup menu. To highlight all the objects of the same type on the current sheet, right-click and select the appropriate command from the Select All Sheet popup menu.
Selected items only	Filter the schematic as described in *Filtering Schematics,* on page 4-60.
A net across sheets	If there are no sheet numbers displayed in a hexagon at the end of the sheet connector, select Options ->HDL Analyst Options and enable Show Sheet Connector Index. Right-click the sheet connector and select the sheet number from the popup as shown in the following figure.


Sheet Connector Symbol

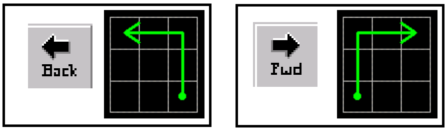
Sheet connector with multisheet popup menu

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.

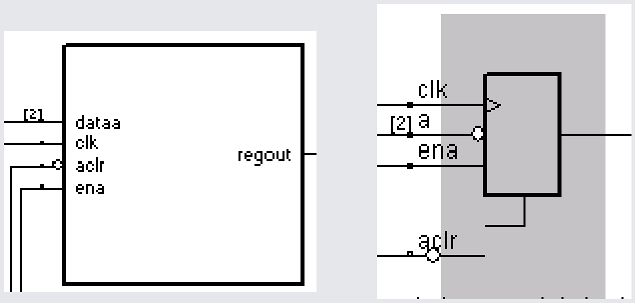
1. Select Options->HDL Analyst Options. For a description of all the options on this form, see *HDL Analyst Options Command*, on page 3-99 in the *Reference Manual*.

2. The following table details some common operations:

To...	Do this...
Display the Hierarchy Browser	Enable Show Hierarchy Browser (General tab).
Control crossprobing from an object to a P&R text file	Enable Enhanced Text Crossprobing. (General tab)
Determine the number of objects displayed on a sheet.	Set the value with Maximum Instances on the Sheet Size tab. Increase the value to display more objects per sheet.
Determine the number of objects displayed on a sheet in a filtered view.	Set the value with Maximum Filtered Instances on the Sheet Size tab. Increase the number to display more objects per sheet. You cannot set this option to a value less than the Maximum Instances value.

Some of these options do not take effect in the current view, but are visible in the next schematic view you open.

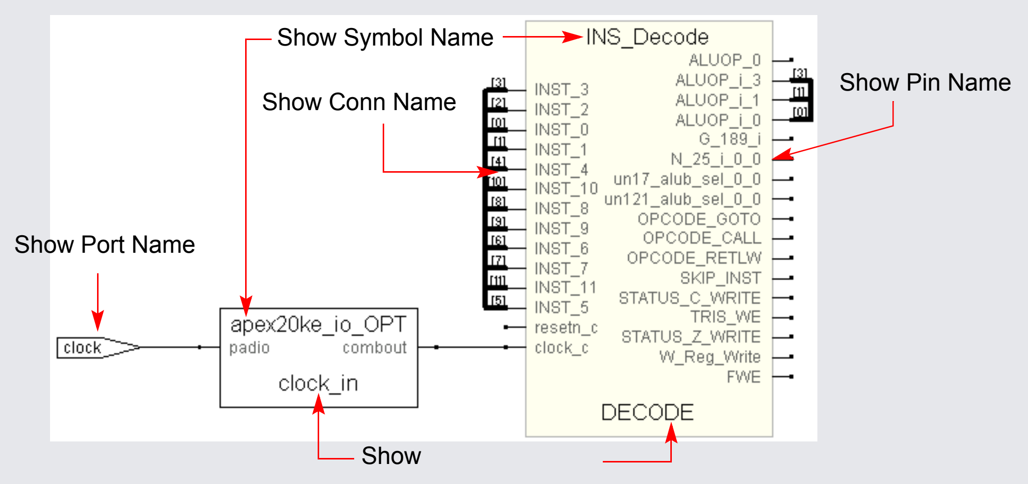
3. To view hierarchy within a cell, enable the General->Show Cell Interiors option.



Show Cell Interior off

Show Cell Interior on

4. 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 6-9* in the *Reference Manual.*

5. Click OK on the HDL Analyst Options form.

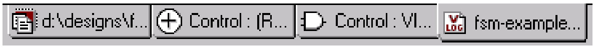
   The software writes the preferences you set to the `.ini` file, and they remain in effect until you change them.

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 4-26*.

1. Toggle on View->Workbook Mode.

   Below the Project view, you see tabs like the following for each open view. The tab for the current view is on top. The symbols in front of the view name on the tab help identify the kind of view.



2. To bring an open view to the front, if the window is not visible, click its tab. If part of the window is visible, click in any part of the window.

   If you previously minimized the view, it will be in minimized form. Double-click the minimized view to open it.

3. To bring the next view to the front, click Ctrl-F6 in that window.

4. Order the display of open views with the commands from the Window menu. You can cascade the views (stack them, slightly offset), or tile them horizontally or vertically.

5. To close a view, press Ctrl-F4 in that window or select File->Close.

Exploring Design Hierarchy

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 4-56. The topics include:

- Traversing Design Hierarchy with the Hierarchy Browser, on page 4-30
- Exploring Object Hierarchy by Pushing/Popping, on page 4-31
- Exploring Object Hierarchy of Transparent Instances, on page 4-36

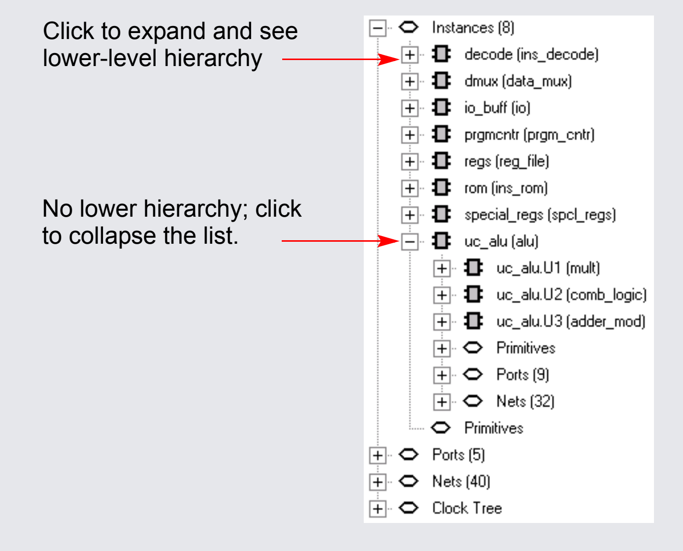
Traversing Design Hierarchy with the Hierarchy Browser

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 4-31 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 2-19 in the *Reference Manual* for an explanation of the symbols.

Exploring Object Hierarchy by Pushing/Popping

To view the internal hierarchy of a specific object, it is best to use Push/Pop mode or examine transparent instances, instead of using the Hierarchy Browser described in *Traversing Design Hierarchy with the Hierarchy Browser*, on page 4-30. 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 4-60, *Expanding Pin and Net Logic*, on page 4-62, and *Expanding and Viewing Connections*, on page 4-66 for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

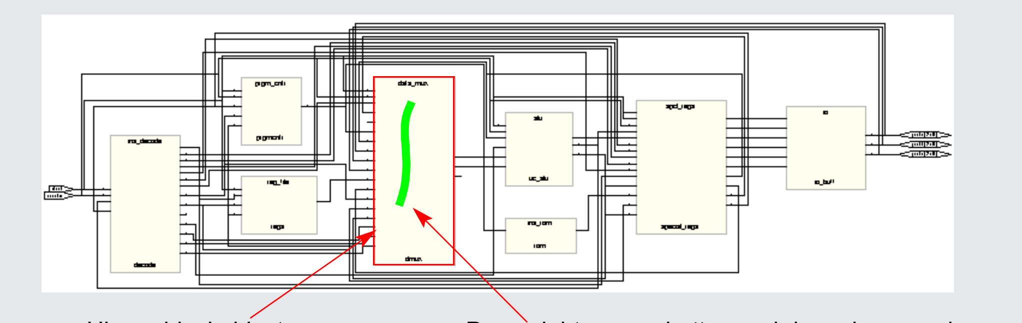
- Pushing into Objects, next
- Popping up a Hierarchical Level, on page 4-35

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).



Hierarchical object

Press right mouse button and draw downward to push into an object

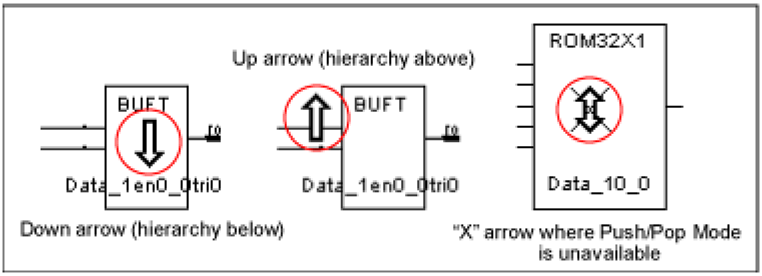
   – Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the Technology view, under Instances - Primitives.

   – 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.



Up arrow (hierarchy above)

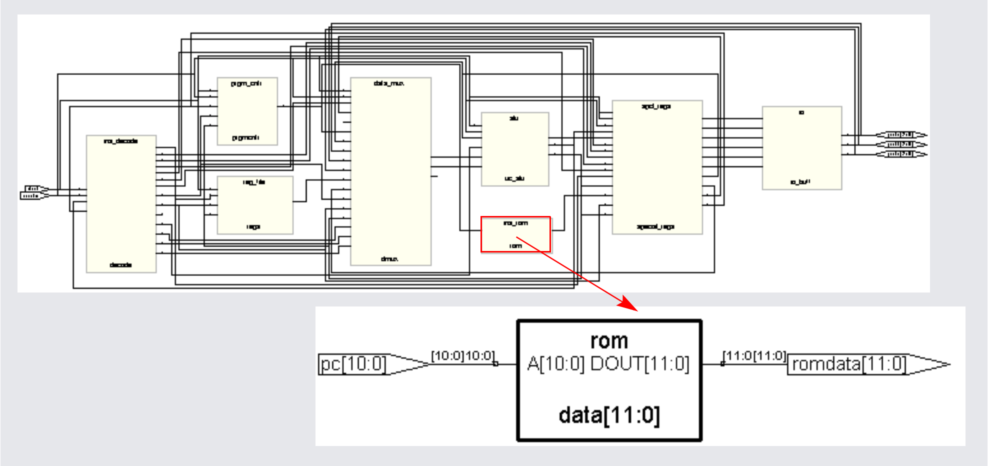
Down arrow (hierarchy below)

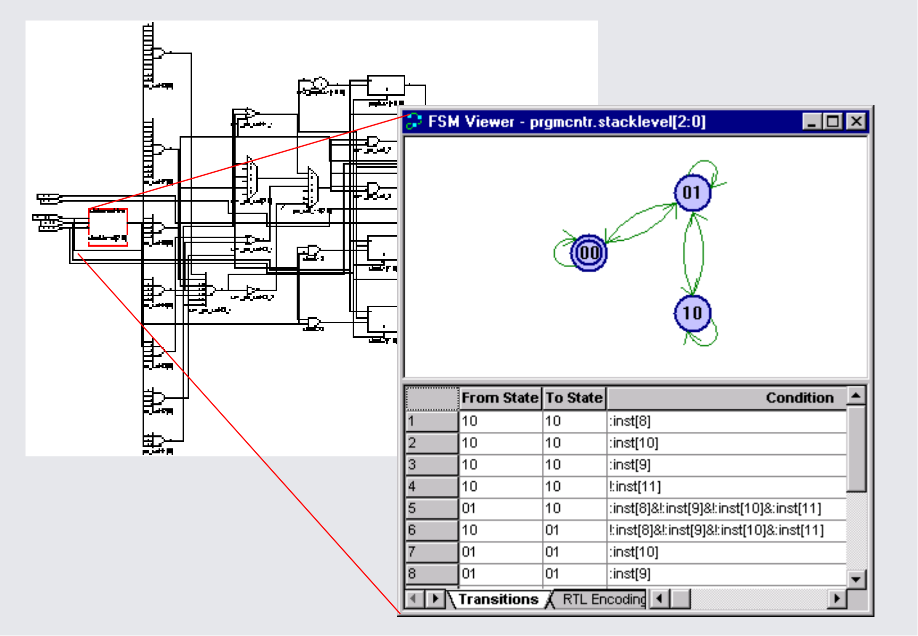
"X" arrow where Push/Pop Mode is unavailable

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 6-25*. If you push into a state machine from the Technology view, you see the underlying logic.

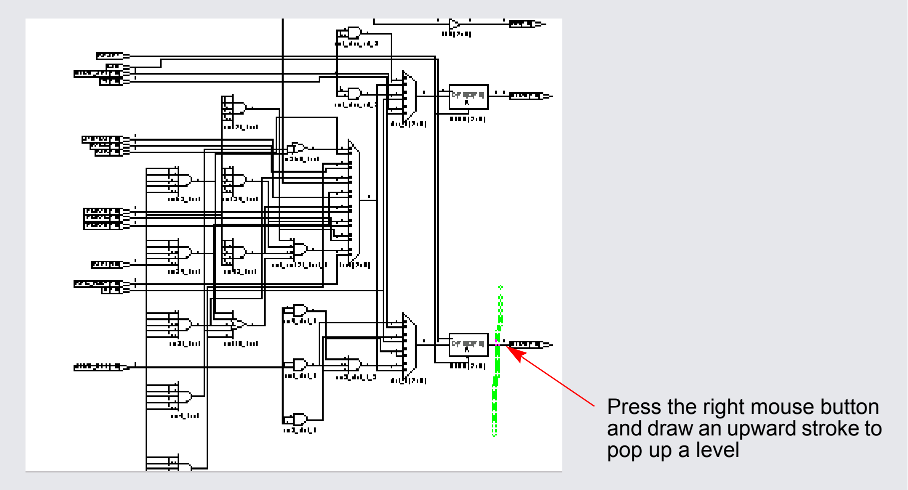




	From State	To State	Condition
1	10	10	:inst[8]
2	10	10	:inst[10]
3	10	10	:inst[9]
4	10	10	!:inst[11]
5	01	10	:inst[8]&!:inst[9]&!:inst[10]&:inst[11]
6	10	01	!:inst[8]&!:inst[9]&!:inst[10]&:inst[11]
7	01	01	:inst[10]
8	01	01	:inst[9]

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.



Press the right mouse button and draw an upward stroke to pop up a level

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 4-32for 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 4-31).

	Pushing	Transparent Instance
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, next
- Using Find for Hierarchical and Restricted Searches, on page 4-39
- Using Wildcards with the Find Command, on page 4-42
- Using Find to Search the Output Netlist, on page 4-45

For infomation about the Tcl Find command, which you use to locate objects, and create collections, see *Tcl find Command*, on page 5-48.

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.

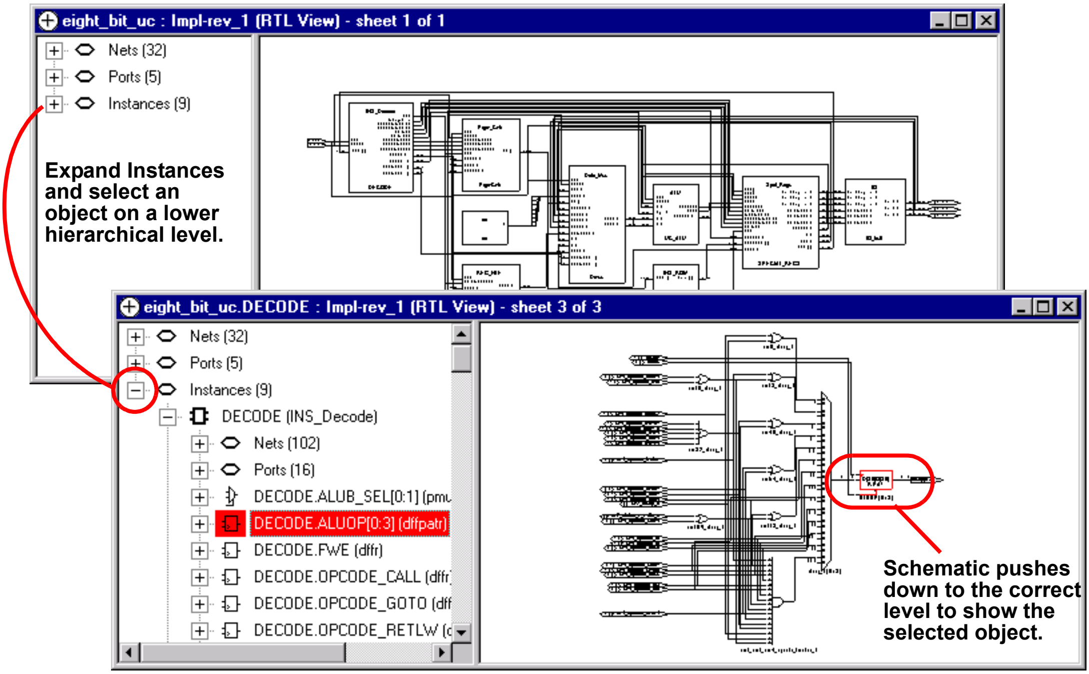
   The software selects and highlights all the objects in the range.

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.



Expand Instances and select an object on a lower hierarchical level.

Schematic pushes down to the correct level to show 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.

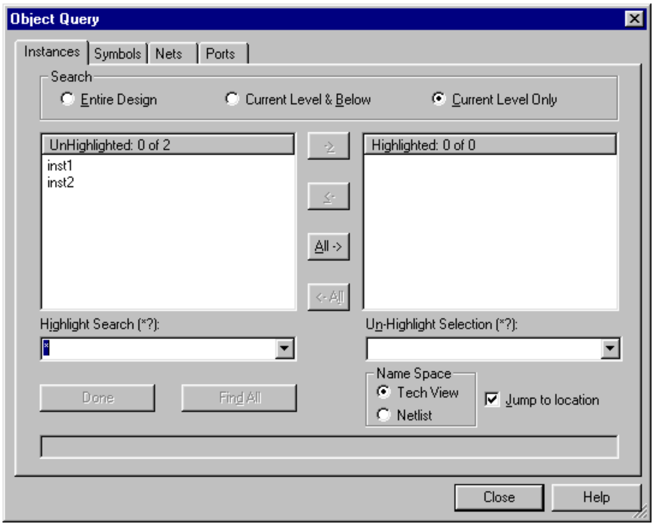
Using Find for Hierarchical and Restricted Searches

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 4-37). 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 4-56 and *Filtering Schematics,* on page 4-60 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. Select HDL Analyst->Find or press Ctrl-f to open the Object Query dialog box. Reposition the dialog box so you can see both your schematic and the dialog box.



Search: ○ Entire Design ○ Current Level & Below ● Current Level Only

UnHighlighted: 0 of 2
inst1
inst2

Highlighted: 0 of 0

Highlight Search (*?): *

Un-Highlight Selection (*?):

Name Space: ● Tech View ○ Netlist

☑ Jump to location

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 4-42 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 4-45.

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 4-42 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*`

8. 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 3-17 of the *Reference Manual*.

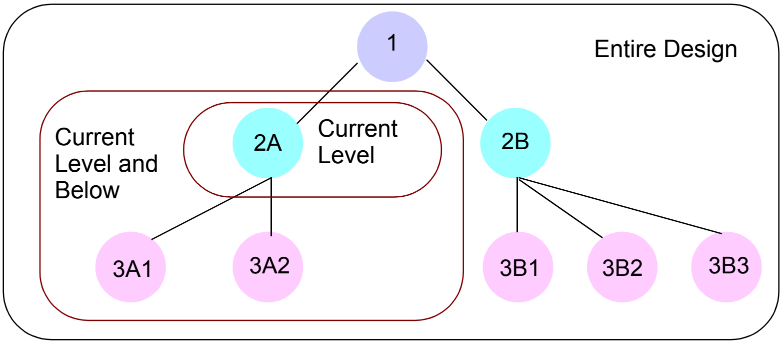
Using Wildcards with the Find Command

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 4-44 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.

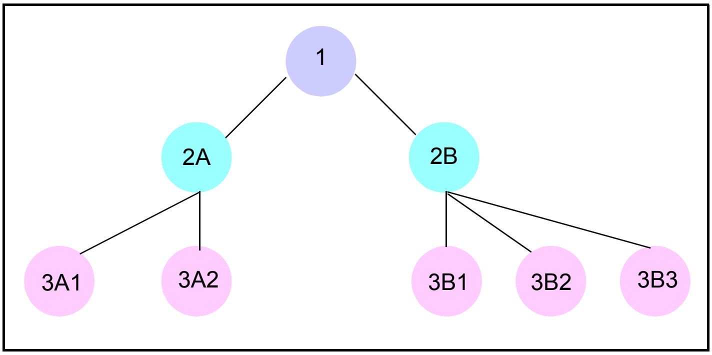
2. The software applies the wildcard pattern to all applicable objects within the range. For Current Level and Current Level and Below, the current level determines the starting point.

   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 4-42 and *Wildcard Search Examples, on page 4-44* 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) 3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B) No matches in 1 (because of the hierarchical dot), unless a name includes a non-hierarchical dot.
Current Level	*	1	1 only (no hierarchical boundary crossing)
	.	2B	2B only. No search of lower levels even though the dot is specified, because the scope is Current Level. No matches, unless a 2B name includes a non-hierarchical dot.

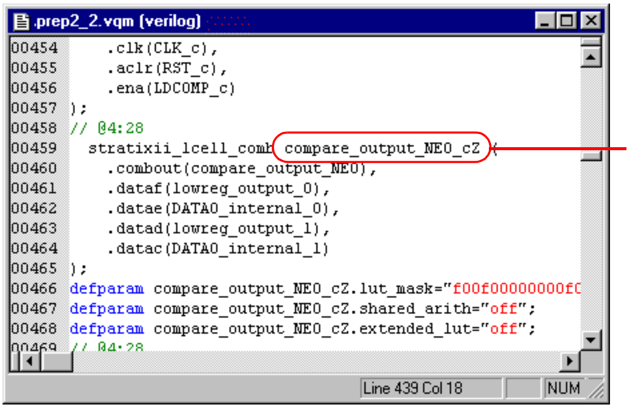
Scope	Pattern	Starting Point	Finds Matches in...
Current Level and Below	*	2A	2A only (no hierarchical boundary crossing)
	.	1	2A and 2B (*.* from 1) 3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B) No matches from 1, because the dot is specified.
	.	2B	3B1, 3B2, and 3B3 (*.* from 2B)
	.	3A2	No matches (no hierarchy below 3A2)
	..*	1	3A1, 3A2, 3B1, 3B2, and 3B3 (*.*.* from 1) Search ends because there is no hierarchy two levels below 2A and 2B.

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 Options for Implementations dialog box.

2. After you synthesize your design, open your output netlist file and select the name of the object you want to find.



.prep2_2.vqm (verilog)

```
    .clk(CLK_c),
    .aclr(RST_c),
    .ena(LDCOMP_c)
);
// @4:28
  stratixii_lcell_comb compare_output_NE0_cZ (
    .combout(compare_output_NE0),
    .dataf(lowreg_output_0),
    .datae(DATA0_internal_0),
    .datad(lowreg_output_1),
    .datac(DATA0_internal_1)
);
defparam compare_output_NE0_cZ.lut_mask="f00f00000000f0
defparam compare_output_NE0_cZ.shared_arith="off";
defparam compare_output_NE0_cZ.extended_lut="off";
// @4:28
```

Line 439 Col 18 NUM

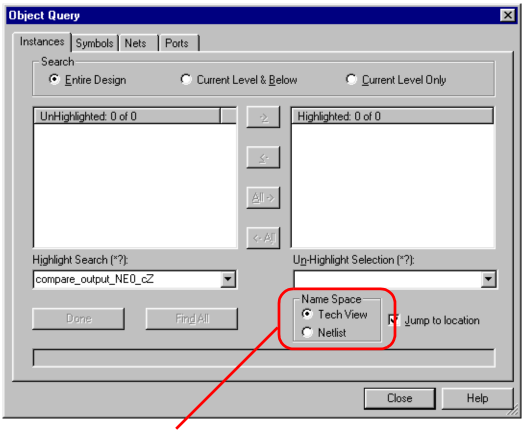
Copy Name

3. Copy the name and open a Technology view.

4. In the Technology view, press Ctrl-f or select Edit->Find to open the Object Query dialog box and do the following:

   – Paste the object name you copied into the Highlight Search field.

   – Set the Name Space option to Netlist and click Find All.



Object Query

Search: ● Entire Design ○ Current Level & Below ○ Current Level Only

UnHighlighted: 0 of 0

Highlighted: 0 of 0

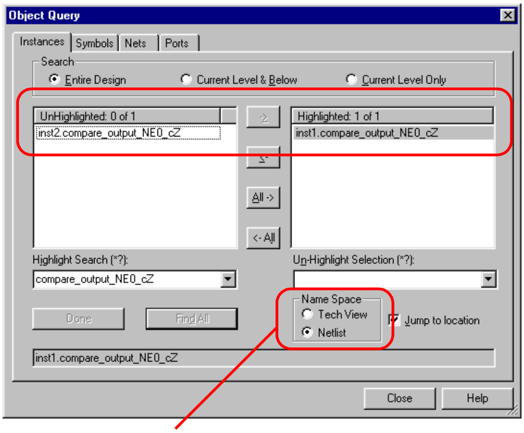
Highlight Search (*?): compare_output_NE0_cZ

Un-Highlight Selection (*?):

Name Space: ● Tech View ○ Netlist

☑ Jump to location

Search by Tech View



Object Query

Search: ● Entire Design ○ Current Level & Below ○ Current Level Only

UnHighlighted: 0 of 1
inst2.compare_output_NE0_cZ

Highlighted: 1 of 1
inst1.compare_output_NE0_cZ

Highlight Search (*?): compare_output_NE0_cZ

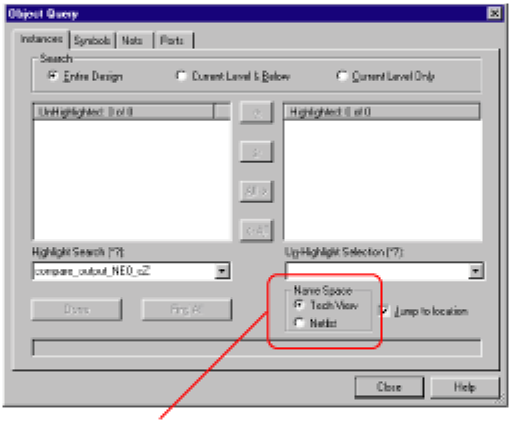
Un-Highlight Selection (*?):

Name Space: ○ Tech View ● Netlist

☑ Jump to location

inst1.compare_output_NE0_cZ

Search by Netlist



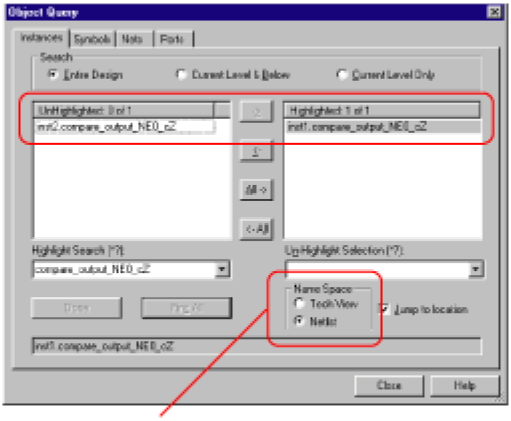
Search: ● Entire Design ○ Current Level & Below ○ Current Level Only

Highlight Search (*?): compare_output_NE0_cZ

Name Space: ● Tech View ○ Netlist

☑ Jump to location

Search by TechView



Search: ● Entire Design ○ Current Level & Below ○ Current Level Only

UnHighlighted: 0 of 1

inst2.compare_output_NE0_cZ

Highlighted: 1 of 1

inst1.compare_output_NE0_cZ

Highlight Search (*?): compare_output_NE0_cZ

Name Space: ○ Tech View ● Netlist

☑ Jump to location

inst1.compare_output_NE0_cZ

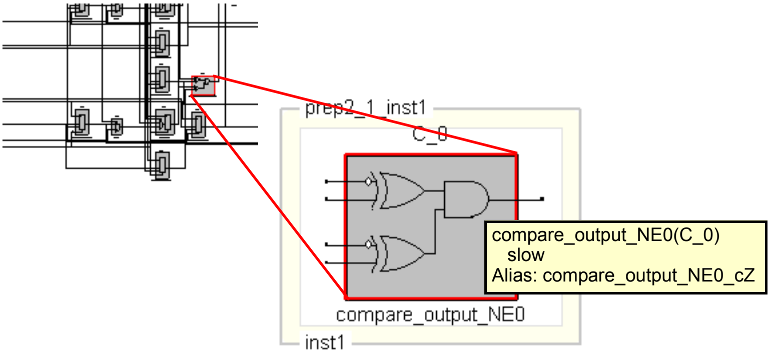
Search by Netlist

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.



compare_output_NE0(C_0)
slow
Alias: compare_output_NE0_cZ

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

This section describes how to crossprobe from different views. It includes the following:

- Crossprobing Description, on page 4-48, next
- Crossprobing within an RTL/Technology View, on page 4-49
- Crossprobing from the RTL/Technology View, on page 4-49
- Crossprobing from the Text Editor Window, on page 4-51
- Crossprobing from the Tcl Script Window, on page 4-54
- Crossprobing from the FSM Viewer, on page 4-54

Crossprobing Description

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.

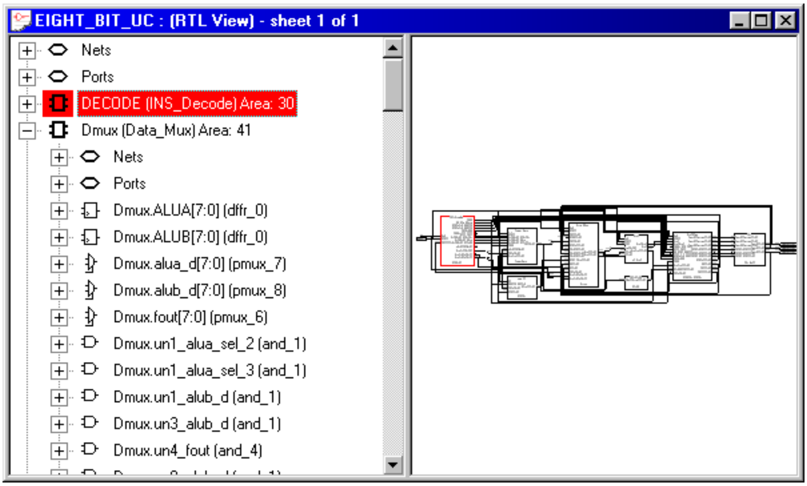
For further details, see *Crossprobing*, on page 4-48 of the *Reference Manual*.

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.



EIGHT_BIT_UC : (RTL View) - sheet 1 of 1

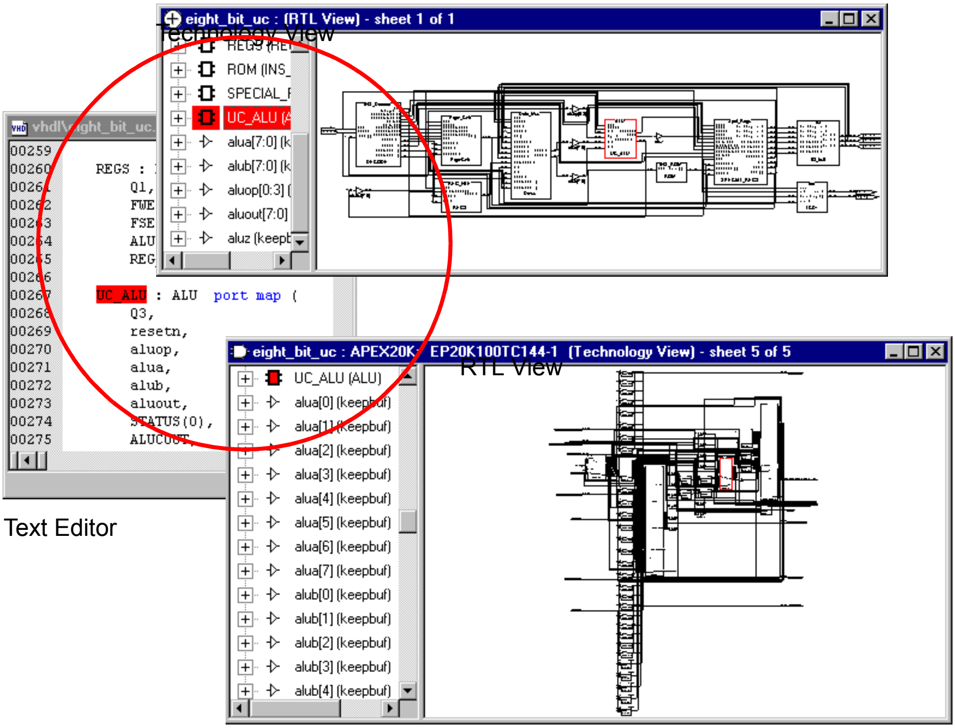
- Nets
- Ports
- DECODE (INS_Decode) Area: 30
- Dmux (Data_Mux) Area: 41
  - Nets
  - Ports
  - Dmux.ALUA[7:0] (dffr_0)
  - Dmux.ALUB[7:0] (dffr_0)
  - Dmux.alua_d[7:0] (pmux_7)
  - Dmux.alub_d[7:0] (pmux_8)
  - Dmux.fout[7:0] (pmux_6)
  - Dmux.un1_alua_sel_2 (and_1)
  - Dmux.un1_alua_sel_3 (and_1)
  - Dmux.un1_alub_d (and_1)
  - Dmux.un3_alub_d (and_1)
  - Dmux.un4_fout (and_4)

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.



```

    REGS : 
        Q1,
        FWE
        FSE
        ALU
        REG

    UC_ALU : ALU  port map (
        Q3,
        resetn,
        aluop,
        alua,
        alub,
        aluout,
        STATUS(0),
        ALUCOUT,
```

2. To crossprobe from the RTL or Technology view to the source file when the source file is not open, double-click on the object in the RTL or Technology view.

   Double-clicking automatically opens the appropriate source code file and highlights the appropriate code. For example, if you double-click an object in a Technology view, the HDL Analyst tool automatically opens an editor window with the source code and highlights the code that contains the selected register.

The following table summarizes the crossprobing capability from the RTL or Technology view.

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. 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 4-52 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.

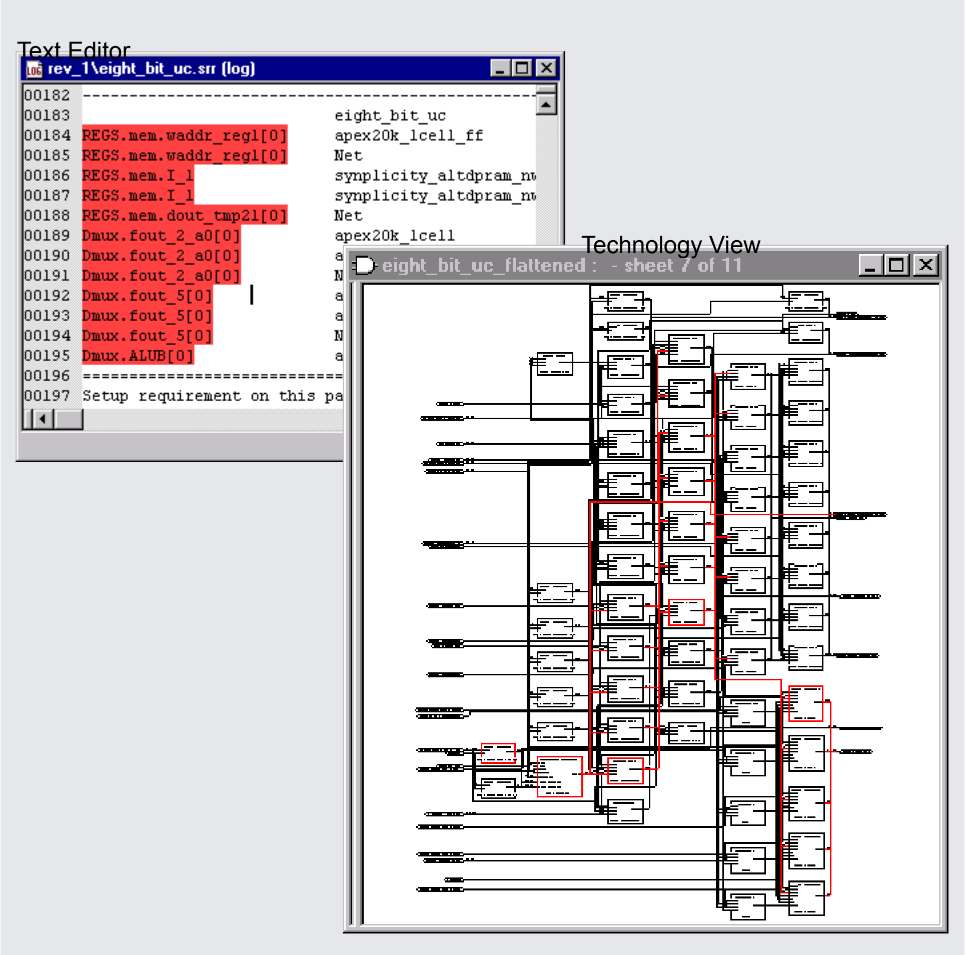
1. Open the log file, the RTL, and Technology views.

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 UNIX and Linux platforms, use the key to which the Alt function is mapped; this is usually the Meta or Diamond key for UNIX or the Ctrl-Alt key combination for Linux.

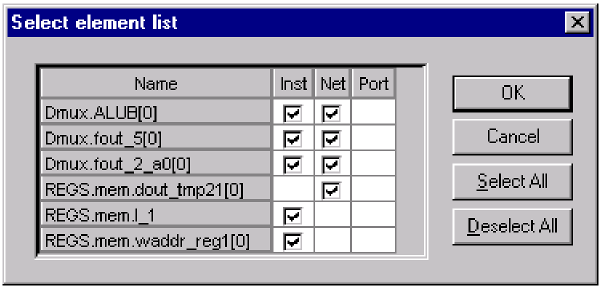
   – To select all the objects in the path, right-click and choose Select All from the popup menu. Alternatively, you can select certain objects only, as described next.

   The software selects the objects in the column, and highlights the path in the open RTL and Technology views.


Text Editor
rev_1\eight_bit_uc.srr (log)
Technology View
eight_bit_uc_flattened : - sheet 7 of 11

```
-----------------------------------------------------
                                          eight_bit_uc
REGS.mem.waddr_regl[0]                    apex20k_lcell_ff
REGS.mem.waddr_regl[0]                    Net
REGS.mem.I_1                              synplicity_altdpram_nw
REGS.mem.I_1                              synplicity_altdpram_nw
REGS.mem.dout_tmp21[0]                    Net
Dmux.fout_2_a0[0]                         apex20k_lcell
Dmux.fout_2_a0[0]                         a
Dmux.fout_2_a0[0]                         N
Dmux.fout_5[0]                            a
Dmux.fout_5[0]                            a
Dmux.fout_5[0]                            N
Dmux.ALUB[0]                              a
============================
Setup requirement on this pa
```

– 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. The corresponding objects are highlighted.



Name	Inst	Net	Port
Dmux.ALUB[0]	☑	☑	
Dmux.fout_5[0]	☑	☑	
Dmux.fout_2_a0[0]	☑	☑	
REGS.mem.dout_tmp21[0]		☑	
REGS.mem.I_1	☑		
REGS.mem.waddr_reg1[0]	☑		

3. To isolate and view only the selected objects, do this in the Technology view: press F12, or right-click and select the Filter Schematic command from the popup menu.

   You see just the selected objects.

Crossprobing from the Tcl Script Window

Crossprobing from the Tcl script window is useful for debugging error messages. 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 software does not support the FSM viewer.

To crossprobe from the FSM Viewer, do the following:

1. Open the view to which you want to crossprobe: RTL/Technology view, or the source code file.

2. Do the following in the open FSM view:

– For FSMs with a onehot encoding style, click the state bubbles in the bubble diagram or the states in the FSM transition table.

– For all other FSMs, click the states in the bubble diagram. You cannot use the transition table because with these encoding styles, the number of registers in the RTL or Technology views do not match the number of registers in the FSM Viewer.

The software highlights the corresponding code or object in the open views. You can only crossprobe from a state in the FSM table if you used a onehot encoding style.

Analyzing With the HDL Analyst Tool

The HDL Analyst tool is a graphical productivity tool that helps you visualize your synthesis results, and improve device performance and area results. The hierarchical RTL-level and technology-primitive level schematics let you graphically view and analyze your design, as described in subsequent sections. See the following for more information:

- Viewing Design Hierarchy and Context, next
- Filtering Schematics, on page 4-60
- Expanding Pin and Net Logic, on page 4-62
- Expanding and Viewing Connections, on page 4-66

The HDL Analyst views also let you analyze timing and crossprobe, and these operations are described in other sections: *Basic Operations in the Schematic Views*, on page 4-16, *Exploring Design Hierarchy*, on page 4-30, *Finding Objects*, on page 4-37, *Crossprobing*, on page 4-48, and *Analyzing Timing*, on page 4-73.

Viewing Design Hierarchy and Context

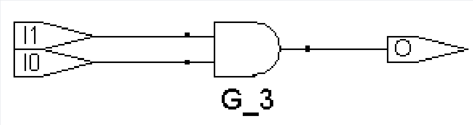
Most large designs are hierarchical, so the synthesis software provides tools that help you view hierarchy details or put the details in context. Alternatively, you can browse and navigate hierarchy with Push/Pop mode, or flatten the design to view internal hierarchy.

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 4-74 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.

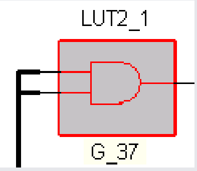
1. To view the internal logic of primitives in your design, do either of the following:
   – To view the logic of an individual primitive, push into it. This generates a new schematic view with the internal details. Click the Back icon to return to the previous view.

– 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 4-60) 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:



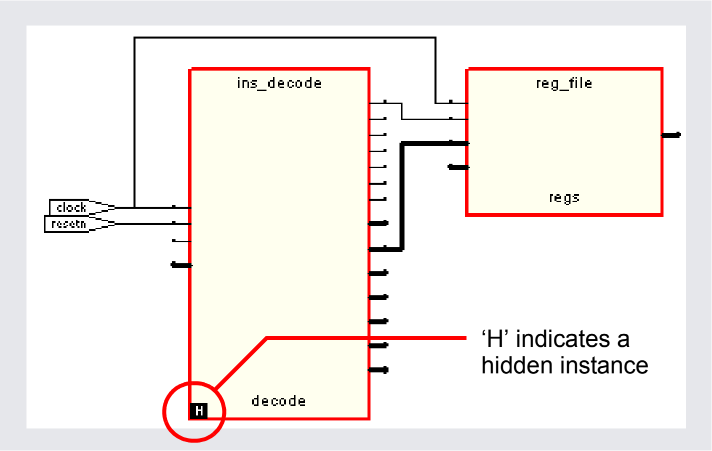
Result of pushing into a primitive (new view of lower-level logic)



Result of enabling Show Cell Interior option (same view with internal logic)

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 Back. See *Exploring Object Hierarchy by Pushing/Popping*, on page 4-31 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 *Flattening Schematic Hierarchy*, on page 4-67 for details about flattening designs. Because this is a new view, you cannot use Back to return to the previous view. To return to the top-level unflattened schematic, right-click in the view and select Unflatten Schematic.
Flattening an instance by dissolving	Generates a view where the hierarchy of the selected instances is flattened, but the rest of the design is unaffected. This provides context. See *Flattening Schematic Hierarchy*, on page 4-67 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

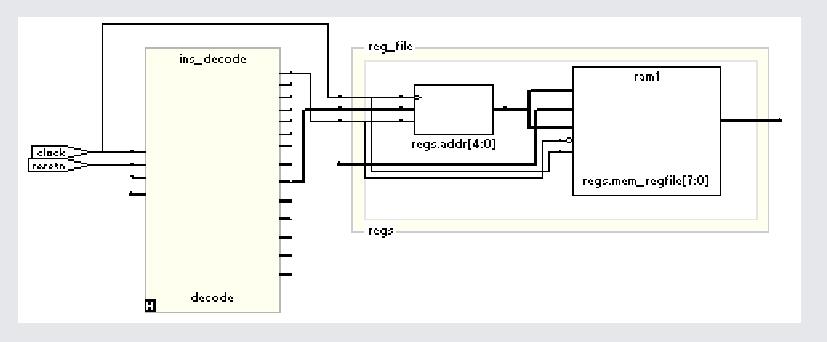
Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter Schematic command to isolate selected 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.

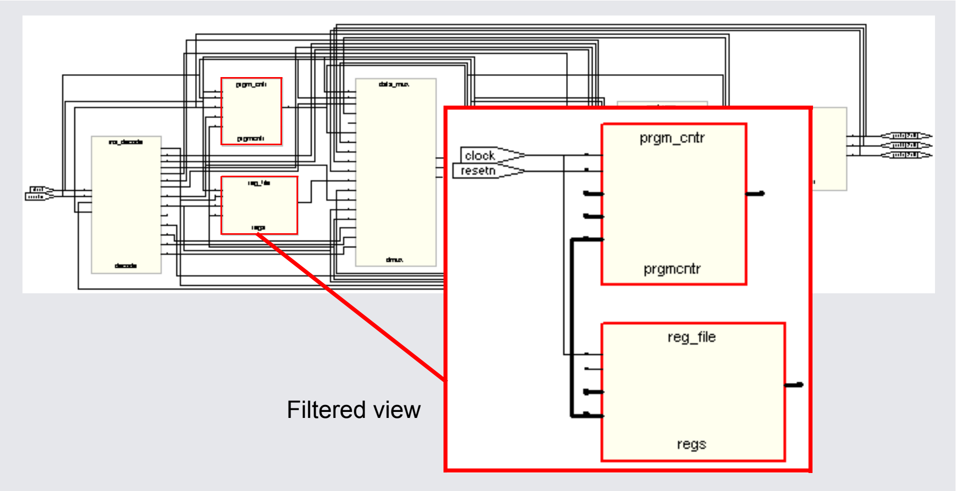


2. Select the Filter Schematic command, using one of these methods:

   – 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 6-2 and *Transparent and Opaque Display of Hierarchical Instances*, on page 6-7 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 4-56, step 4.



Filtered view

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 *Expanding Pin and Net Logic,* on page 4-62 and *Expanding and Viewing Connections,* on page 4-66
Filter further	Select objects and filter again
Find objects	See *Finding Objects,* on page 4-37
Flatten, or hide and flatten	See *Flattening Schematic Hierarchy,* on page 4-67. You can hide transparent or opaque instances.
Crossprobe from filtered view	See *Crossprobing from the RTL/Technology View,* on page 4-49

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 4-60 and *Flattening Schematic Hierarchy,* on page 4-67 of the *Reference Manual*.

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 4-66.

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 4-60 and *Flattening Schematic Hierarchy,* on page 4-67 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 *Expanding Filtered Logic Example,* on page 4-64.
See all cells that are connected to a pin, up to the next register	Select a pin and select Expand to Register/Port. See *Expanding Filtered Logic to Register/Port Example,* on page 4-65.
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 *Expanding Inwards Example,* on page 4-65.

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 3-69 of the *Reference Manual*.

2. To expand logic from a pin at the current level only, do the following:

   – Select a pin, and go to the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.

   – Select Expand or Expand to Register/Ports. The commands work as described in the previous step, but they do not cross hierarchical boundaries.

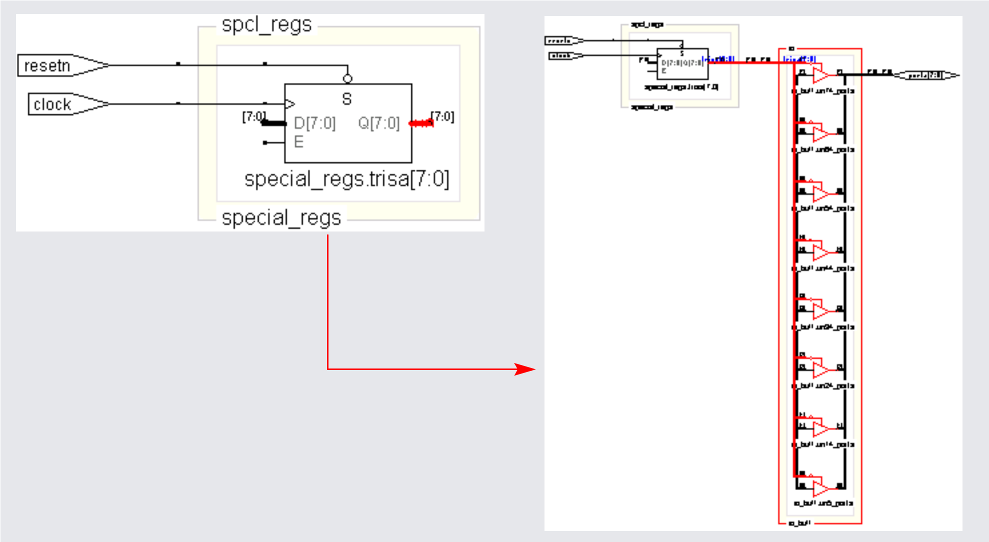
3. To expand logic from a net, use the commands shown in the following table.

   – To expand at the current level and below, select the commands from the HDL Analyst->Hierarchical menu or the right-click popup menu.

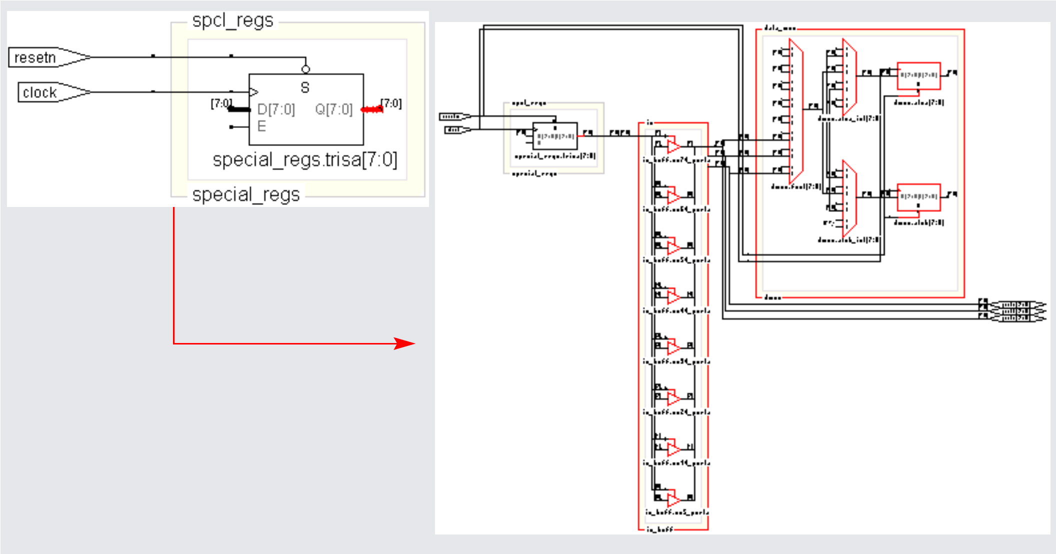
   – To expand at the current level only, select the commands from the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.

To...	Do this...
Select the driver of a net	Select a net and select Select Net Driver. The result is a filtered view with the net driver selected (*Selecting the Net Driver Example*, on page 4-66).
Trace the driver, across sheets if needed	Select a net and select Go to Net Driver. The software shows a view that includes the net driver.
Select all instances on a net	Select a net and select Select Net Instances. You see a filtered view of all instances connected to the selected net.

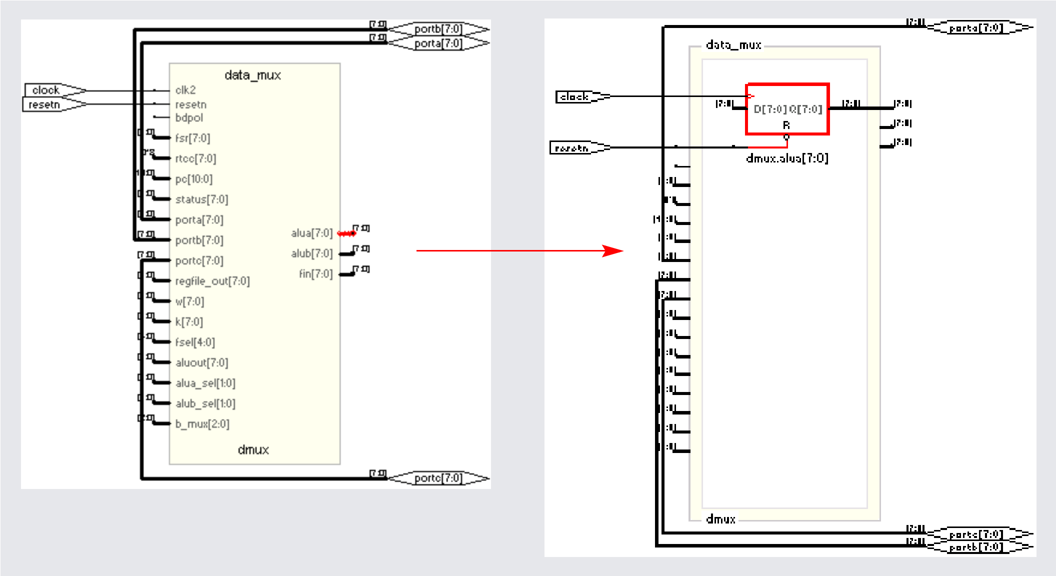
Expanding Filtered Logic Example



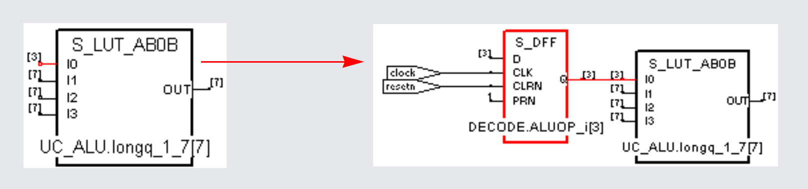
Expanding Filtered Logic to Register/Port Example



Expanding Inwards Example



Selecting the Net Driver Example



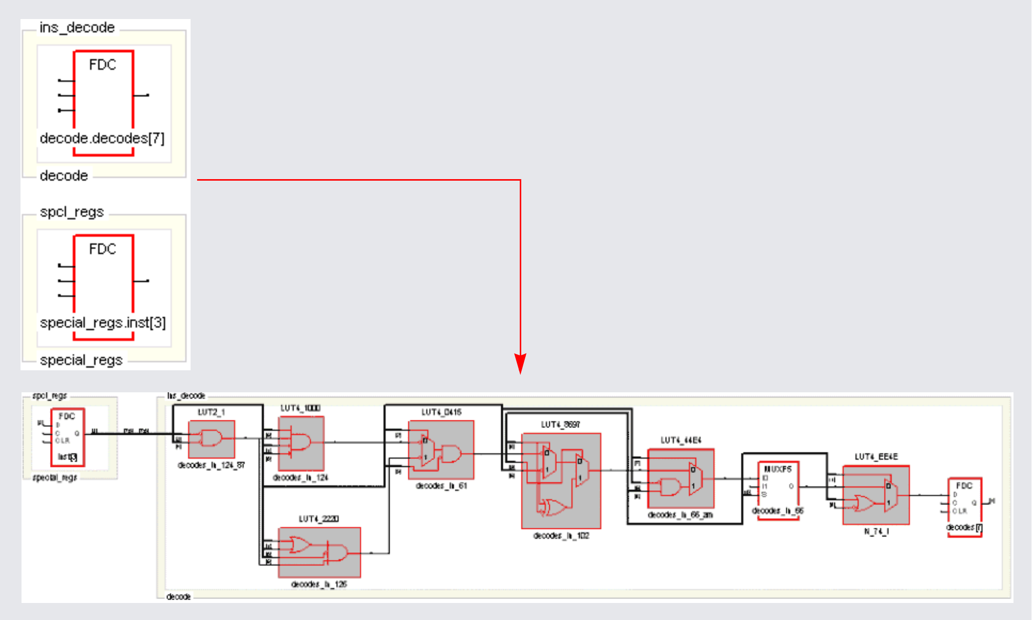
Expanding and Viewing Connections

This section describes commands that expand logic between two or more objects; to expand logic out from a net or pin, see *Expanding Pin and Net Logic*, on page 4-62. 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*, on page 4-73.

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 6-28 in the *Reference Manual.*

1. To expand and view connections between selected objects, do the following:
   - Select two or more points.
   - To expand the logic at the current level only, select HDL Analyst->Current Level->Expand Paths or popup menu->Current Level Expand Paths.
   - To expand the logic at the current level and below, select HDL Analyst->Hierarchical->Expand Paths or popup menu->Expand Paths.



2. To view connections from all pins of a selected instance, right-click and select Isolate Paths from the popup menu.

Starting Point	The Filtered View Traces Paths (Forward and Back) From All Pins of the Selected Instance...
Filtered view	Traces through all sheets of the filtered view, up to the next port, register, hierarchical instance, or black box.
Unfiltered view	Traces paths on the current schematic sheet only, up to the next port, register, hierarchical instance, or black box.

Unlike the Expand Paths command, the connections are based on the schematic used as the starting point; the software does not add any objects that were not in the starting schematic.

Flattening Schematic Hierarchy

Flattening removes hierarchy so you can view the logic without hierarchical levels. In most cases, you do not have to flatten your hierarchical schematic to debug and analyze your design, because you can use a combination of filtering, Push/Pop mode, and expanding to view logic at different levels. However, if you must flatten the design, use the following techniques., which include flattening, dissolving, and hiding instances.

1. To flatten an entire design down to logic cells, use one of the following commands:

   – For an RTL view, select HDL Analyst->RTL->Flattened View. This flattens the design to generic logic cells.

   – For a Technology view, select Flattened View or Flattened to Gates View from the HDL Analyst->Technology menu. Use the former command to flatten the design to the technology primitive level, and the latter command to flatten it further to the equivalent Boolean logic.

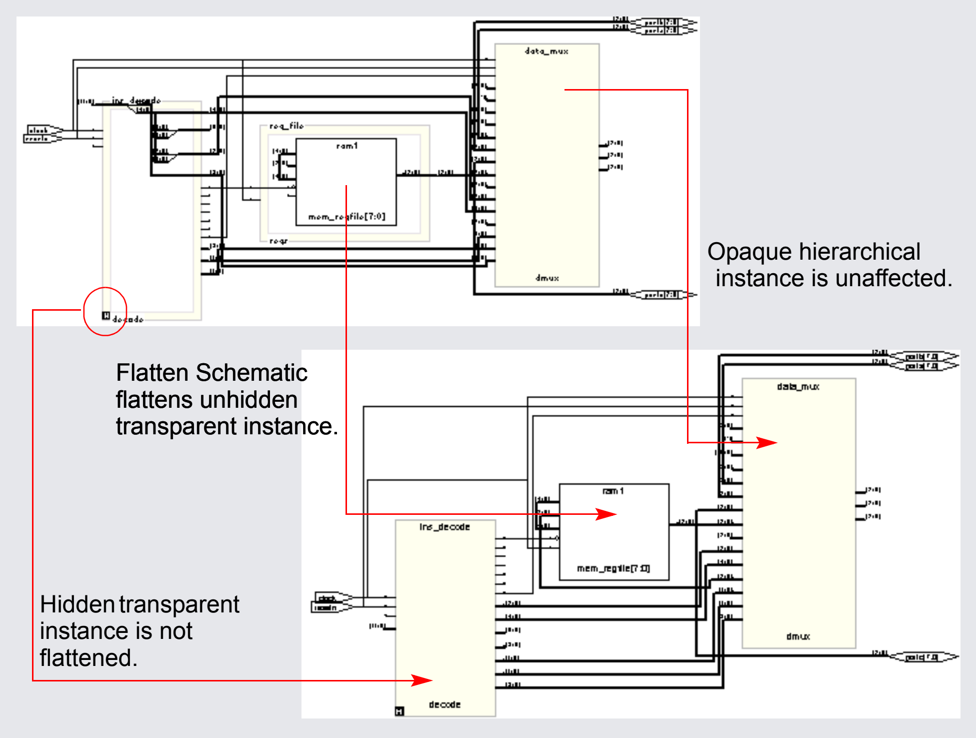
   The software flattens the top-level design and displays it in a new window. To return to the top-level design, right-click and select Unflatten Schematic.

   Unless you really require the entire design to be flattened, use Push/Pop mode and the filtering commands (*Filtering Schematics,* on page 4-60) 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.


data_mux
dmux
reg_file
ram1
mem_regfile[7:0]
ins_decode
decode
clock
resetn
Opaque hierarchical instance is unaffected.
Flatten Schematic flattens unhidden transparent instance.
Hidden transparent instance is not flattened.

Because the flattened view is a new view, you cannot use Back to return to the unflattened view or the views before it. Use Unflatten Schematic to return to the unflattened top-level view.

3. To selectively flatten the design by hiding instances, select hierarchical instances whose hierarchy you do not want to flatten, right-click, and select Hide Instances. Then flatten the hierarchy using one of the Flatten commands described above.

   Use this technique if you want to flatten most of your design. If you want to flatten only part of your design, use the approach described in the next step.

   When you hide instances, the software generates a new view where the hidden instances are not flattened, but marked with an H in the lower left corner. The rest of the design is flattened. If unhidden hierarchical instances are not flattened by this procedure, use the Flattened View or Flattened to Gates View commands described in step 1 instead of the Flatten

Current Schematic command described in step 2, which only flattens transparent instances in filtered views.

You can select the hidden instances, right-click, and select Unhide Instances to make their hierarchy accessible again. To return to the unflattened top-level view, right-click in the schematic and select Unflatten Schematic.

4. To selectively flatten some hierarchical instances in your design by dissolving them, do the following:

   – If you want to flatten more than one level, select Options->HDL Analyst Options and change the value of Dissolve Levels. If you want to flatten just one level, leave the default setting.

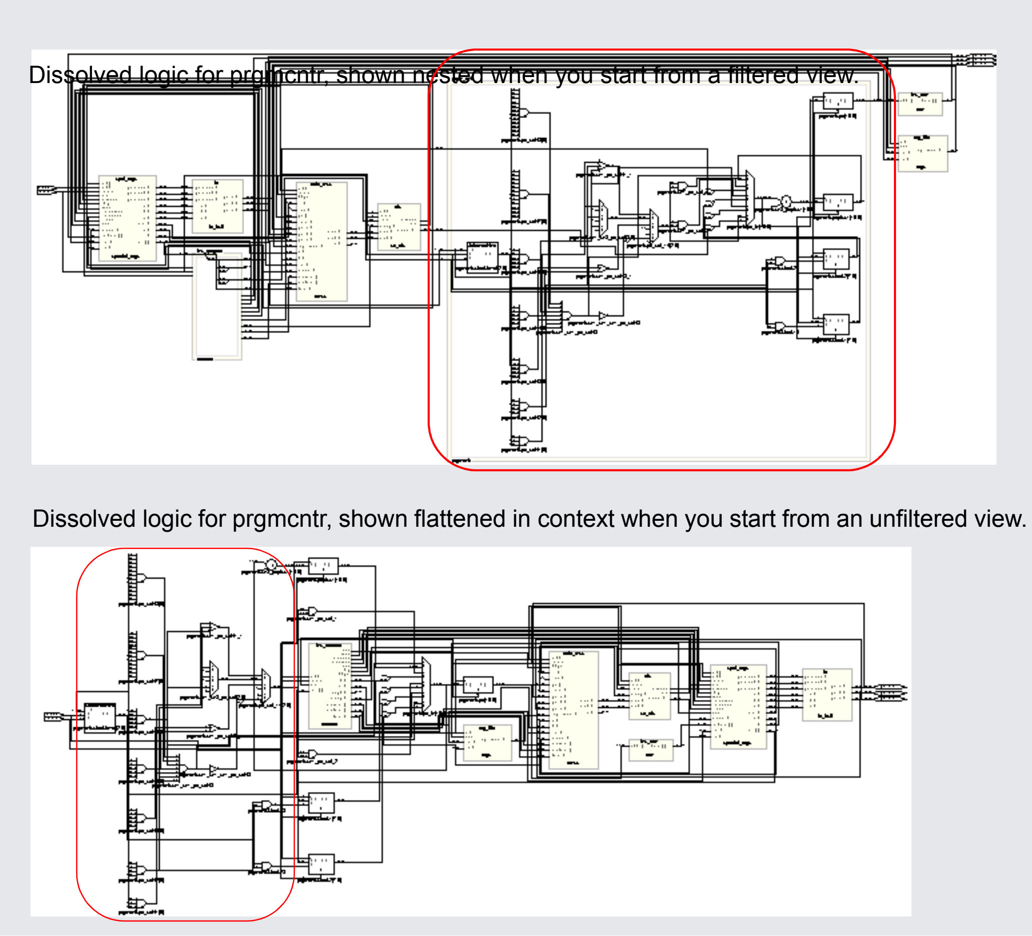
   – Select the instances to be flattened.

   – Right-click and select Dissolve Instances.

   The results differ slightly, depending on the kind of view from which you dissolve instances.

Starting View	**Software Generates a...**
Filtered	Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of *Viewing Design Hierarchy and Context,* on page 4-56. Use the Back button to return to the undissolved view.
Unfiltered	New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. Select Unflatten Schematic to return to the top-level unflattened view. You cannot use the Back button to return to previous views because this is a new view.

The following figure illustrates this.



Dissolved logic for prgmcntr, shown nested when you start from a 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 4-67 for details. For more information on the Expand Paths and Isolate Paths commands, see *RTL View and Technology View Popup Menu Commands,* on page 3-129 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.

Analyzing Timing

You can use the Timing Analyst and HDL Analyst functionality to analyze timing. This section describes the following:

- Analyzing Clock Trees in the RTL View, next
- Viewing Critical Paths, on page 4-74
- Analyzing Paths with the Timing Analyst, on page 4-76
- Analyzing Paths with the Synplify Premier Timing Analyst, on page 4-79
- Handling Negative Slack, on page 4-81

Synplify Premier users can also use island timing reports for analysis, as described in *The Island Timing Report*, on page 4-83.

Analyzing Clock Trees in the RTL View

To analyze clock trees in the RTL view:

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 4-60, *Expanding Pin and Net Logic*, on page 4-62 and *Expanding and Viewing Connections*, on page 4-66. For more information on filtering schematics, see *HDL Analyst Menu: Filtering and Flattening Commands*, on page 3-73 of the *Reference Manual*.

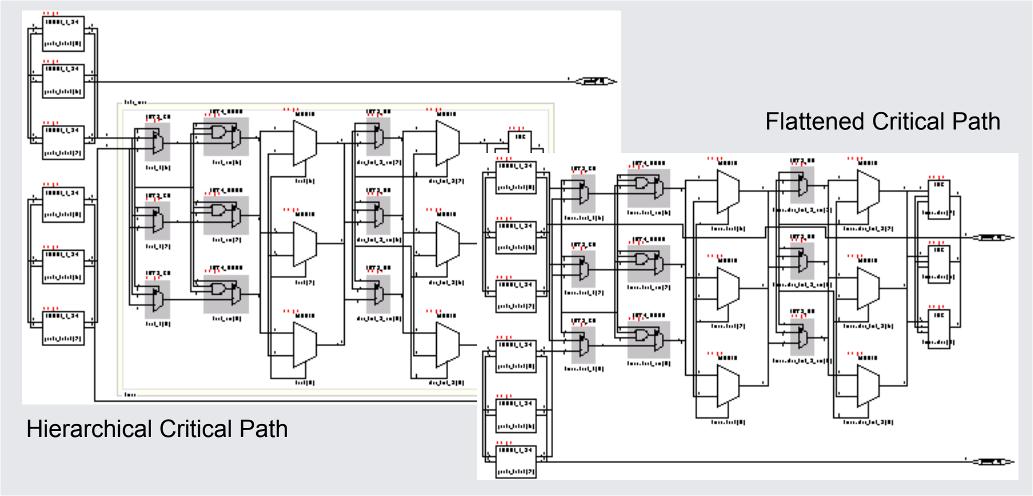
3. Check that your defined clock constraints cover the objects in the design.

   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 *Analyzing Paths with the Timing Analyst,* on page 4-76 and *Analyzing Paths with the Synplify Premier Timing Analyst,* on page 4-79 to view this and other paths.

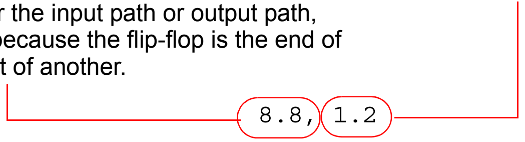
1. If needed, set the slack time for your design.
   - Select HDL Analyst->Set Slack Margin.
   - To view only instances with the worst-case slack time, enter a zero.
   - To set a slack margin range, type a value for the slack margin, and click OK. The software gets a range by subtracting this number 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 4-81.

   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.

Analyzing Paths with the Timing Analyst

The Synplify Pro and Synplify Premier Timing Analyst is an analysis tool that lets you analyze the timing between any two points in applicable Actel, Altera and Xilinx technologies, without resynthesizing your design. You can use this tool for any path, but for critical paths it is easier to use the procedure described in *Viewing Critical Paths,* on page 4-74. The following procedure describes how to use the Timing Analyst; see *Analyzing Paths with the Synplify Premier Timing Analyst,* on page 4-79 for information about the Synplify Premier tool.

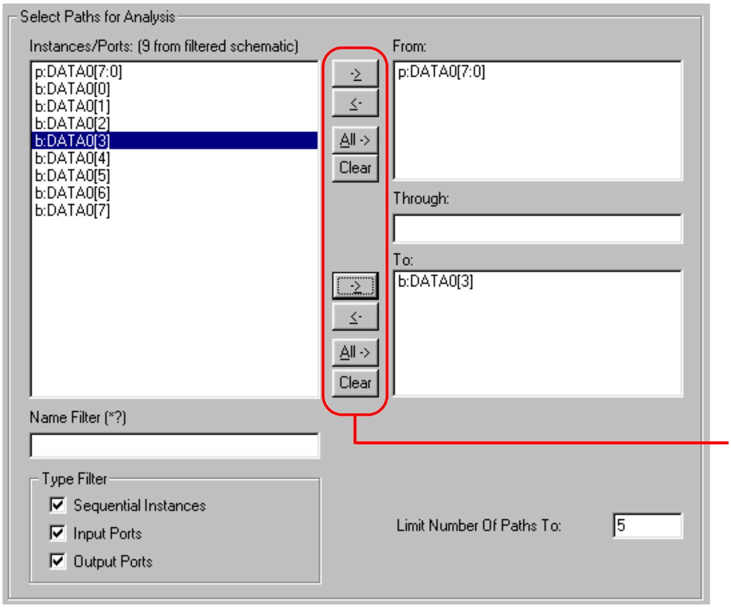
1. Open a Technology view. You can only analyze paths in a mapped design.

2. Select instances from the schematic or use Find to find and select start and end registers or ports. You can filter or expand your design to find start and end points at different levels of hierarchy.

3. Open the Timing Analyst window by clicking the icon or by selecting HDL Analyst->Timing Analyst.

   The Timing Analyst window opens with the selected start and end points in the box on the left side. If you did not select start and end points, it shows all the signal names. For detailed descriptions of the options in this window, see *Timing Analyst Command*, on page 3-80 in the *Reference Manual*.

4. Select the objects and move them into the start and end point boxes using the appropriate arrows.



Select Paths for Analysis

Instances/Ports: (9 from filtered schematic)

p:DATA0[7:0]
b:DATA0[0]
b:DATA0[1]
b:DATA0[2]
b:DATA0[3]
b:DATA0[4]
b:DATA0[5]
b:DATA0[6]
b:DATA0[7]

From: p:DATA0[7:0]

Through:

To: b:DATA0[3]

Name Filter (*?)

Type Filter
☑ Sequential Instances
☑ Input Ports
☑ Output Ports

Limit Number Of Paths To: 5

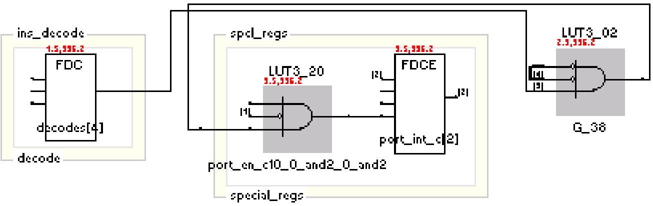
Use arrows to move objects and specify start and end points

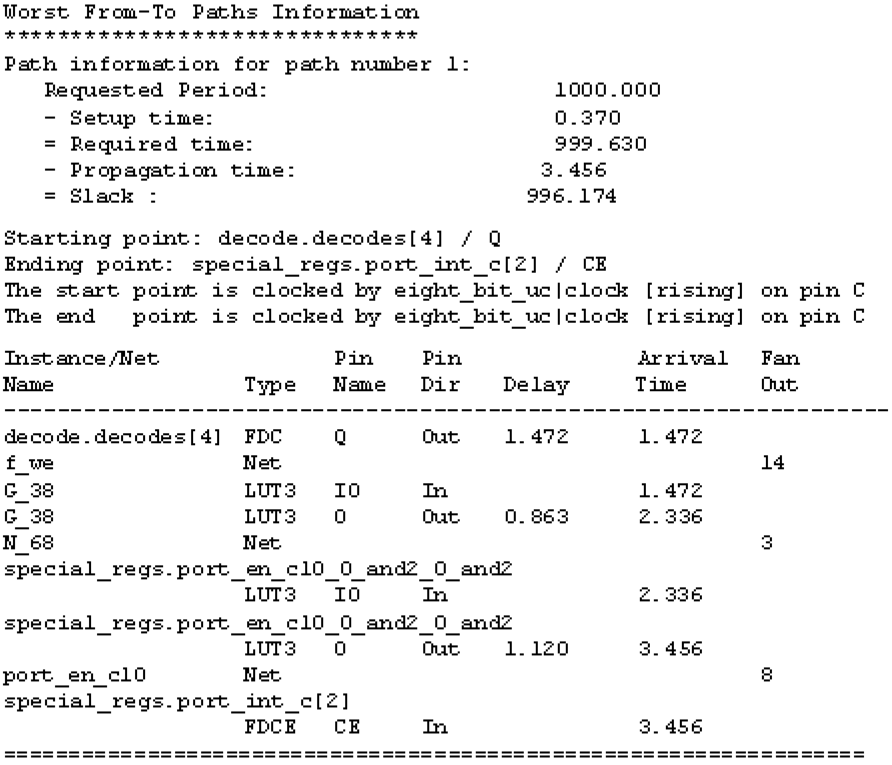
5. Click Generate.

   The software generates and opens a timing report and a timing view schematic. The timing report (`.ta` file in the Implementation Results view) contains from-to information for just the path you specified, and is different from the timing report for the entire design that is in the log file. The timing view schematic (_ta.srm) is a filtered Technology view that shows the path between the start and end points You cannot generate a critical path or use the Timing Analyst from this view. If you close the report and schematic windows, reopen them by selecting the `.ta` (timing report) and *name*`_ta.srm` (Timing view) files from the Implementation Results view.

6. View the results.

   The following figure shows a timing view for a path; the file excerpt that follows shows associated details from the timing report.





```
Worst From-To Paths Information
*******************************
Path information for path number 1:
   Requested Period:                      1000.000
   - Setup time:                          0.370
   = Required time:                       999.630
   - Propagation time:                   3.456
   = Slack :                             996.174

Starting point: decode.decodes[4] / Q
Ending point: special_regs.port_int_c[2] / CE
The start point is clocked by eight_bit_uc|clock [rising] on pin C
The end   point is clocked by eight_bit_uc|clock [rising] on pin C

Instance/Net                 Pin    Pin                Arrival    Fan
Name                  Type   Name   Dir    Delay       Time       Out
-------------------------------------------------------------------
decode.decodes[4]     FDC    Q      Out    1.472       1.472
f_we                  Net                                         14
G_38                  LUT3   I0     In                 1.472
G_38                  LUT3   O      Out    0.863       2.336
N_68                  Net                                         3
special_regs.port_en_c10_0_and2_0_and2
                      LUT3   I0     In                 2.336
special_regs.port_en_c10_0_and2_0_and2
                      LUT3   O      Out    1.120       3.456
port_en_c10           Net                                         8
special_regs.port_int_c[2]
                      FDCE   CE     In                 3.456
=================================================================
```

7. Use the HDL analyst commands described in *Analyzing With the HDL Analyst Tool,* on page 4-56 to analyze the path.

Analyzing Paths with the Synplify Premier Timing Analyst

The Timing Analyst feature is available for Altera and Xilinx technologies. This feature allows you to do point-to-point timing analysis without resynthesizing your design. You can use this tool to generate a custom timing report for any user-specified path.

A timing report can be generated by the timing analyst at various times after the design has been mapped, and based on the type of Synplify Premier process flow implemented. You can request a timing report after creating an implementation:

- Without a design plan
- With a deisgn plan
- With a design plan and fully placed regions enabled

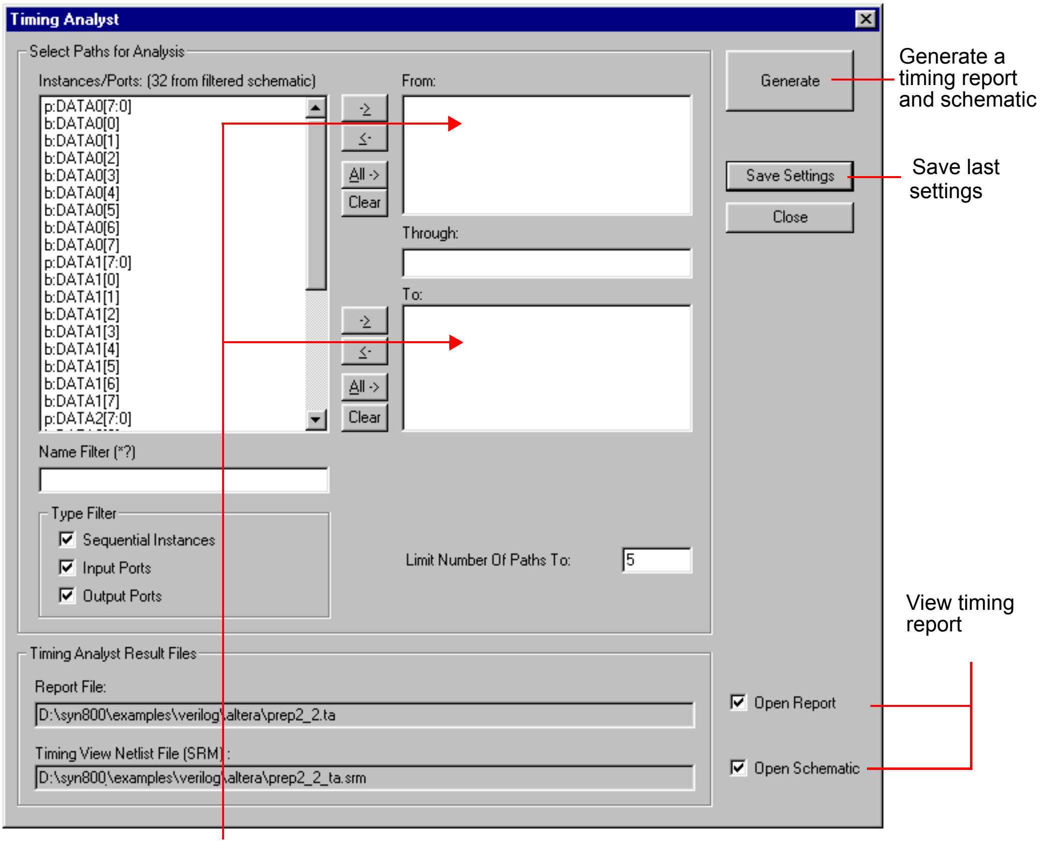
Applicable timing information is used for each type of implementation. For some implementations, exact placement and net delay information is included in the calculations resulting in more accurate timing reports.

Creating the Timing Report

To generate a timing report, do the following:

1. Open a Technology view. You can only analyze paths in a mapped design.
2. Select instances from the schematic or use the Find command to find and select start and end registers or ports. You can also filter or expand your design to find start and end points at different levels of hierarchy.
3. Open the Timing Analyst window by clicking the icon or by selecting HDL Analyst->Timing Analyst from the popup menu.
4. Select the objects and move them into the start and end point boxes using the appropriate arrows.

You can specify any of the following: From points, To points, or From and To points.



Timing Analyst

Select Paths for Analysis

Instances/Ports: (32 from filtered schematic)

```
p:DATA0[7:0]
b:DATA0[0]
b:DATA0[1]
b:DATA0[2]
b:DATA0[3]
b:DATA0[4]
b:DATA0[5]
b:DATA0[6]
b:DATA0[7]
p:DATA1[7:0]
b:DATA1[0]
b:DATA1[1]
b:DATA1[2]
b:DATA1[3]
b:DATA1[4]
b:DATA1[5]
b:DATA1[6]
b:DATA1[7]
p:DATA2[7:0]
```

From:

Through:

To:

Name Filter (*?)

Type Filter

- ☑ Sequential Instances
- ☑ Input Ports
- ☑ Output Ports

Limit Number Of Paths To: 5

Timing Analyst Result Files

Report File: D:\syn800\examples\verilog\altera\prep2_2.ta

Timing View Netlist File (SRM) : D:\syn800\examples\verilog\altera\prep2_2_ta.srm

☑ Open Report

☑ Open Schematic

Generate a timing report and schematic

Save last settings

View timing report

Use arrow buttons to move objects to From/To points.

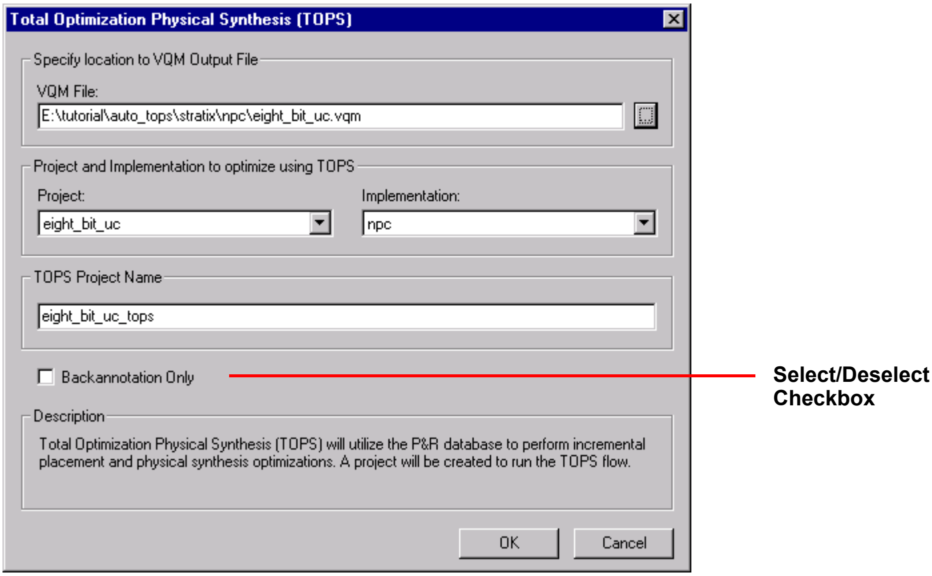
5. Click Generate.

   The software generates and opens a timing report and a Timing view schematic. The timing report (`.ta` file) contains from-to information for just the path you specified, and is different from the timing report for the entire design that is in the log file. The timing view is a filtered Technology view that shows the path between the start and end points. By default, the software filters Sequential Instances, Input Ports, and Output Ports. You can also enter a limit for the number of paths to display.

6. View the results.

   By default, the software opens the `.ta` timing report file and the Timing view for the path. If you close these windows, reopen them by selecting the `.ta` (timing report) and `.srm` (Timing view) files from the Implementation Results view. See *Timing Report*, on page 7-71 in the *Reference Manual*.

7. Use the HDL analyst commands to analyze the path.



Total Optimization Physical Synthesis (TOPS)

Specify location to VQM Output File

VQM File: E:\tutorial\auto_tops\stratix\npc\eight_bit_uc.vqm

Project and Implementation to optimize using TOPS

Project: eight_bit_uc

Implementation: npc

TOPS Project Name: eight_bit_uc_tops

□ Backannotation Only

Description

Total Optimization Physical Synthesis (TOPS) will utilize the P&R database to perform incremental placement and physical synthesis optimizations. A project will be created to run the TOPS flow.

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 *Filtering Schematics*, on page 4-60, *Expanding Pin and Net Logic*, on page 4-62, and *Expanding and Viewing Connections*, on page 4-66. For more information on filtering schematics, see *Filtering Schematics*, on page 4-60.

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
   - Enabling options like pipelining (*Pipelining*, on page 6-40), retiming (*Retiming*, on page 6-44), FSM exploration (*Using FSM Explorer*, on page 6-22), or resource sharing. The Synplify product does not support pipelining, retiming, and FSM exploration.
   - Modifying the source code.

5. Rerun synthesis and check your results.

The Island Timing Report

After you synthesize a design, you can generate a timing report that contains a hierarchical display for groups of connected critical paths called islands. This timing report can be generated for certain target Xilinx and Altera technologies. The island timing report is useful for creating a design plan and analyzing critical paths (routing vs. logic delay), because it identifies which instances or pins belong to multiple paths and how the critical paths in an island group are connected together. Critical paths with a large percentage of total route delay typically have better improvements with design planning. For details about the information in this report, see *Synplify Premier Island Timing Report*, on page 7-76 in the *Reference Manual*. The topics in this section include:

- Generating the Island Timing Report, on page 4-83
- Automatic Island Timing Report, on page 4-84
- Defining the Group Range and Global Range, on page 4-85
- Interactive Island Timing Analyst, on page 4-86
- Viewing the Island Timing Report, on page 4-87

Generating the Island Timing Report

In the applicable Xilinx (Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3) and Altera (Stratix, Stratix GX, Stratix II, Cyclone, and Cyclone II) technologies, you can either automatically generate the Island Timing Report during the mapper phase or interactively use the Island Timing Analyst after running physical synthesis. See the following sections:

- Automatic Island Timing Report, on page 4-84
- Defining the Group Range and Global Range, on page 4-85
- Interactive Island Timing Analyst, on page 4-86
- Viewing the Island Timing Report, on page 4-87

Automatic Island Timing Report

Use the following process to automatically generate the hierarchical-based island timing report during the mapper phase.

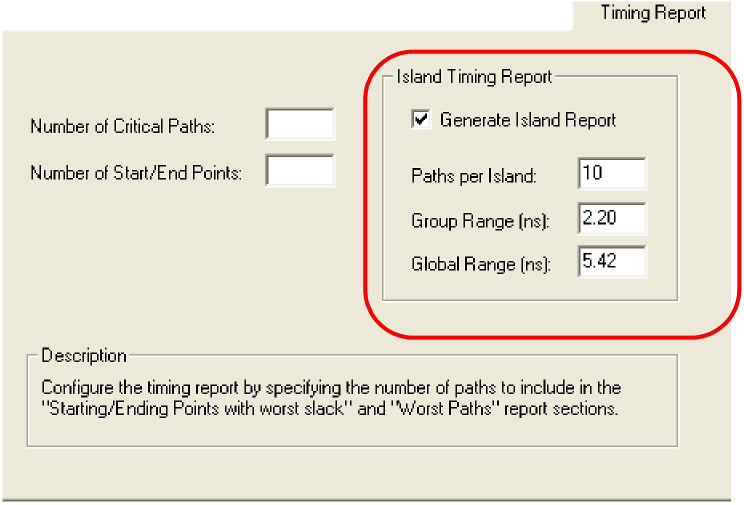
1. Select the Timing Report tab of the Implementation Options panel and in the Island Timing Report section of this pane, check the Generate Island Report switch to enable this option. A timing report can be generated only for devices with this switch.

2. Then you must select a value for the following:

   – Paths per Island — specify the number of paths to report for each island.

   – Group Range (ns) — specify a group range in nano seconds from the worst case slack of the island to determine the critical paths for each island.

   – Global Range (ns) — specify a global range in nano seconds from the worst case slack of the design to determine the number of islands displayed in the timing report.



Number of Critical Paths:

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. After you have set all the implementation option settings, click OK and close the dialog box.

4. 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 4-87 for further information.

Defining the Group Range and Global Range

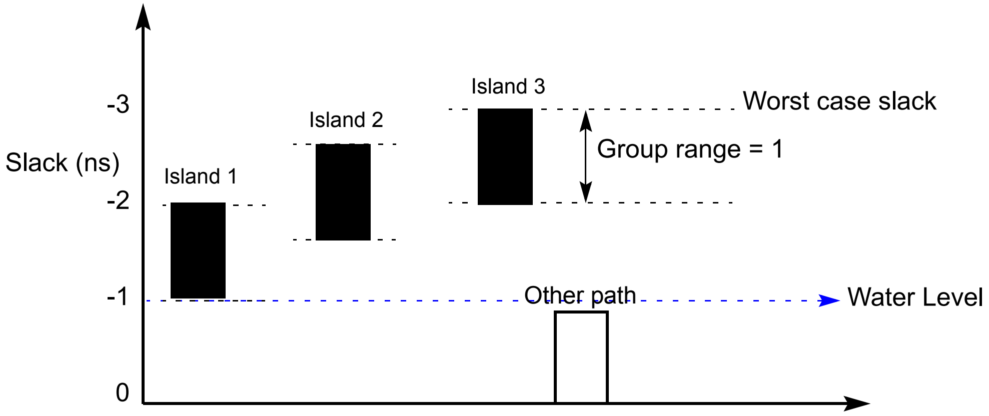
Global Range specifies the lower bound (or water level) for the timing report. The Group Range value specifies the range from the worst case slack, and thus determines the instances that are reported for each individual island.

Example

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).



Interactive Island Timing Analyst

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 the Group Range and Global Range*, on page 4-85.

5. Click on the Generate Report button from the Islands/Paths Control panel.

See *Viewing the Island Timing Report*, on page 4-87 for more detailed information.

Viewing the Island Timing Report

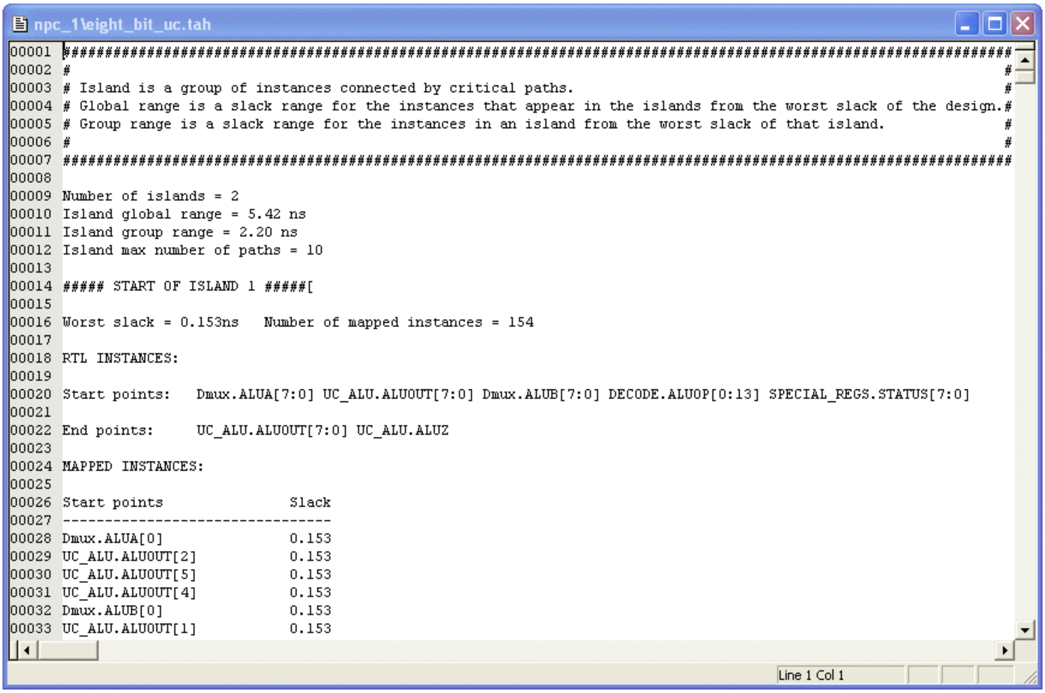
The timing report lists the islands with their worst paths displayed based on the number of paths you requested for the island. For each path, the logic elements and nets and net delays are displayed in an ordered from-to list.

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.



```
#################################################################################################################
#                                                                                                               #
# Island is a group of instances connected by critical paths.                                                   #
# Global range is a slack range for the instances that appear in the islands from the worst slack of the design.#
# Group range is a slack range for the instances in an island from the worst slack of that island.              #
#                                                                                                               #
#################################################################################################################

Number of islands = 2
Island global range = 5.42 ns
Island group range = 2.20 ns
Island max number of paths = 10

##### START OF ISLAND 1 #####[

Worst slack = 0.153ns   Number of mapped instances = 154

RTL INSTANCES:

Start points:   Dmux.ALUA[7:0] UC_ALU.ALUOUT[7:0] Dmux.ALUB[7:0] DECODE.ALUOP[0:13] SPECIAL_REGS.STATUS[7:0]

End points:     UC_ALU.ALUOUT[7:0] UC_ALU.ALUZ

MAPPED INSTANCES:

Start points               Slack
--------------------------------
Dmux.ALUA[0]               0.153
UC_ALU.ALUOUT[2]           0.153
UC_ALU.ALUOUT[5]           0.153
UC_ALU.ALUOUT[4]           0.153
Dmux.ALUB[0]               0.153
UC_ALU.ALUOUT[1]           0.153
```

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 *Islands/Paths Summary View*, on page 4-92 and *Islands/Paths Details View*, on page 4-95. See *Island Timing Report Critical Paths*, on page 4-98 for details about how to use the critical path timing information in this file or the tool for QoR improvements with physical synthesis.

The Island Timing Analyst

The following topics describe the Timing Analyst and its usage:

- Islands/Paths Control Panel, on page 4-89
- Islands/Paths Summary View, on page 4-92
- Islands/Paths Summary Management, on page 4-93
- Islands/Paths Details View, on page 4-95

Use the Island Timing Analyst to generate and display the Islands/Paths Summary and Details reports. You can also cross probe these critical paths to the HDL Analyst view. The Island Timing Analyst contains the following:

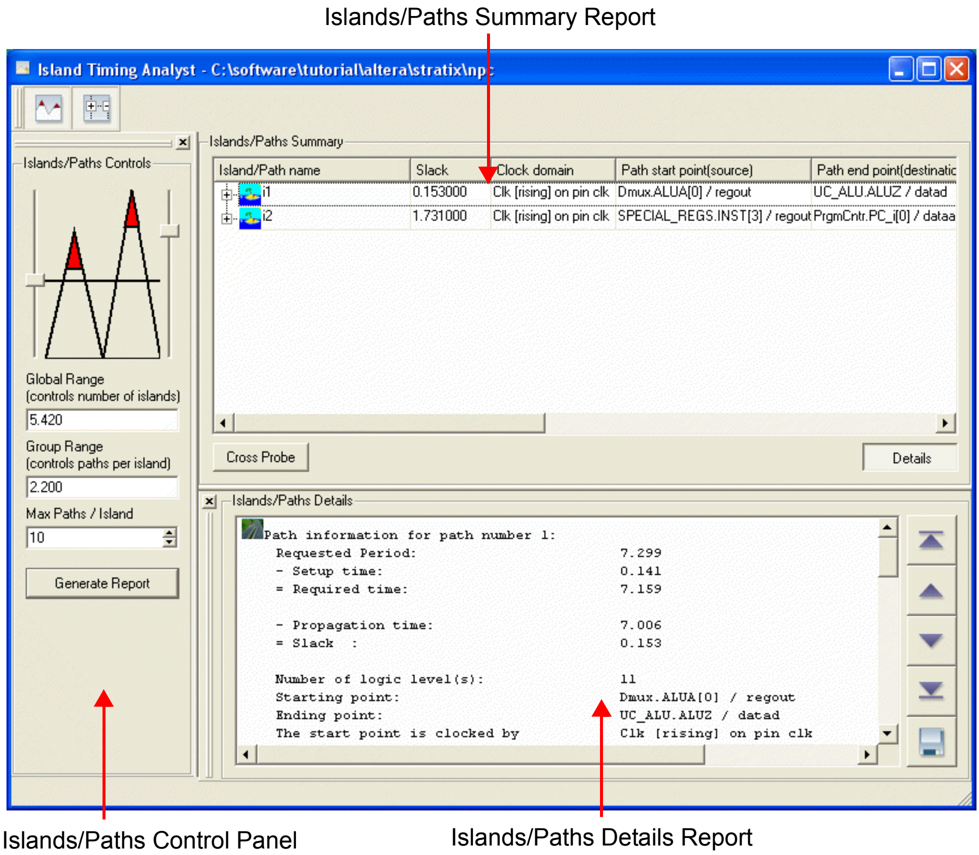
- Islands/Paths Control Panel — use to set values for Global Range, Group Range, and Max Paths/Island and generate the island timing report.
- Islands/Paths Summary View— displays a spread-sheet window that contains all islands and their associated critical paths within the ranges specified for generating the island timing report.
- Islands/Paths Details View — displays detail information for islands or paths selected from the Islands/Paths Summary report within this window.

Refer to the following sections for more information about the:

- Islands/Paths Control Panel, on page 4-89
- Islands/Paths Summary View, on page 4-92
- Islands/Paths Summary Management, on page 4-93

- Islands/Paths Details View, on page 4-95

The following figure shows the UI for the Island Timing Analyst.


Islands/Paths Summary Report
Island Timing Analyst - C:\software\tutorial\altera\stratix\np
Islands/Paths Controls
Islands/Paths Summary
Cross Probe
Details
Islands/Paths Details
Generate Report
Islands/Paths Control Panel
Islands/Paths Details Report

Global Range (controls number of islands): 5.420

Group Range (controls paths per island): 2.200

Max Paths / Island: 10

Island/Path name	Slack	Clock domain	Path start point(source)	Path end point(destinatic
i1	0.153000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad
i2	1.731000	Clk [rising] on pin clk	SPECIAL_REGS.INST[3] / regout	PrgmCntr.PC_i[0] / dataa

```
Path information for path number 1:
  Requested Period:                          7.299
  - Setup time:                              0.141
  = Required time:                           7.159

  - Propagation time:                        7.006
  = Slack   :                                0.153

  Number of logic level(s):                  11
  Starting point:                            Dmux.ALUA[0] / regout
  Ending point:                              UC_ALU.ALUZ / datad
  The start point is clocked by              Clk [rising] on pin clk
```

Islands/Paths Control Panel

The Islands/Paths control panel is displayed by default. You can toggle the display to show or hide this panel in the Island Timing Analyst by:

- Clicking on the Controls icon ( ).
- Using the keyboard shortcut key Ctrl-p.
- Right-click and select Controls from the popup menu.

Note that the Line Up command is currently not applicable in the Island Timing Analyst.

Use the Islands/Paths control panel to generate an island timing report. To do this:

1. You must specify the following parameters either manually in their appropriate parameter fields, or else using the slider controls on the control panel.

   – Global Range (ns) — specify a global range in nano seconds from the worst case slack of the design to determine the number of islands displayed in the timing report. Type the range value in the parameter field or use the slider control.

     Only the islands above this global slack range (water level) are displayed in the island report.

   – Group Range (ns) — specify a group range in nano seconds from the worst case slack of the island to determine the critical paths for each island. Type the range value in the parameter field or use the slider control.

     Only the paths within this group slack range for an island are displayed in the island report.

   – 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.

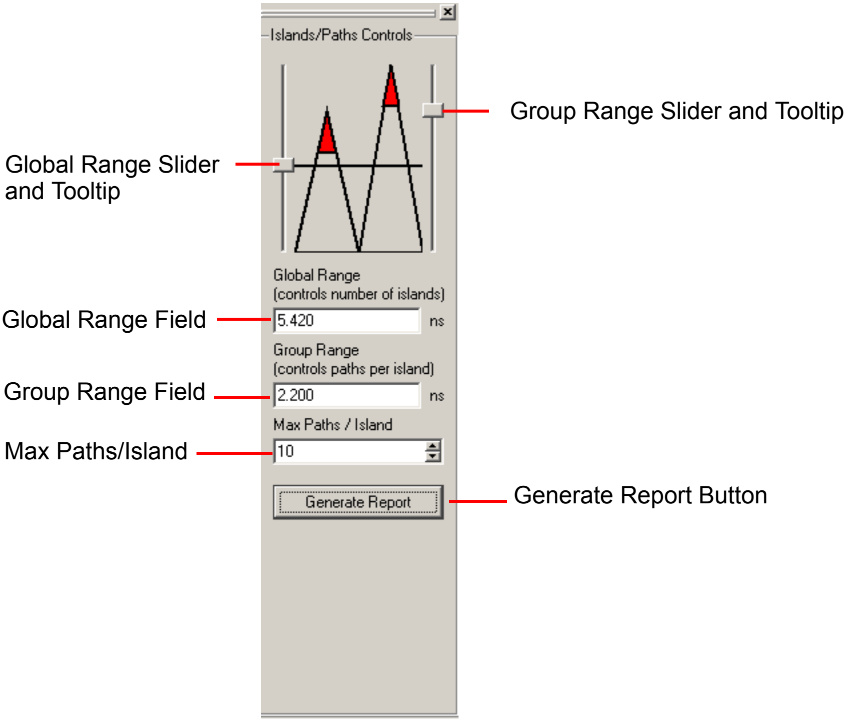
     If the number of paths which do not meet the specified slack for an island exceeds the maximum number of paths specified, then the most critical paths within this limit are displayed in the island timing report.

2. Then click on 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 *The Island Timing Report,* on page 4-83.

The following figure shows the UI for the Islands/Paths control panel.



Islands/Paths Controls

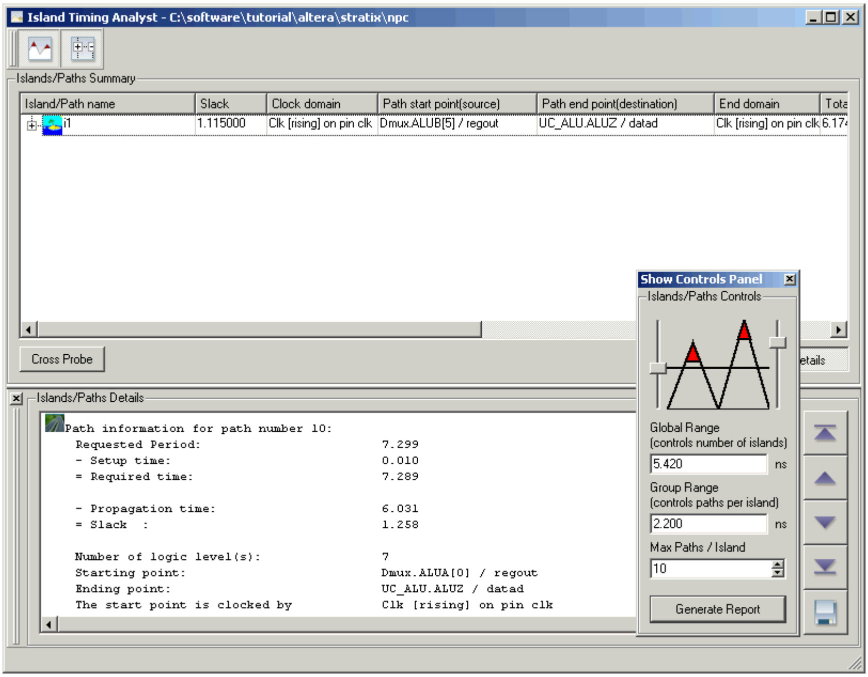
Global Range
(controls number of islands)
5.420 ns

Group Range
(controls paths per island)
2.200 ns

Max Paths / Island
10

To dock the control panel in the Island Timing Analyst to another location in the view, double-click on the top edge of the control panel where the pointer (✥) appears. To reposition, double-click on the header in the control panel. You can also move and then resize this window by selecting and dragging its edges.

The figure below shows the Islands/Paths control panel floating in the view.



Island/Path name	Slack	Clock domain	Path start point(source)	Path end point(destination)	End domain	Tota
i1	1.115000	Clk [rising] on pin clk	Dmux.ALUB[5] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	6.17

```
Path information for path number 10:
  Requested Period:                         7.299
  - Setup time:                             0.010
  = Required time:                          7.289

  - Propagation time:                       6.031
  = Slack  :                                1.258

  Number of logic level(s):                 7
  Starting point:                           Dmux.ALUA[0] / regout
  Ending point:                             UC_ALU.ALUZ / datad
  The start point is clocked by             Clk [rising] on pin clk
```

Global Range (controls number of islands): 5.420 ns

Group Range (controls paths per island): 2.200 ns

Max Paths / Island: 10

Islands/Paths Summary View

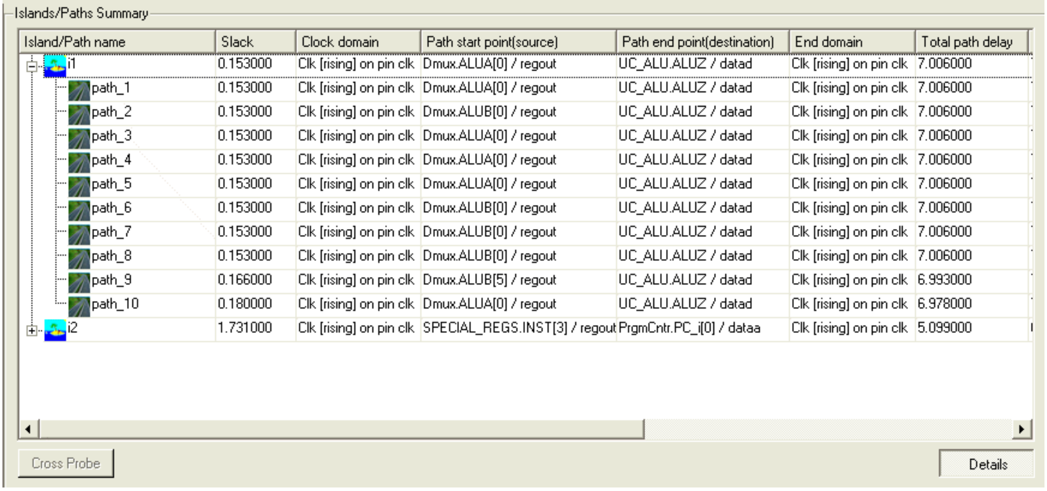
The Islands/Paths Summary is displayed after you generate the island report. The island report is a spread-sheet that contains rows for all hierarchical islands and their critical paths and columns that contain data and calculations for these critical paths. The island rows can be expanded (+) or collapsed (-) to show or hide all the critical paths for the specified island.

The columns consist of the following:

- Island/Path name
- Slack
- Clock domain
- Path start point (source)
- Path end point (destination)

- End domain
- Total path delay
- Path required time
- Logic delay
- Max time allowed for route delay
- Route delay

The following figure show the UI for the Islands/Paths Summary. You must scroll to see all the column information for the critical paths.



Islands/Paths Summary

Island/Path name	Slack	Clock domain	Path start point(source)	Path end point(destination)	End domain	Total path delay
i1	0.153000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_1	0.153000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_2	0.153000	Clk [rising] on pin clk	Dmux.ALUB[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_3	0.153000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_4	0.153000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_5	0.153000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_6	0.153000	Clk [rising] on pin clk	Dmux.ALUB[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_7	0.153000	Clk [rising] on pin clk	Dmux.ALUB[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_8	0.153000	Clk [rising] on pin clk	Dmux.ALUB[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	7.006000
path_9	0.166000	Clk [rising] on pin clk	Dmux.ALUB[5] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	6.993000
path_10	0.180000	Clk [rising] on pin clk	Dmux.ALUA[0] / regout	UC_ALU.ALUZ / datad	Clk [rising] on pin clk	6.978000
i2	1.731000	Clk [rising] on pin clk	SPECIAL_REGS.INST[3] / regout	PrgmCntr.PC_i[0] / dataa	Clk [rising] on pin clk	5.099000

You can choose to group by islands or display all the paths at a flat level. To toggle between these two display modes:

- Click on the Show Islands icon (⊞).
- Use the keyboard shortcut key Ctrl-i.

Islands/Paths Summary Management

In the Islands/Paths Summary view, you can also do the following:

- Sort Columns
- Reorder the Columns

- Select Islands or Critical Paths
- Crossprobe to the HDL Analyst

Sort Columns

You can sort columns in ascending and descending order. To do this, click on the column header. 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

To reorder the columns, drag and drop from the column header to the new location.

Select Islands or Critical Paths

To select islands or critical paths, use the left-mouse button to select either islands or paths in the display. To select multiple islands or paths use either the Shift or Ctrl key with the left-mouse button.

Crossprobe to the HDL Analyst

To crossprobe to the HDL Analyst view, do the following:

- 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.
- Then select islands or critical paths from the Islands/Paths Summary display.
- Click on the Cross Probe button.
- You can then filter critical paths in the HDL Analyst view.
- Use crossprobing from the HDL Analyst view, to see timing data in the Technology view.

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.

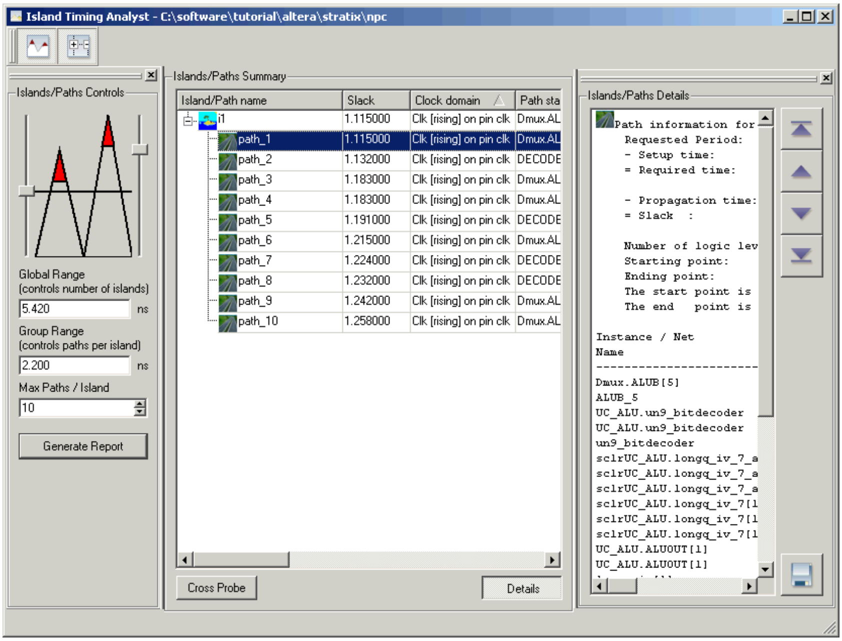
Islands/Paths Details View

The Islands/Paths Details timing report reformats and displays additional information for the islands and critical paths selected from the Islands/Paths Summary report. Critical paths are generated by specifying the sequential elements on the islands as from and to instances. The island critical paths contain the following levels of hierarchy:

- Path properties display information, such as, worst case slack, number of logic levels, start points, and end points. At the end of the timing report for each path, the total path delay (propagation time + setup time) is computed as a ratio of the logic and the routing.
- Logic elements and nets include the following: cell logic element or net, pin name, pin direction, delay, arrival time, and number of fanouts, if applicable.

For more information about the contents of the Island Timing Report, see *Synplify Premier Island Timing Report,* on page 7-76 in the *Reference Manual*.

To dock the Islands/Paths Details view in the Island Timing Analyst to another location in the view, double-click on the left edge of the window where the pointer (✥) appears. To reposition, double-click on the header in the Islands/Paths Details view. You can also move and then resize this window by selecting and dragging its edges. See the following figures:



Island Timing Analyst - C:\software\tutorial\altera\stratix\npc

Islands/Paths Controls

Global Range (controls number of islands): 5.420 ns

Group Range (controls paths per island): 2.200 ns

Max Paths / Island: 10

Generate Report

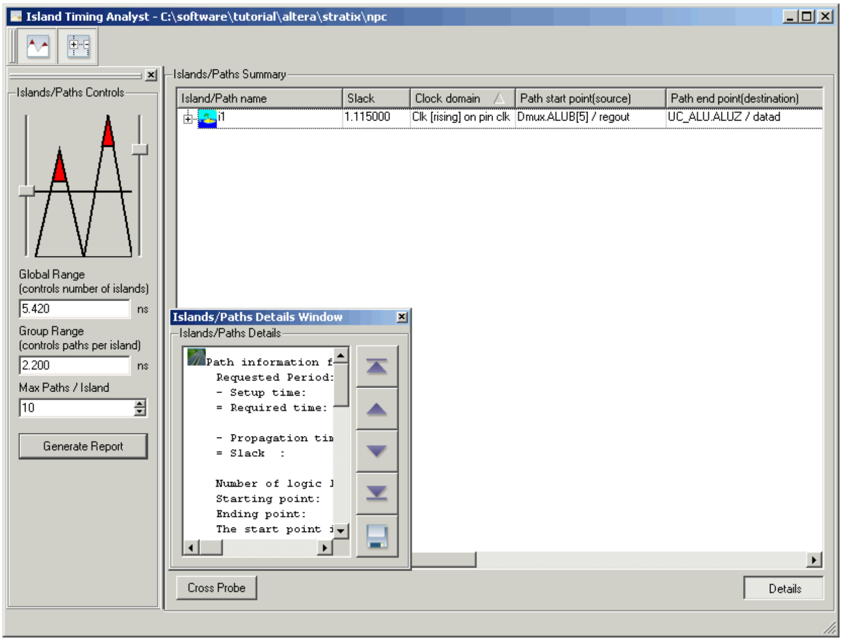
Islands/Paths Summary

Island/Path name	Slack	Clock domain	Path sta
i1	1.115000	Clk [rising] on pin clk	Dmux.AL
path_1	1.115000	Clk [rising] on pin clk	Dmux.AL
path_2	1.132000	Clk [rising] on pin clk	DECODE
path_3	1.183000	Clk [rising] on pin clk	Dmux.AL
path_4	1.183000	Clk [rising] on pin clk	Dmux.AL
path_5	1.191000	Clk [rising] on pin clk	DECODE
path_6	1.215000	Clk [rising] on pin clk	Dmux.AL
path_7	1.224000	Clk [rising] on pin clk	DECODE
path_8	1.232000	Clk [rising] on pin clk	DECODE
path_9	1.242000	Clk [rising] on pin clk	Dmux.AL
path_10	1.258000	Clk [rising] on pin clk	Dmux.AL

Cross Probe   Details

Islands/Paths Details

```
Path information for
  Requested Period:
  - Setup time:
  = Required time:

  - Propagation time:
  = Slack  :

  Number of logic lev
  Starting point:
  Ending point:
  The start point is
  The end   point is

Instance / Net
Name
-----------------------
Dmux.ALUB[5]
ALUB_5
UC_ALU.un9_bitdecoder
UC_ALU.un9_bitdecoder
un9_bitdecoder
sclrUC_ALU.longq_iv_7_a
sclrUC_ALU.longq_iv_7_a
sclrUC_ALU.longq_iv_7_a
sclrUC_ALU.longq_iv_7[1
sclrUC_ALU.longq_iv_7[1
sclrUC_ALU.longq_iv_7[1
UC_ALU.ALUOUT[1]
UC_ALU.ALUOUT[1]
```



Island Timing Analyst - C:\software\tutorial\altera\stratix\npc

Islands/Paths Controls

Global Range (controls number of islands): 5.420 ns

Group Range (controls paths per island): 2.200 ns

Max Paths / Island: 10

Generate Report

Islands/Paths Summary

Island/Path name	Slack	Clock domain	Path start point(source)	Path end point(destination)
i1	1.115000	Clk [rising] on pin clk	Dmux.ALUB[5] / regout	UC_ALU.ALUZ / datad

Islands/Paths Details Window

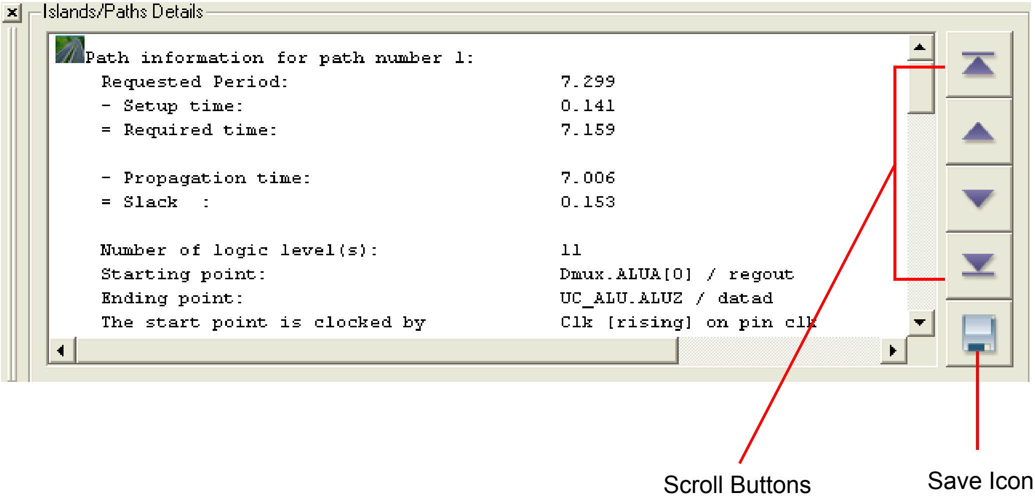
Islands/Paths Details

```
Path information f
  Requested Period:
  - Setup time:
  = Required time:

  - Propagation tim
  = Slack  :

  Number of logic l
  Starting point:
  Ending point:
  The start point i
```

Cross Probe   Details

Use the Details button to hide or show the Islands/Paths Details report. When multiple islands or paths are selected from the Islands/Paths Summary report, you can:

- Use the arrow keys to scroll to the desired island or critical path in the timing report.
- Select and copy island and critical path information from this window to a log file. To do this, click on the save icon (💾) and then add this log file to your project. You can also right-click and select Select All (Ctrl-a), then Copy (Ctrl-c) from the popup menu to a log file.
- Crossprobe to the HDL Analyst view. To do this, click on the Cross Probe button.



```
Path information for path number 1:
  Requested Period:                       7.299
  - Setup time:                           0.141
  = Required time:                        7.159

  - Propagation time:                     7.006
  = Slack   :                             0.153

  Number of logic level(s):               11
  Starting point:                         Dmux.ALUA[0] / regout
  Ending point:                           UC_ALU.ALUZ / datad
  The start point is clocked by           Clk [rising] on pin clk
```

Island Timing Report Critical Paths

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.

Assigning Critical Paths to a Region

The following procedure explains how to assign island critical paths to a region:

1. Synthesize (compile and map) the design.
2. Create a new implementation for the project, which includes a design plan.
3. Click on the New Design Plan icon button (▣) to open the Design Planner view.
4. Open the flattened RTL view.
5. 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.
6. 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 these selected gates in the flattened RTL view. Currently, for crossprobing to work properly in the Island Timing Analyst, open the flattened Technology view also.
7. Right-click and select Expand Paths from the popup menu in the flattened RTL view.
8. 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.

9. Run estimation for any design plans created.
10. Save these assignments to the Synplify Premier design plan file (`.sfp`).

Run synthesis for this implementation with a design plan.

CHAPTER 5

Physical Analyst

This document describes typical analysis tasks using graphical analysis with the Physical Analyst tool. It covers the following:

Synplify Premier Physical Analyst Tool

The Physical Analyst tool provides a visual display of the placement and global routing of the design after running place and route with backannotation.

The Physical Analyst is available in each of the Synplify Premier flows with the following criteria:

- Graph-based Physical Synthesis—Physical Synthesis is enabled without a Design Plan or place-and-route implementation. The tool automatically performs placement with backannotation during the physical synthesis run. For Xilinx Virtex-II Pro, Virtex-4, and Spartan-3.
- Graph-based Physical Synthesis with a Design Plan—Physical Synthesis is enabled with a Design Plan, but without a place-and route-implementation. The tool automatically performs placement with backannotation during the physical synthesis run. For Xilinx Virtex-II Pro, Virtex-4, and Spartan-3.
- Design-plan based Physical Synthesis—Physical Synthesis is enabled with a Design Plan and a place-and-route implementation created with backannotation enabled. For Altera Cyclone, Cyclone-II, Stratix, Stratix-GX, Stratix-II, and Xilinx Virtex-II.

The Physical Analyst includes the following capabilities:

- Display placement information such as cell locations and signal pins.
- Analyze netlists using various commands, such as filter, show critical path, or route all nets.
- Query object and properties of instances and nets.
- Cross probe between the Synplify Premier Physical Analyst and either the HDL Analyst or the source code text file.

This section describes basic procedures you use in the Physical Analyst view. The information is organized into these topics:

- Opening the Physical Analyst View, on page 5-4
- Using the Physical Analyst Control Panel, on page 5-5

The Physical Analyst view uses the following input files:

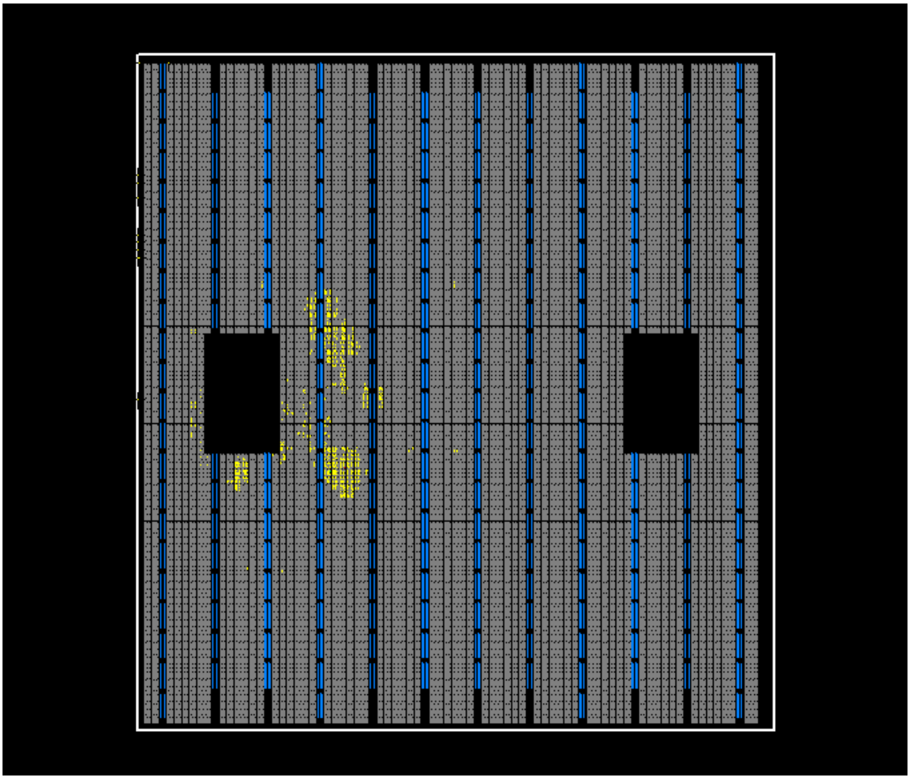
- All appropriate `.lef` files providing physical cell library information for the various devices
- All appropriate `.def` files defined for the device floorplan
- `.srm` file for the netlist and instance placement

Opening the Physical Analyst View

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.

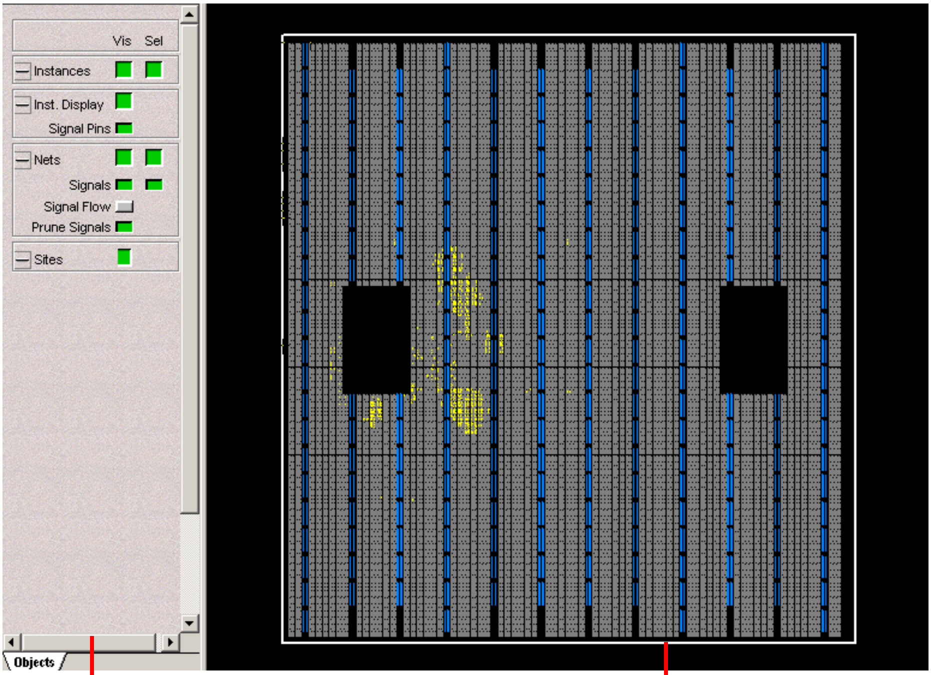
The Physical Analyst view is capable of showing instances and nets. The objects displayed are controlled by the Objects pane of the control panel (see *Using the Physical Analyst Control Panel* on page 5-5). The following figure shows the initial Physical Analyst default view for a Xilinx Virtex2p device.



Using the Physical Analyst Control Panel

The Physical Analyst control panel is an embedded pane displayed on the left side of the Physical Analyst view. The Objects pane is displayed from the tab across the bottom of the control panel. If the control panel is not present, do one of the following to display it:

- Click on the Physical Analyst Control Panel icon (☰) in the Physical Analyst toolbar.
- Select Options->Physical Analyst Control Panel from the menu in this view.
- Use the keyboard shortcut key Ctrl-k.



Control Panel

Device

The control panel includes the Objects tab along the bottom of the pane that contains controls for displaying or hiding objects in the view. You can change the width of the panel by selecting the right side of the pane and dragging it to the desired size.

To close the Physical Analyst control panel:

- Click on the Physical Analyst Control Panel icon (☰) from the Physical Analyst toolbar.
- Select Options->Physical Analyst Control Panel from the menu in this view.
- Toggle off the display using the keyboard shortcut key Ctrl-k.
- Right-click in the control panel pane and select Hide from the popup menu.

Setting Object Controls

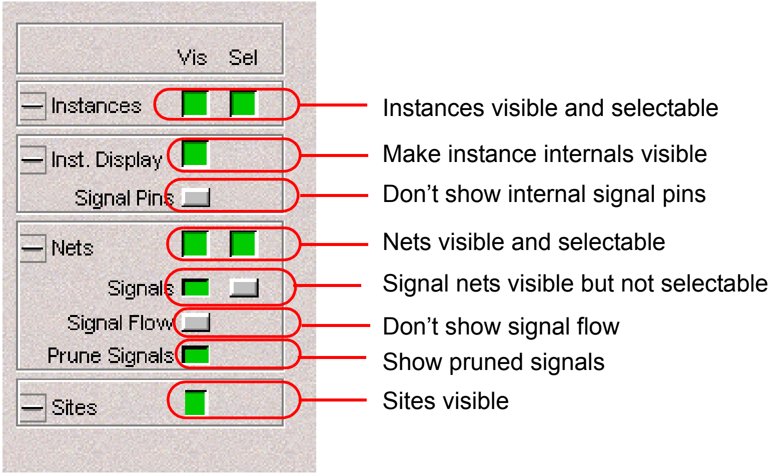
Use the Objects pane of the control panel to set the options for objects you want displayed in the Physical Analyst view. The control panel provides one-click access to the following frequently used commands:

- Object visibility for instances, nets, and sites
- Object selectability for instances, nets, and sites
- Controls for net pruning, signal flow display, and for the display of instance signal pins in the view

Setting Visibility Controls

Object visibility is determined through the control panel's Objects pane. Selections on this pane determine which and how objects are displayed in the Physical Analyst view.

By selecting the appropriate boxes, you can make objects both visible and selectable. For example, if you make instances visible by selecting the Vis box in the Instances section but you do not enable the Sel box, the Physical Analyst view displays the core cell instances within the design, but you are unable to select them. Also, making instances visible makes the cell boundaries visible and making the instances selectable provides more detailed tool tips. This configuration is useful when you are analyzing the critical path. You might want to view all nets and instances along the critical path, and have the nets only visible but not selectable.



Instances visible and selectable

Make instance internals visible

Don’t show internal signal pins

Nets visible and selectable

Signal nets visible but not selectable

Don’t show signal flow

Show pruned signals

Sites visible

- Instances – instances can be made visible or visible and selectable in the device view. Cell instances (Core) are drawn at their placement location and orientation. Unplaced instances are not drawn, but can be located using the Find command. For more information about instances, see *Displaying Instances* on page 5-10.
- Instance Display – signal pins for core instances can be shown or not shown.
- Nets – nets are routed on demand using the command View->Route All Nets. Nets are routed on one metal layer displaying their point-to-point connections from output pins to input pins. Once nets are routed, you can make the nets visible or visible and selectable. Signal Flow adds directional arrows to nets, and Pruned Signals disables the display of signals that are unconnected.
- Sites – as defined in the vendor-specific cell library files. Sites can only be visible or hidden. Use tool tips to display the site row number and its boundaries (Instances must not be selectable).

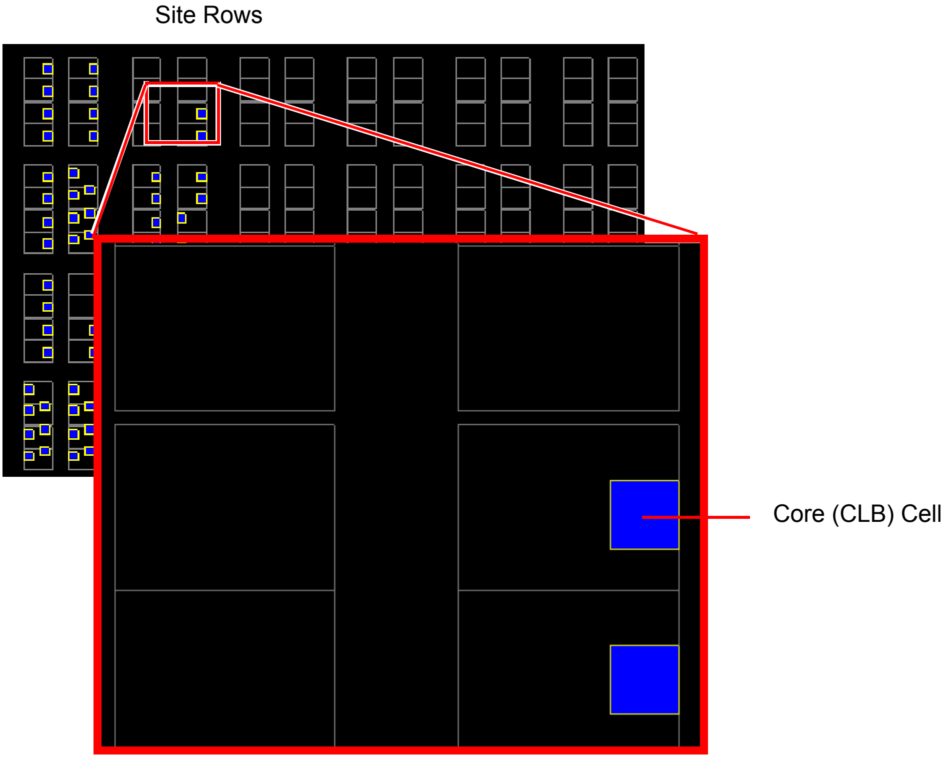
Using the Physical Analyst Device View

The device view shows cell placement and connectivity. The device view displays

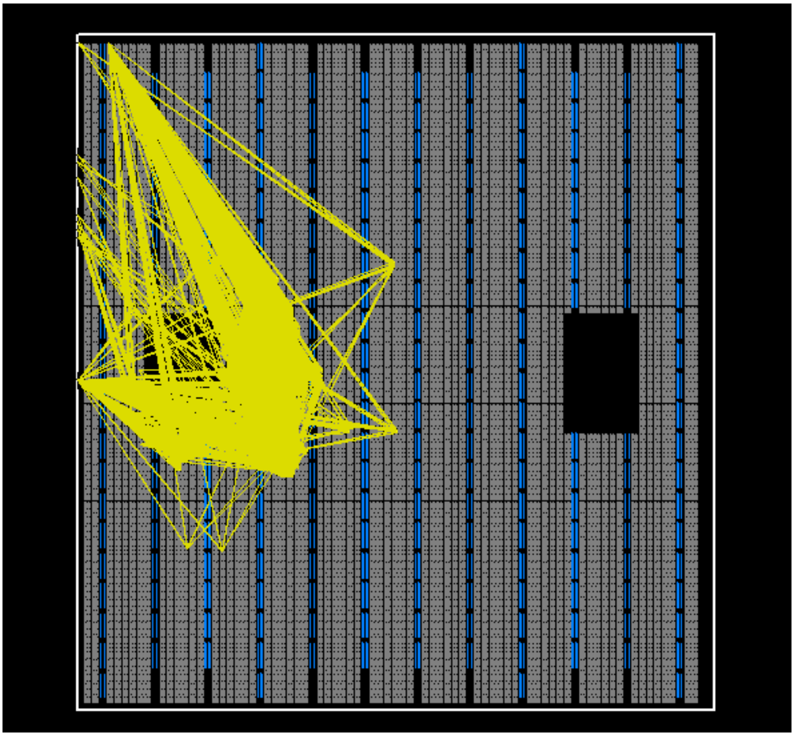
- Device and cell boundaries
- Placement site rows
- Nets

The following figures provide some examples.

Cell Display (Xilinx Device)



Net Display (Route All Nets)



The Physical Analyst view is used to graphically analyze your design. To help you do this, use the information organized into the following topics:

- Setting Object Display Options, on page 5-10
- Selecting Objects, on page 5-14
- Viewing Object Information, on page 5-17
- Finding Objects, on page 5-23
- Crossprobing in Physical Analyst, on page 5-34
- Analyzing Timing with the Physical Analyst, on page 5-51

Setting Object Display Options

This section explains how to display objects for the following:

- Displaying Instances, on page 5-10
- Displaying Signal Pins, on page 5-10
- Displaying Signal Flow for Selected Nets, on page 5-11
- Routing Nets to Display, on page 5-12

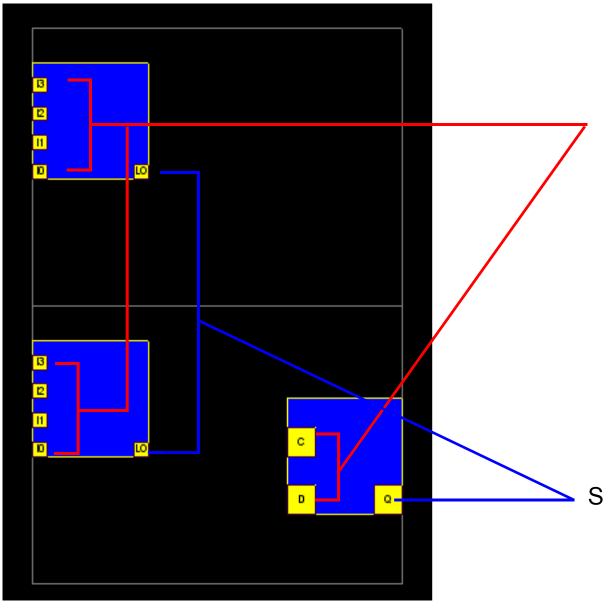
Displaying Instances

To display instances in the Physical Analyst view, first make instances visible by setting the Vis option for objects from the Objects pane of the control panel. Instances that have physical placement information are shown. Instance features include:

- Instance bounds
- Instance locations
- Signal pins

Displaying Signal Pins

Instances are not displayed with signal pins by default. To view signal pins for instances, enable the Signal Pins box on the Objects pane of the control panel.



Signal Pins (Inputs)

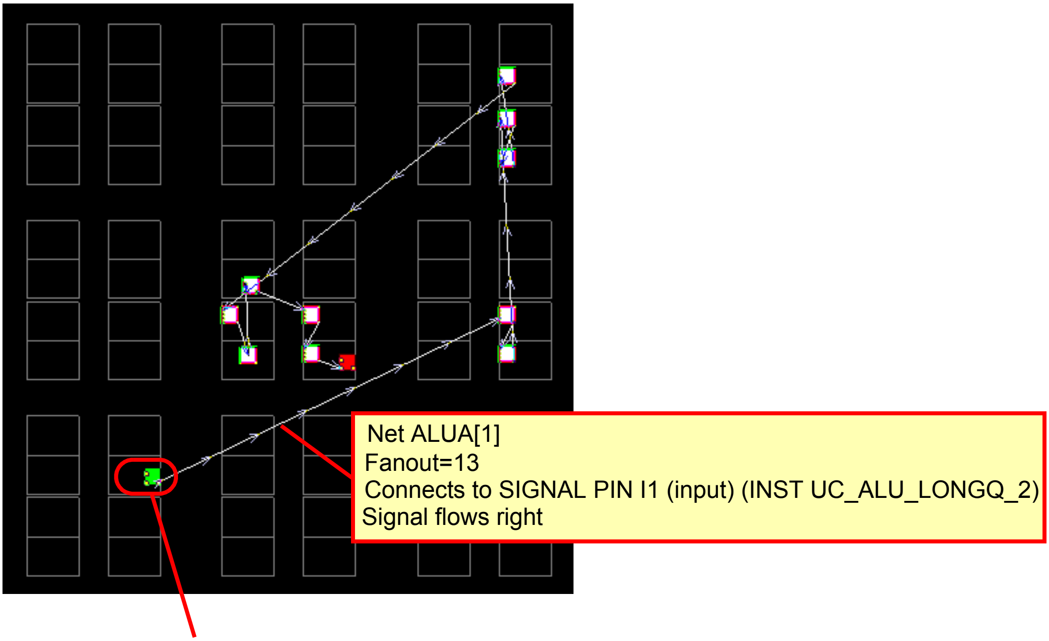
Signal Pins (Outputs)

Displaying Signal Flow for Selected Nets

To display the signal flow of nets in the Physical Analyst view, first make nets visible and selectable. To display the signal flow of nets, right-click and select Signal Flow from the popup menu or control panel.

When you select a net, 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.

Whether or not the Signal Flow option is enabled, place the cursor over a net to show the predominant direction of its signal flow (right, left, up, or down).



Net ALUA[1]
Fanout=13
Connects to SIGNAL PIN I1 (input) (INST UC_ALU_LONGQ_2)
Signal flows right

Critical Path Start

Routing Nets to Display

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. Nets are routed in the following ways:

- On demand using netlist commands such as Expand or Show Critical Path. For more netlist commands, see *Analyzing Netlist with the Physical Analyst* on page 5-42.
- By selecting View->Route All Nets from the menu option. After the nets are routed, this command is grayed out on the menu.

When nets are routed, they are connected to their selective instances. Because of the long load time and the limited visibility when nets are superimposed on the view, net routes are not displayed by default. To enable the display of nets, you must explicitly unfilter (show) the nets or use the find command as an example.

Nets also have a Pruning option. When enabled, a net segment with an end connecting only to an invisible instance (for example because of filtering) is drawn in a diminished color.

To reset the original view to hide all nets in the display, select Unfilter->Show All Instances, Hide All Nets from the popup menu or click the Reset filter icon (⬚→▯) on the Physical Analyst toolbar.

Selecting Objects

This section describes how to select objects, as well as implement the following:

- Selecting Multiple Nets, on page 5-15
- Transcribing Object Selections, on page 5-16

To select an object you must first enable the object to be selectable, and then you can click on the object. To select multiple objects, use one of these methods.

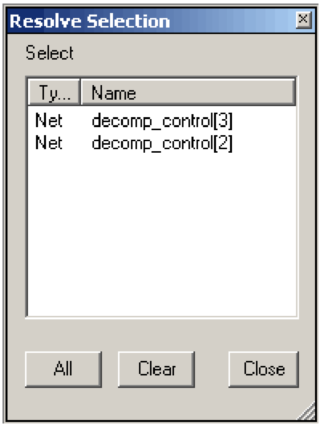
- Draw a rectangle around the objects.
- Select an object, press Ctrl, and click other objects you want to select. You can also deselect from the list of currently selected objects while holding the Ctrl key.
- Position the cursor over an object and click the right mouse button; the object is automatically 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 following from the popup menu:
  - Select->All Instances and Nets
  - Select->All Instances
  - Select->All Nets
  - Select->Deselect All
- 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 (instances, symbols, nets, or ports). See *Finding Objects with the Find Command* on page 5-23.
- Go directly to a coordinate pair location. For example, use the Go to Location command to specify an object location in microns. See *Finding Object Locations* on page 5-27.

The selected objects are highlighted in the Physical Analyst view. If you have other windows open, the selected object is highlighted in the other windows as well (if cross probing is enabled). An exception occurs when you want to cross probe on demand, then you must disable the View->CrossProbing->Send Crossprobes when selecting option. Some commands might affect selection of objects: for example, the Filter or Unfilter commands.

You can enable or disable a class of objects to be selected. For example, you may wish to display nets but not have them selectable. You can only display site rows; they are not selectable.

Selecting Multiple Nets

When you click on objects that overlap, such as nets or instances, the software cannot always detect which objects should be selected in the Physical Analyst view. As you move the cursor over objects, a special cursor indicates that you need to resolve the selection. When you click the cursor at that location, the Resolve Selection dialog box pops up in the display.



Ty...	Name
Net	decomp_control[3]
Net	decomp_control[2]

On the Resolve Selection dialog box, you can choose to

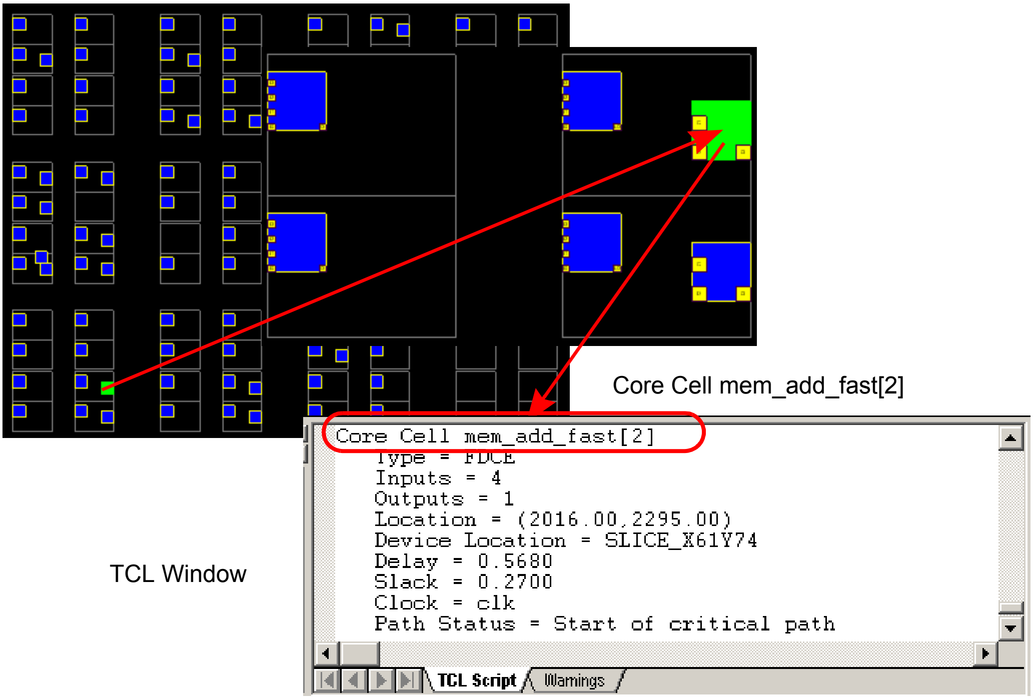
- Display selected net or cell instance
- Select all nets or cell instances in question
- Clear selection of all nets or cell instances

Transcribing Object Selections

The Selection Transcription command allows you to select an object in the Physical Analyst view, then display the object's tool tip information in the TCL window. To enable or disable this command, either:

- Right-click in the Physical Analyst view and select Selection Transcription from the popup menu.
- Select the View->Selection Transcription menu option.

The Selection Transcription command is enabled by default. Transcription only occurs when a single object is selected. Do not use transcription with area selections or when objects are selected using other commands such as Expand or Find. The following figure shows an example of an object selected on the device and its tool tip information displayed in the TCL window.



Core Cell mem_add_fast[2]

TCL Window

```
Core Cell mem_add_fast[2]
   Type = FDCE
   Inputs = 4
   Outputs = 1
   Location = (2016.00,2295.00)
   Device Location = SLICE_X61Y74
   Delay = 0.5680
   Slack = 0.2700
   Clock = clk
   Path Status = Start of critical path
```

You can also use the information from the TCL window display to copy and paste into other windows or files such as SCOPE, the Find Object dialog box, or a text file.

Viewing Object Information

The following Physical Analyst tools can help you use the viewer efficiently:

- Viewing Properties, on page 5-17
- Using Tool Tips, on page 5-20
- Using Mouse Strokes, on page 5-20
- Using Keyboard Shortcuts, on page 5-21
- Zooming in the Physical Analyst, on page 5-22

Viewing Properties

You can view properties for the device design and for selected objects displayed in the view. See:

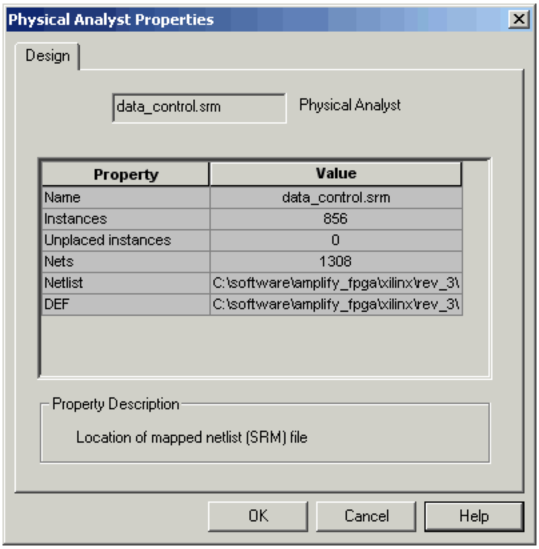
- Viewing Physical Analyst Properties, on page 5-18
- Viewing Object Properties, on page 5-19

Viewing Physical Analyst Properties

Design and device property information is available from the Physical Analyst view. To display the Physical Analyst Properties dialog box, right click anywhere in the view and select Physical Analyst Properties at the bottom of the popup menu.

The dialog box includes the following read-only pane:

- Design – includes the design name and number of instances, unplaced instances, routed nets in the design, and location of the netlist and floorplan (`.def`) files.



data_control.srm Physical Analyst

Property	Value
Name	data_control.srm
Instances	856
Unplaced instances	0
Nets	1308
Netlist	C:\software\amplify_fpga\xilinx\rev_3\
DEF	C:\software\amplify_fpga\xilinx\rev_3\

Property Description

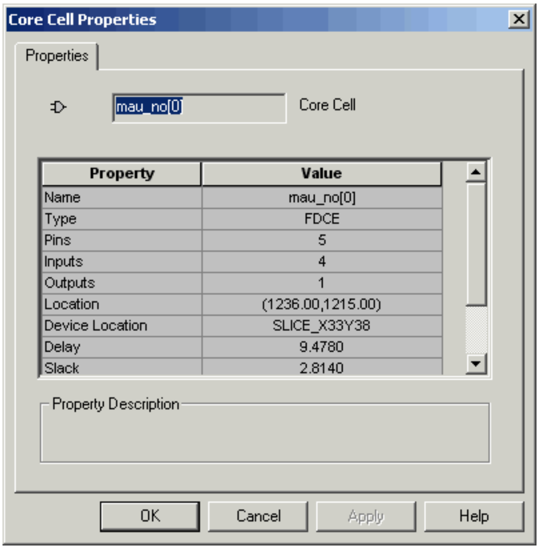
Location of mapped netlist (SRM) file

Viewing Object Properties

You can display properties for a selected object from the corresponding Properties dialog box. For example, depending on the type of object currently selected, right-click and select one of the following property dialog boxes from the popup menu:

- Core Cell Properties for information about instances including instance name, type, pins, placement location, device-specific location, delay, slack, clock signal, and if the instance is included in the critical path.
- Net Properties for information about nets including net name, logical nets, pin count, fanout, if the net is globally routed, and if the net is a clock.

The following example shows the properties for a core cell that is selected in the Physical Analyst view.

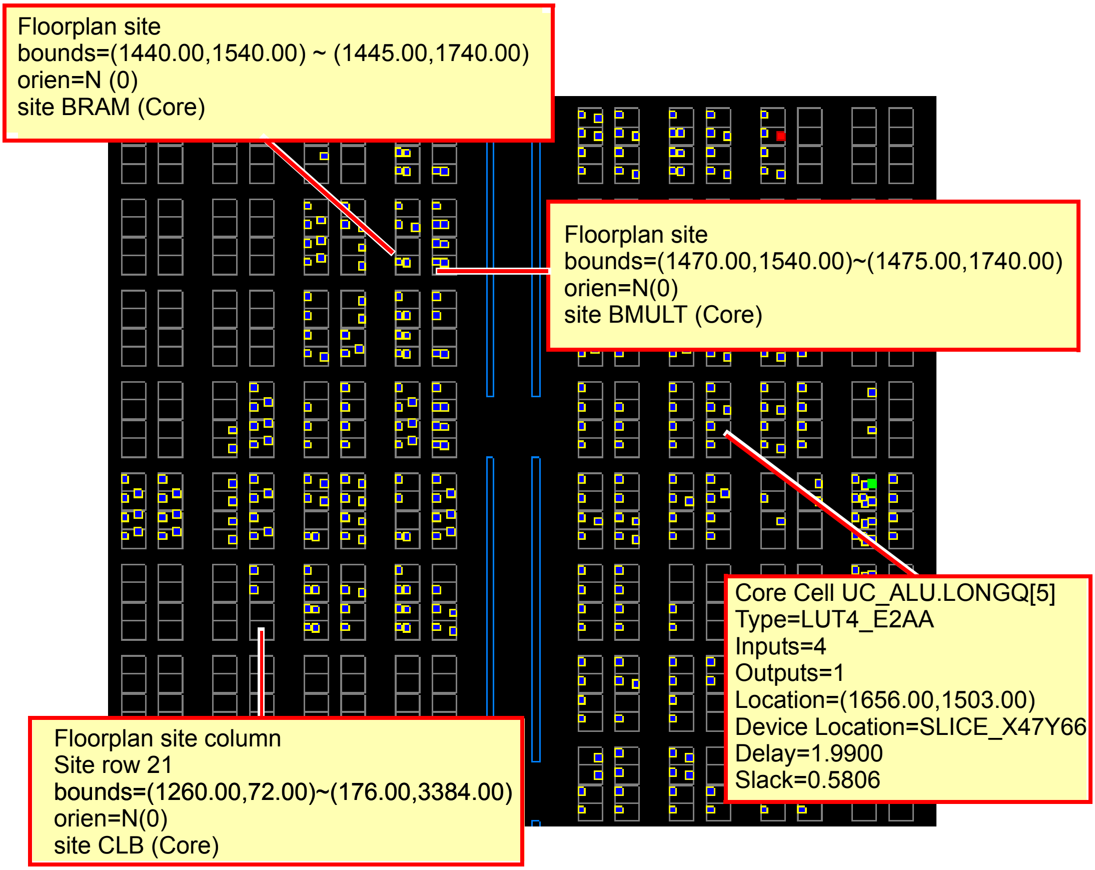


mau_no[0] Core Cell

Property	Value
Name	mau_no[0]
Type	FDCE
Pins	5
Inputs	4
Outputs	1
Location	(1236.00,1215.00)
Device Location	SLICE_X33Y38
Delay	9.4780
Slack	2.8140

Using Tool Tips

As you move the mouse over the viewer, tool tips are displayed for the various features and objects, as well as signal nets. The status bar provides coordinates of the objects in microns. There are also tool tips for Toolbar icons and various check boxes and fields on the Control Panel.



Floorplan site
bounds=(1440.00,1540.00) ~ (1445.00,1740.00)
orien=N (0)
site BRAM (Core)

Floorplan site
bounds=(1470.00,1540.00)~(1475.00,1740.00)
orien=N(0)
site BMULT (Core)

Core Cell UC_ALU.LONGQ[5]
Type=LUT4_E2AA
Inputs=4
Outputs=1
Location=(1656.00,1503.00)
Device Location=SLICE_X47Y66
Delay=1.9900
Slack=0.5806

Floorplan site column
Site row 21
bounds=(1260.00,72.00)~(176.00,3384.00)
orien=N(0)
site CLB (Core)

Using Mouse Strokes

The software supports predefined mouse strokes as shortcuts to common commands. Mouse strokes can be used in Physical Analyst windows for operations like zooming and displaying the previous or next view.

1. To view the current list of mouse stroke operations, select Help->Mouse Stroke Tutor. Select the operation from the list on the left and the corresponding mouse stroke is drawn on the right.
2. In a Physical Analyst window, hold down the right mouse button and draw the mouse stroke.

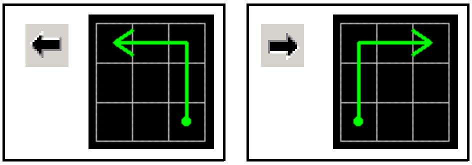
Some mouse strokes only apply to HDL Analyst commands such as push and pop hierarchy. These mouse strokes are ignored while using the Physical Analyst viewer.

Moving Between Views in a Window

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.

   The software displays the last view, including the zoom factor. This does not work in a newly generated view because there is no history.

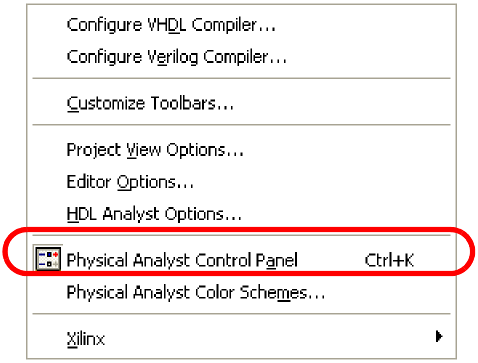
   

2. To move forward again, click the Forward icon or draw the appropriate mouse stroke.

   The software displays the next view in the display history.

Using Keyboard Shortcuts

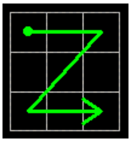
Keyboard shortcuts are key sequences that you type in order to run a command. Menus list keyboard shortcuts next to the corresponding commands. For example, to display the Physical Analyst control panel, you can press and hold the Ctrl key and the letter K, instead of using the menu command Options->Physical Analyst Control Panel as shown in the following example.


Configure VHDL Compiler...
Configure Verilog Compiler...
Customize Toolbars...
Project View Options...
Editor Options...
HDL Analyst Options...
Physical Analyst Control Panel Ctrl+K
Physical Analyst Color Schemes...
Xilinx

Zooming in the Physical Analyst

Since the objects displayed in the Physical Analyst full view appear 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

- Clicking the Zoom Selected ( ) icon.
- Right-click and selecting Zoom Selected from the popup menu.
- Using the following mouse stroke. See *Using Mouse Strokes* on page 5-20.



The object or objects selected are centered in the view.

When objects are not selected, the Zoom Selected command is disabled; you can 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 3-22 in the *Reference Manual*. For a description of the mouse strokes, see Help->Mouse Stroke Tutor.

Finding Objects

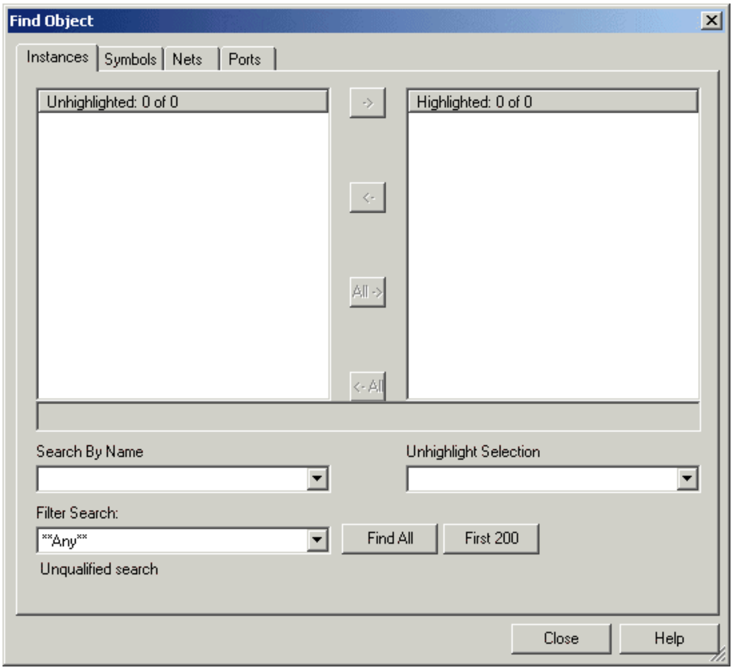
To find and display objects in the Physical Analyst view, use the following options:

- Finding Objects with the Find Command, on page 5-23
- Finding Object Locations, on page 5-27
- Using Markers, on page 5-29
- Changing Color Schemes, on page 5-31
- Configuring Enhanced Instance Display, on page 5-31

Finding Objects with the Find Command

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.

1. In the Physical Analyst view, right-click and select Find from the popup menu or press Ctrl-f to open the Find Object dialog box. Move the dialog box so you can see both the view and the dialog box.



Unhighlighted: 0 of 0

Highlighted: 0 of 0

Filter Search: **Any**

Unqualified search

Note: You can also bring up the Find Object dialog box by selecting Edit->Find from the menu or by clicking the Find (binoculars) icon in the tool bar

---

2. 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).

Note: The Find command does not include physical instances in its search.

---

3. You can choose to restrict your search for the design in the following ways:
   - search for objects that you select from the list, go to step 4.

  - use Search by Name, which filters the design depending on the name you specify in this field, go to step 5.
  - use Filter Search, which filters the design depending on the type of filter you choose from the pull-down list, go to step 6.

  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.

4. Do the following to select objects from the list. To use wildcards in your selection, see the next step.
   - Click First 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 on the objects you want from the list. If the object name exceeds the width of the Unhighlighted box, click the entry in the list to display the entire name in the field below the Unhighlighted box.
   - Click the right arrow to move the objects into the Highlighted box on the right, or double-click individual names.

   Objects transferred to the Highlighted box are automatically highlighted in the view.

5. Do the following to select objects using patterns or wildcards.
   - Type a pattern in the Search By Name field. When you use wildcards between hierarchies, all pattern matching is displayed from the top level to the lowest level hierarchy, inclusively. See *Using Wildcards with the Find Command* on page 5-26 for a detailed discussion of wildcards.
   - Click First 200 or Find All. The former finds the first 200 matches, and then you can click the button again to find the next 200.

     The Unhighlighted list shows the objects that match the wildcard criteria. If the object name exceeds the width of the Unhighlighted box, click the entry in the list to display the entire name in the field below the Unhighlighted box.
   - Click the right arrow to transfer the selections to the Highlighted box on the right, or double-click individual names. The objects are automatically highlighted in the view.

   You can use wildcards to avoid typing long path names. Start with a general pattern, and then make it more specific.

6. Do the following to select objects by choosing a type of search to filter.
   - Select the type of filter search you want to perform from the pull-down list. See *Using Filter Search With the Find Command* on page 5-26 for a complete list of selection options.
   - Click First 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 selected objects into the Highlighted box on the right, or double-click individual names.

   For large designs, reduce the scope of the search using this technique.

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

Use the following wildcards when you search the design:

*	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.

Using Filter Search With the Find Command

Use the Filter Search option on the Find Object dialog box to limit the scope of the search for your design. You can choose a search subcategory from a drop-down selection list for any of the object types. See Chapter 3, *User Interface Commands* of the *Reference Manual* for more information about filtering design objects in the Physical Analyst view using the Find command.

Unfiltering Nets

To enable the display of nets, you must explicitly unfilter (show) the nets that you want to display using the find command. To filter all nets:

1. Open the Find Object dialog box.
2. Select the Nets tab at the top of the dialog box.

3. Click the Find All button.

Finding Object 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. After selecting this location, a description of the object is displayed in the dialog box window, if applicable. A running history is kept of the selected locations and the markers created.

The Go to Location option is useful when you want to analyze a particular instance from the log file (`.srr`) or timing analyst file (`.ta`). Simply use the location of the instance from any of these files, then this instance can be easily located and displayed in the Physical Analyst view.

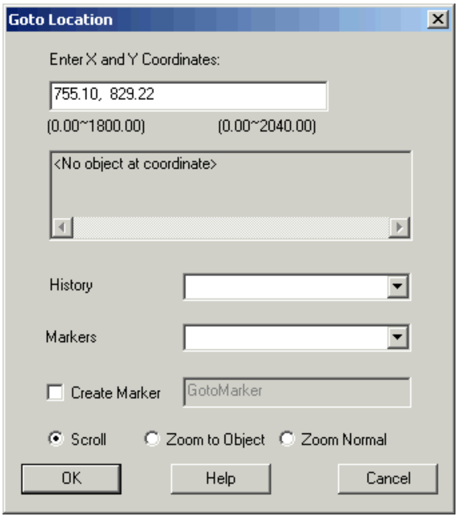
Note: Note, the unit of measurement used in the `.srr` and `.ta` files is microns, but the `.def` file uses database units. Use the UNITS DISTANCE MICRONS factor from the `.def` file to convert database units to microns.

Specifying Location of an Object

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, you can access the Go to Location command by using one of the following:
   – Right-click and select Go to Location from the popup menu
   – Ctrl-g shortcut key
   – Select View->Go to Location from the menu bar

   The command displays the Goto Location dialog box.



Enter X and Y Coordinates: 755.10, 829.22

(0.00~1800.00) (0.00~2040.00)

<No object at coordinate>

□ Create Marker GotoMarker

● Scroll ○ Zoom to Object ○ Zoom Normal

2. Enter a coordinate pair (X and Y) location value in microns. To do this:
   - You can simply 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, the coordinate pair location can be enclosed in parentheses.
   - You can also copy and paste a coordinate pair location from a log file (`.srr`) to the Go to Location dialog box.

     You must first copy (Edit->Copy or Ctrl-c) a coordinate pair location from the log file, then open the Go to Location dialog box in the Physical Analyst view and paste (Edit->Paste or Ctrl-v) this coordinate pair in to the field.

   Note that the unit of measurement used in the `.def` file is database units.

   If a history of location pairs exist, you can highlight a selection to reuse these values. Selecting a coordinate pair automatically updates the X and Y coordinate fields.
3. You can create a marker at the coordinate pair location. To do this:
   - Check the Create Marker box in the dialog box.

- Use the default marker name (GotoMarker*n*) or specify a name.

A marker symbol (⊞) appears in the Physical Analyst view at the location specified. A tool tip can be displayed over the marker that shows the marker name and its X and Y coordinates.

If a history of markers exists, you can highlight a selection to reuse the marker. Selecting a marker automatically updates the X and Y coordinate fields.

4. Select a zoom mode and then click OK. You can choose one of the following zoom modes:
   - Scroll – centers the specified location without zooming
   - Zoom to Object – similar to selecting an object and using Zoom Selected
   - Zoom Normal – similar to using the Zoom 100% (🔍) icon

Using Markers

Markers are bookmarks for physical coordinates in the Physical Analyst view. Markers can be:

- Added
- Moved
- Deleted

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.

Markers are useful when you want to analyze floorplan placement in the Physical Analyst view. You can find an object, such as an instance, then create a marker on this instance.

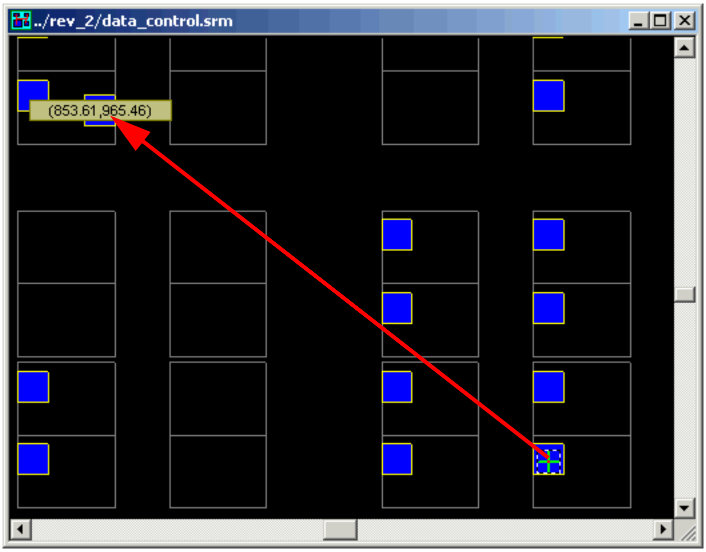
Adding Markers

To create a marker, right-click and select Markers->Add marker from the popup menu or use the Ctrl-m shortcut key. When a marker is created at an instance or net location, the marker takes on the object's name. Otherwise, markers are identified as Marker1, Marker2, and so on. A tool tip can be displayed over a marker to show its name and X and Y coordinates.

Markers can also be added using the Go to Location command. See *Finding Objects with the Find Command* on page 5-23.

Moving Markers

When you select a marker symbol (⊞), its dotted drag outline appears. To move a marker, press and hold the left-mouse button while dragging the marker to its new location and then release the mouse button.



Deleting Markers

To delete a marker, highlight the marker then right-click and select Markers->Remove Selected from the popup menu or use the Del key. To delete all markers, right-click and select Markers->Remove All from the popup menu.

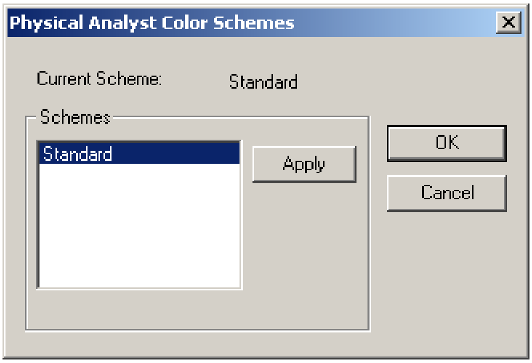
Measuring Distances

You can measure the distance from a selected marker to a cursor location. The manhattan (X+Y) distance, calculated in microns, from the marker to the cursor and the X and Y coordinates of the cursor are displayed in the status bar at the bottom of the Physical Analyst view. If you select two markers, the distance between the two markers is calculated and displayed in the status bar. If you select more than two markers, the distance measurement is ignored.

To advance to a next or previous marker, right-click and select either Markers->Go to Next or Markers->Go to Previous from the popup menu or use the F2 and Shift+F2 keys, respectively. Markers are selected in the order they were created, either forward or reverse. If the view is zoomed, the selected marker is centered in the view.

Changing Color Schemes

Currently, only one standard color scheme can be displayed in the Physical Analyst view. To display this dialog box, select Options->Physical Analyst Color Schemes from the menu.



Physical Analyst Color Schemes

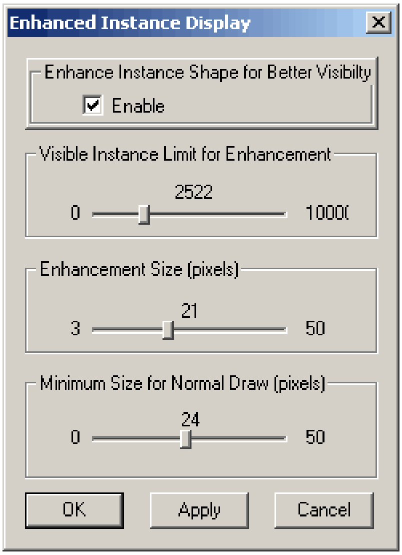
Current Scheme: Standard

Schemes

Standard

Configuring Enhanced Instance Display

The enhanced display mode, when enabled, causes core cells to be drawn as diamonds of fixed size regardless of zoom level. Selecting View->Configure Enhanced Instance Display in an active Physical Analyst view brings up the Enhanced Instance Display dialog box which is used to set the enhanced display parameters.



Enhanced Instance Display

Enhance Instance Shape for Better Visibilty

☑ Enable

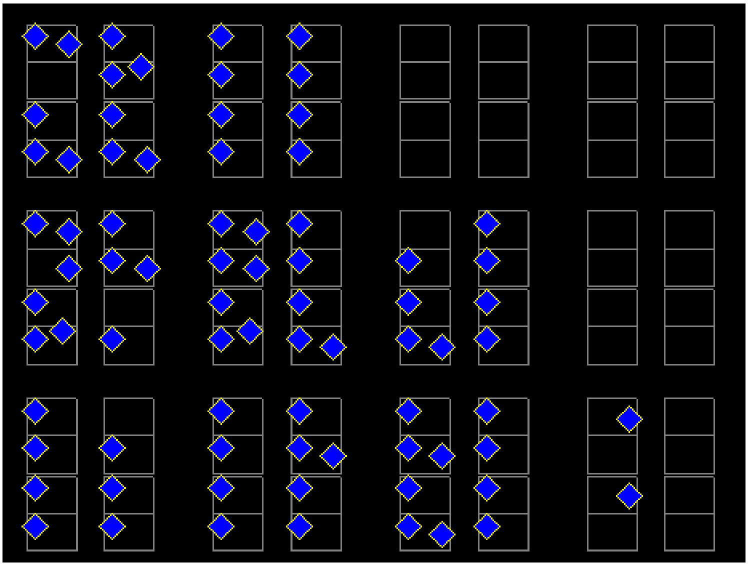
Visible Instance Limit for Enhancement: 2522 (0 – 1000()

Enhancement Size (pixels): 21 (3 – 50)

Minimum Size for Normal Draw (pixels): 24 (0 – 50)

Option	**Description**
Enhance Instance Shape for Better Visibility	Enables enhanced instance display when checked (same as selecting Enhance in the Inst. Display section of the Objects pane).
Visible Instance Limit for Enhancement	Sets the maximum number of visible core cells that can be displayed in enhanced instance display mode.
Enhancement Size (pixels)	Sets the size (in pixels) of the instance; instances are drawn as diamonds to differentiate them from normal cell shapes.
Minimum Size for Normal Draw (pixels)	Sets the minimum size of an average core cell when the cell is drawn in normal mode and not enhanced

The following figure shows how enhanced instances are displayed in the Physical Analyst view.



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 cross probes depending on the options enabled through the View->Cross Probing menu bar. The following options are enabled by default:

- Send Crossprobes when selecting
- Cross Probing from RTL Analyst
- Cross Probing from Tech Analyst
- Cross Probing to HDL Source
- Auto route cross probe insts

The Physical Analyst responds to incoming cross probes as well as sending out cross probes in response to selections. For efficiency reasons, you may want to send cross probes from the Physical Analyst only on demand by disabling the Send Crossprobes when selecting option. To automatically send cross probes from the Physical Analyst tool, the Send Crossprobes when selecting option must be enabled. To automatically route cross-probed instances, enable the Auto route cross probe insts option.

Crossprobing works in conjunction with filtering. If an object is filtered (hidden) and a cross probe message is received, the object is unfiltered (in a new filter state). For example, you can filter all objects, then select objects in the HDL Analyst view. As objects are selected, they become visible and highlighted in the Physical Analyst view. You can select objects in an HDL Analyst view using the graphic view, hierarchy browser, or the Object Query dialog box.

The following table summarizes the cross probing capabilities to and from the Physical Analyst view.

From	To	Procedure
Physical Analyst	Source code	Double-click an instance. If the source code file is not open, a Text Editor window is opened 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.
Text File	Physical Analyst	The Physical Analyst view must be open. Highlight the appropriate portion of text (for example, hierarchical instance name or instance name) in the text editor. In some cases, you may have to select the entire block of text to cross probe. From the log file, right click and select Select in Analyst or, to show only the object selected, click the Filter on Selected Gates icon in the menu bar after highlighting the text (use the Reset filter icon to redisplay the unfiltered objects).
RTL Analyst	Physical Analyst	The Physical Analyst view must be open. Click the object (instance or macro) to highlight and crossprobe. *Usage Note:* • To cross probe from the RTL Analyst view to the Physical Analyst view, you must enable the Cross Probing from RTL Analyst option. • 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.

From	To	Procedure
Physical Analyst	RTL Analyst	The RTL Analyst view must be open. Click the object to highlight and cross probe. *Usage Note:* • To automatically cross probe from the Physical Analyst view to the RTL Analyst view, you must enable the Send crossprobes when selecting option. • To cross probe only on demand, you must disable the Send cross probes when selecting option. Click the object to highlight, then right-click and select Crossprobe Selected from the popup menu.
Technology Analyst	Physical Analyst	The Physical Analyst view must be open. Click the object to highlight and cross probe. *Usage Note:* To cross probe from the Technology Analyst view to the Physical Analyst view, you must enable the Cross Probing from Tech Analyst option.
Physical Analyst	Technology Analyst	The Technology Analyst view must be open. Click the object to highlight and cross probe. *Usage Note:* • To automatically cross probe from the Physical Analyst view to the Technology Analyst view, you must enable the Send crossprobes when selecting option. • To cross probe only on demand, you must disable the Send cross probes when selecting option. Click the object to highlight, then right-click and select Crossprobe Selected from the popup menu.

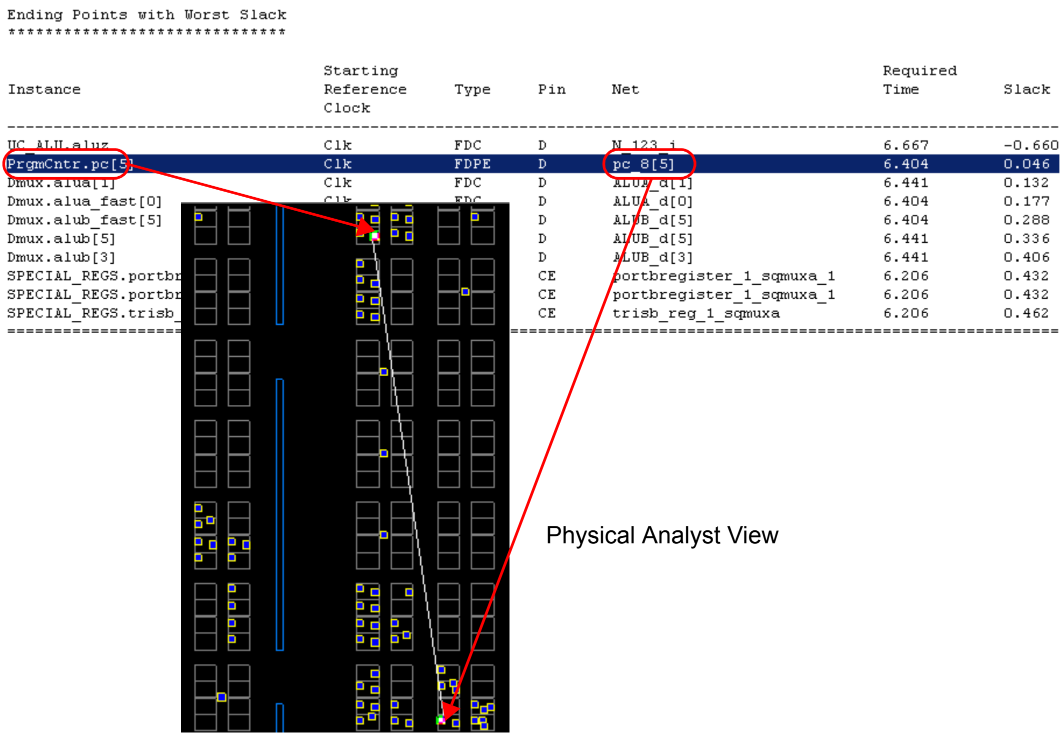
Crossprobing from a Text File

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. Select the instances to highlight them from a text file, such as the log file.
2. Then right-click and select Select in Analyst from the popup menu in this file.

3. Check the Physical Analyst view. Selected instances are highlighted in this view.

Log File

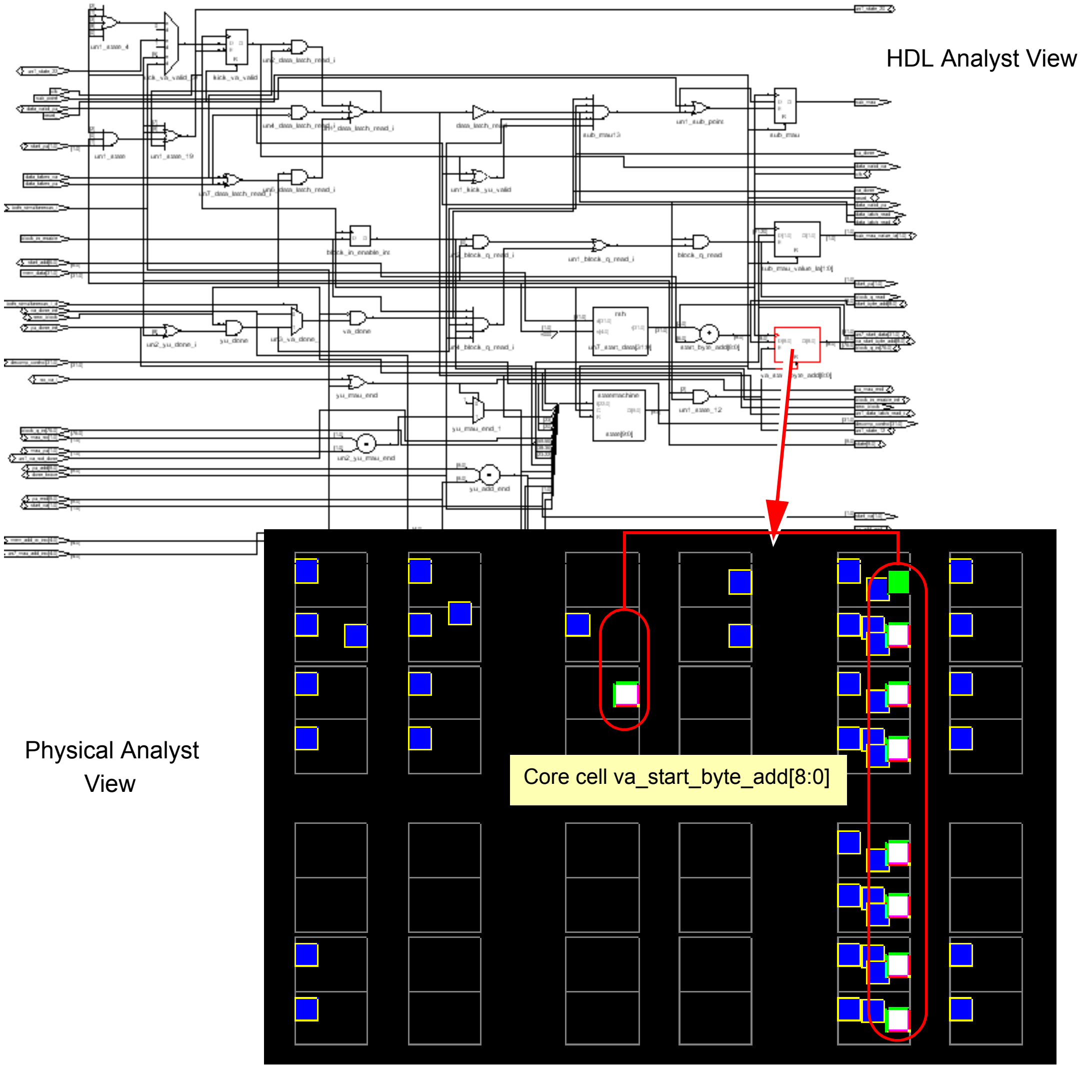


```
Ending Points with Worst Slack
******************************

                                   Starting                                                      Required
Instance                           Reference      Type     Pin     Net                           Time         Slack
                                   Clock
------------------------------------------------------------------------------------------------------------------
UC_ALU.aluz                        Clk            FDC      D       N_123_i                       6.667        -0.660
PrgmCntr.pc[5]                     Clk            FDPE     D       pc_8[5]                       6.404        0.046
Dmux.alua[1]                       Clk            FDC      D       ALUA_d[1]                     6.441        0.132
Dmux.alua_fast[0]                  Clk            FDC      D       ALUA_d[0]                     6.404        0.177
Dmux.alub_fast[5]                                          D       ALUB_d[5]                     6.404        0.288
Dmux.alub[5]                                               D       ALUB_d[5]                     6.441        0.336
Dmux.alub[3]                                               D       ALUB_d[3]                     6.441        0.406
SPECIAL_REGS.portbr                                        CE      portbregister_1_sqmuxa_1      6.206        0.432
SPECIAL_REGS.portbr                                        CE      portbregister_1_sqmuxa_1      6.206        0.432
SPECIAL_REGS.trisb_                                        CE      trisb_reg_1_sqmuxa            6.206        0.462
==================                                         =======================================================
```

Physical Analyst View

Crossprobing from the RTL View

When the mapped netlist (SRM) contains information relating mapped instances to RTL instances (RTL name field), instances will be highlighted in the Physical Analyst in response to the selection of modules or primitives in the RTL view. In the case of a module, all mapped objects with physical information implementing the module are highlighted. See the following example.



Crossprobing from the Technology View

Each instance in the Physical Analyst has its counterpart in the Technology view. When selecting an instance in the Technology view that is a placed primitive in the Physical Analyst, the instance in the Physical Analyst is also highlighted. This is similar to crossprobing from the RTL view.

Crossprobing from the Physical Analyst View on Demand

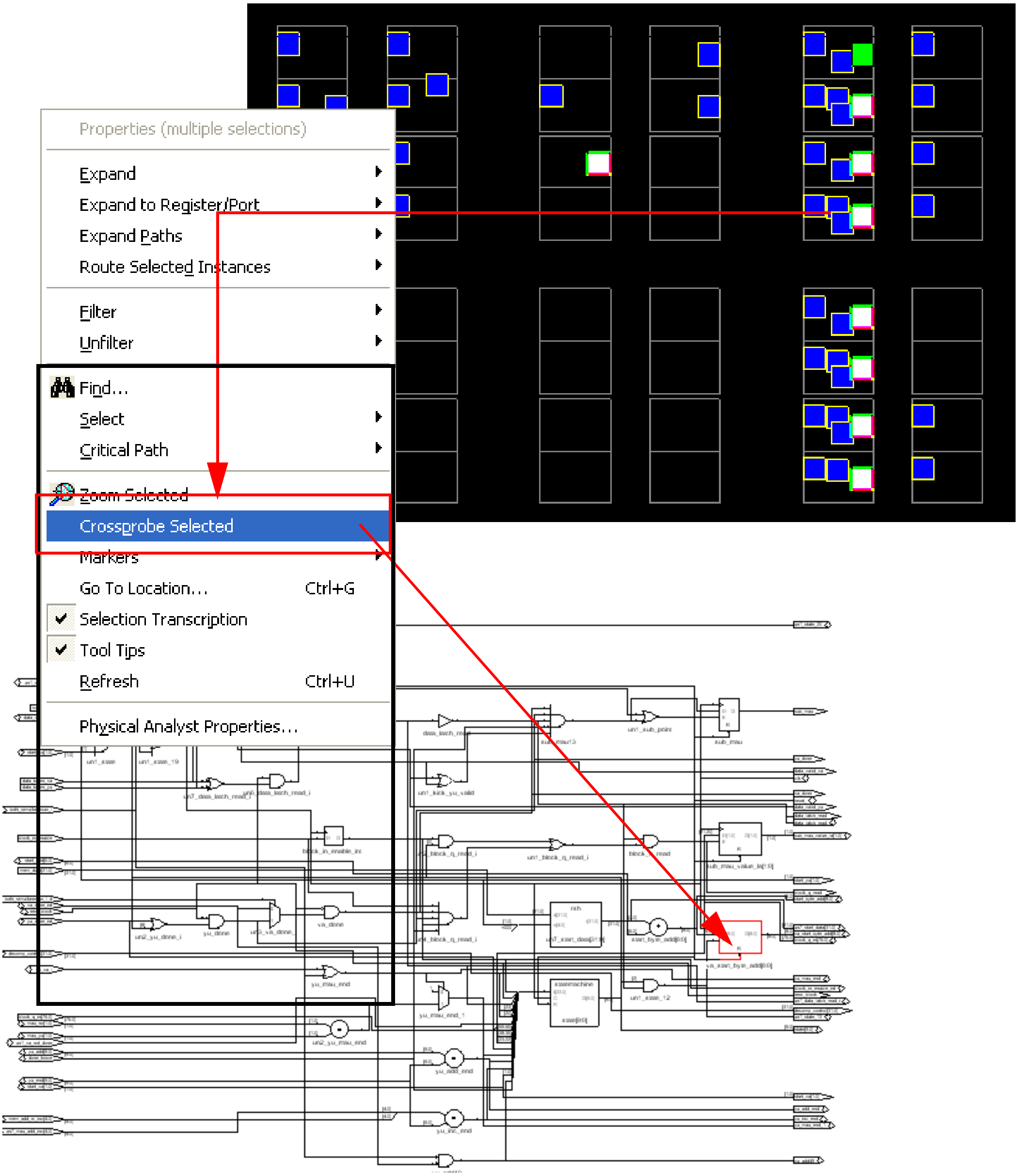
For very large designs, you might want to cross probe from the Physical Analyst view to the RTL or Technology view only at your request. To cross probe on demand:

1. The View->Cross Probing->Send Crossprobes when selecting option must be disabled. If the Send Crossprobes when selecting option is enabled (the default), cross probing from the Physical Analyst view occurs automatically.

   The RTL or Technology view must already be open.

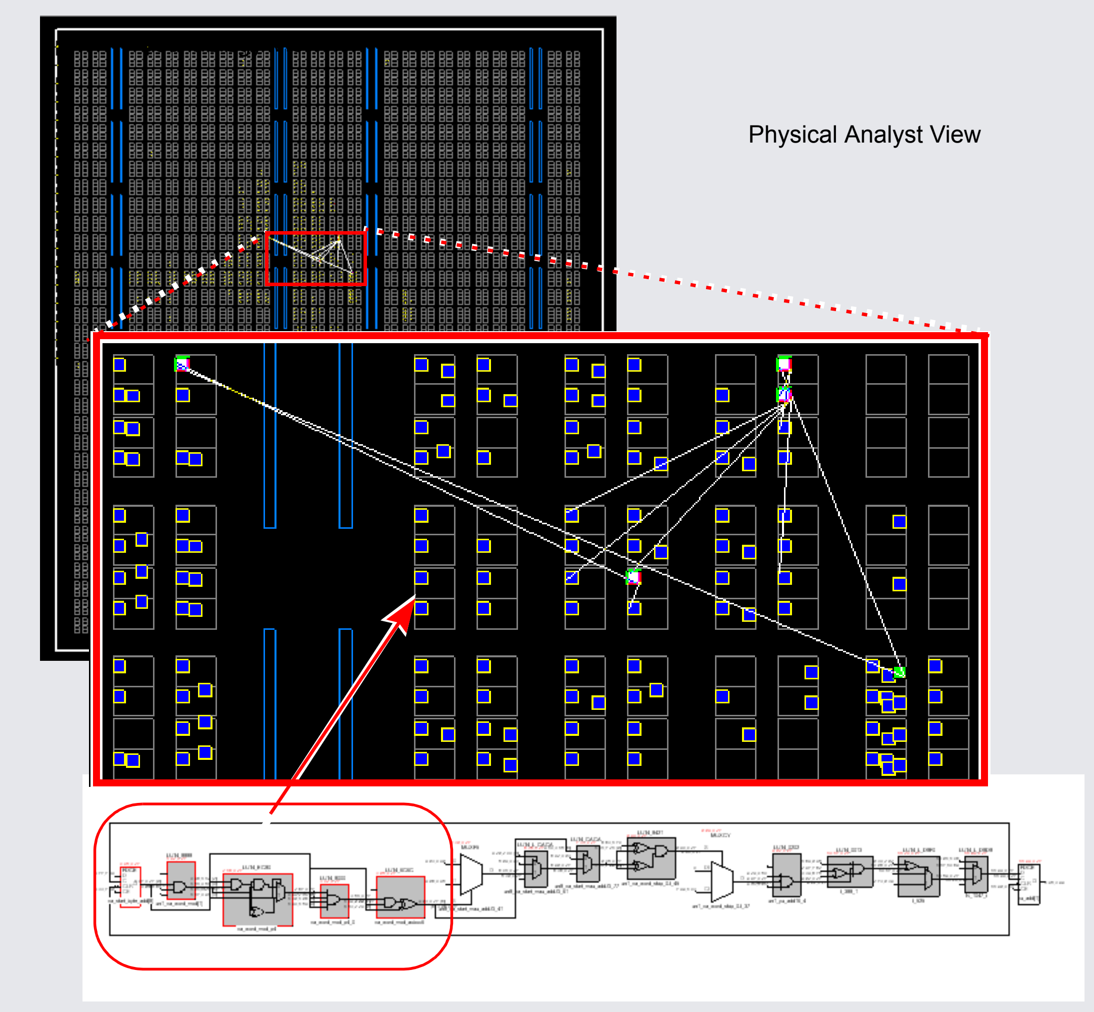
2. Click on the object(s) to highlight from the Physical Analyst view.

3. Right-click and select Crossprobe Selected from the popup menu. The corresponding object(s) will be highlighted in the RTL or Technology view.


Properties (multiple selections)
Expand
Expand to Register/Port
Expand Paths
Route Selected Instances
Filter
Unfilter
Find...
Select
Critical Path
Zoom Selected
Crossprobe Selected
Markers
Go To Location... Ctrl+G
Selection Transcription
Tool Tips
Refresh Ctrl+U
Physical Analyst Properties...

Auto Route Crossprobing

If you want 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 example shows cross probing from the Technology view to the Physical Analyst view with the Auto route cross probe insts option enabled.



Analyzing Netlist with the Physical Analyst

There are a number of commands to use for analyzing the netlist that generally involve tracing logic. These commands are available from the right-click popup menus in the Physical Analyst view.

Different commands are available, depending on what is currently selected and whether you right-click an object or the background. If one or more objects are selected and you right-click in the view background, the menu includes global commands as well as selection-specific commands for the selected objects. See Chapter 3, *User Interface Commands* of the *Reference Manual* for a complete list of View menu and Physical Analyst popup menu commands.

Filtering the View

Filtering is a useful first step in analysis, because it focuses analysis 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 on Selected Gates icon ( ).

   – Press Alt 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. You can now analyze the objects and perform operations such as:

   – Trace paths, build up logic

   – Filter further

   – Find objects

   – Hide objects

   – Cross probe from a filtered state

3. To return to the previous schematic view, click the Back (←) icon.

Expanding Pin and Net Logic

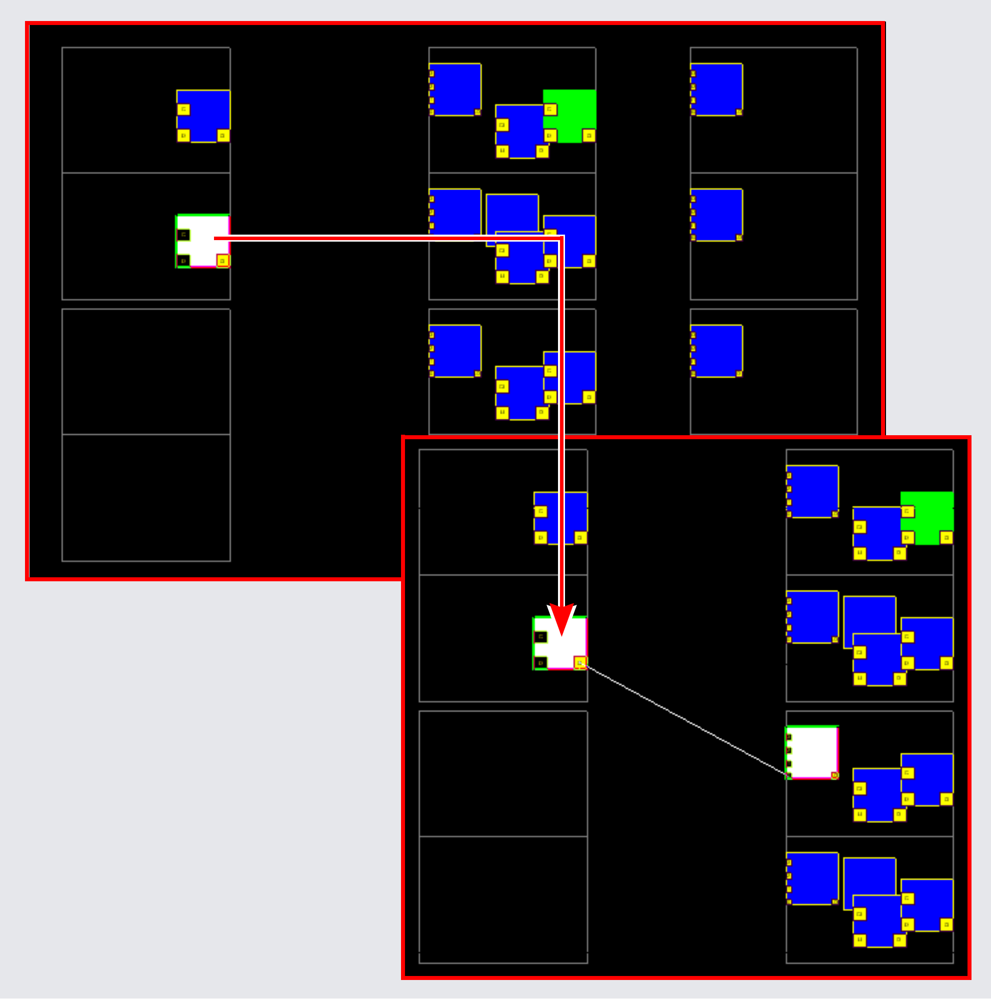
When you are working in a filtered view, you might need to include more logic in your selected set to analyze your design. This section describes commands that expand logic fanning out from pins or nets; to expand paths, see *Expanding and Viewing Connections* on page 5-48.

Use the Expand commands with the Filter and Nets->Visible commands to isolate and connect the logic that you want to examine.

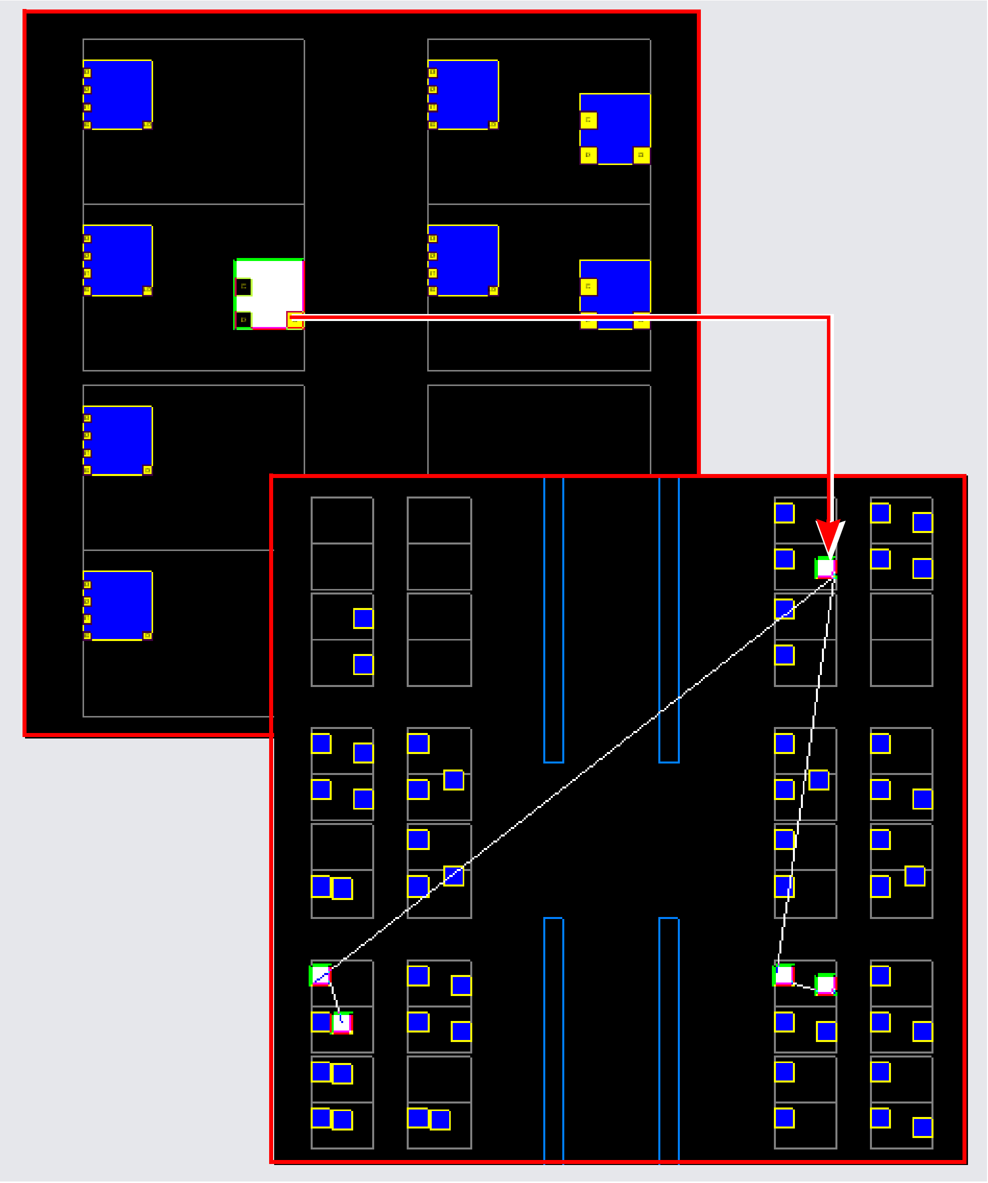
1. To expand logic from a pin, use the following commands.

To...	Do this (Physical Analyst Popup menu)...
see all cells connected to a pin	Select a pin from the cell instance and select Expand->Selected Pins. See *Expanding Logic Example* on page 5-44. *Usage Note*: You can also select to expand from output pins, input pins, or all pins.
see all cells that are connected to a pin, up to the next register/port	Select a pin from the cell instance and select Expand to Register/Port->Selected Pins. See *Expanding Logic to Register/Port Example* on page 5-45. *Usage Note*: You can also select to expand to register/port from output pins, input pins, or all pins.

Expanding Logic Example



Expanding Logic to Register/Port Example



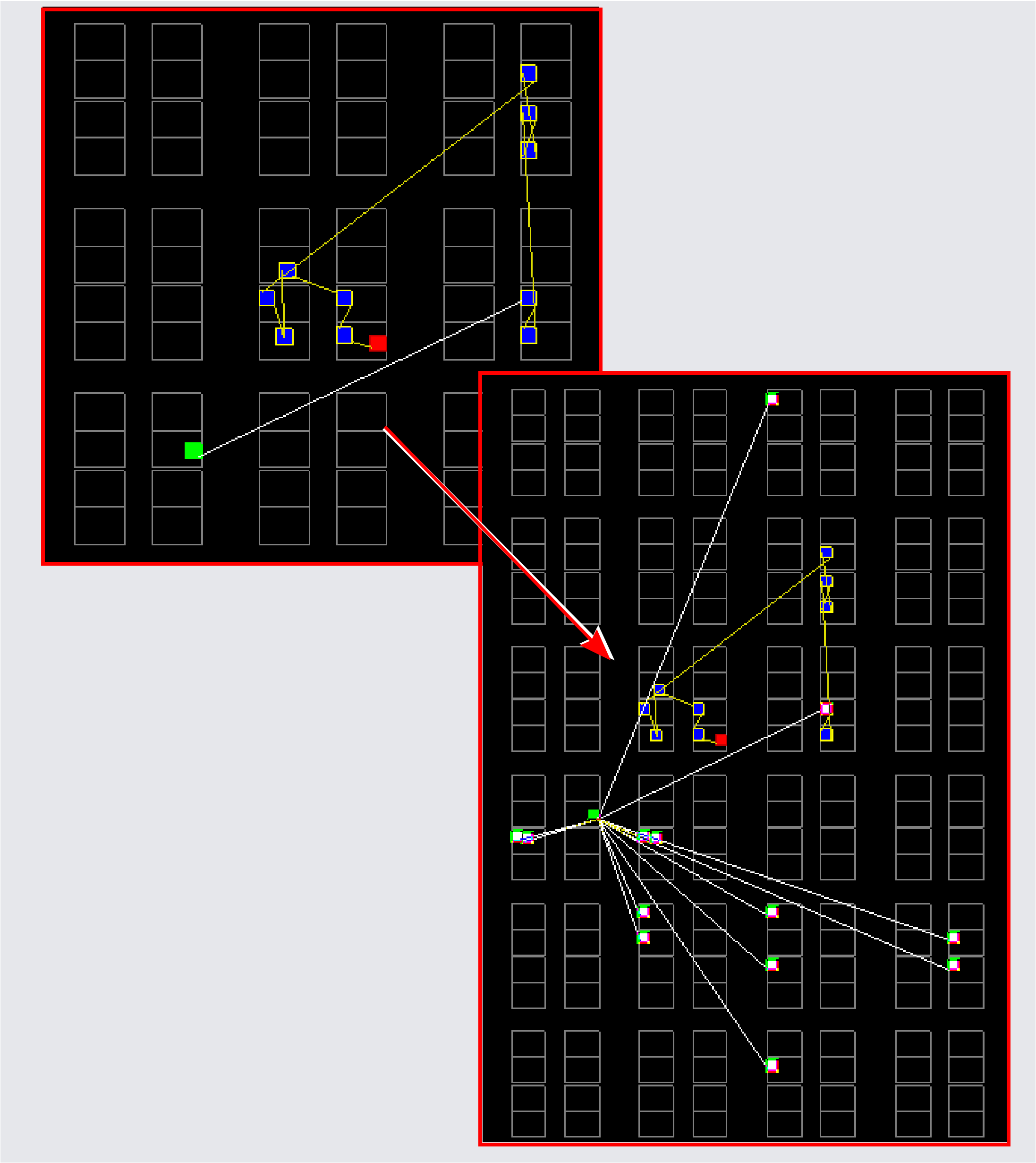
2. To expand logic from a net, use the commands shown in the following table.

To...	**Do this...**
select all instances on a net	Select a net and select Select Net Instances->All Pins. The software shows an unfiltered view that includes all the instances connected to the net along the signal path. *Usage Note:* You can also select to show output pins or input pins.
highlight all visible instances on a net	Select a net and select Highlight Visible Net Instances->All Pins. You see a filtered view of all instances connected to the selected net along the signal path. *Usage Note:* You can also select to show output pins or input pins.
select net driver	Select a net and select Select Net Driver. Shows an unfiltered view that includes the driver of the net.
go to net driver	Select a net and select Go to Net Driver. Shows and scrolls to the driver of the net.

Selecting all Instances of a Net Example

The following example shows the critical path for a design in the Physical Analyst view.

1. The critical path is filtered.
2. A net on the critical path is selected.
3. Then right-click and select Select Net Instances->All Pins, for example.



Expanding and Viewing Connections

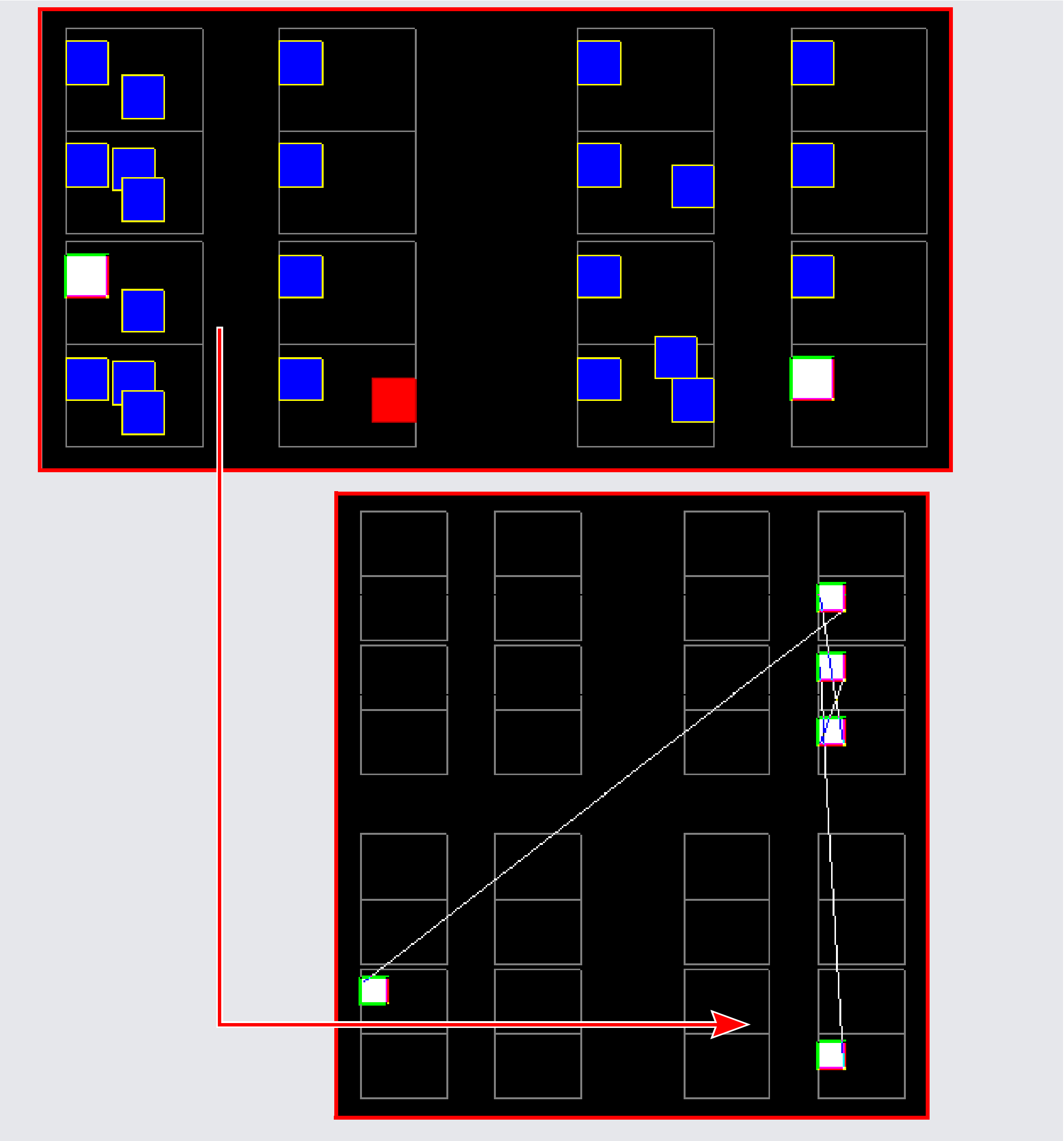
This section describes commands that expand logic between two or more objects; to expand logic out from a net or pin, see *Expanding Pin and Net Logic* on page 5-43. You can also isolate the critical path or use the Timing Analyst to generate a schematic for a path between objects, as described in *Analyzing Timing with the Physical Analyst* on page 5-51.

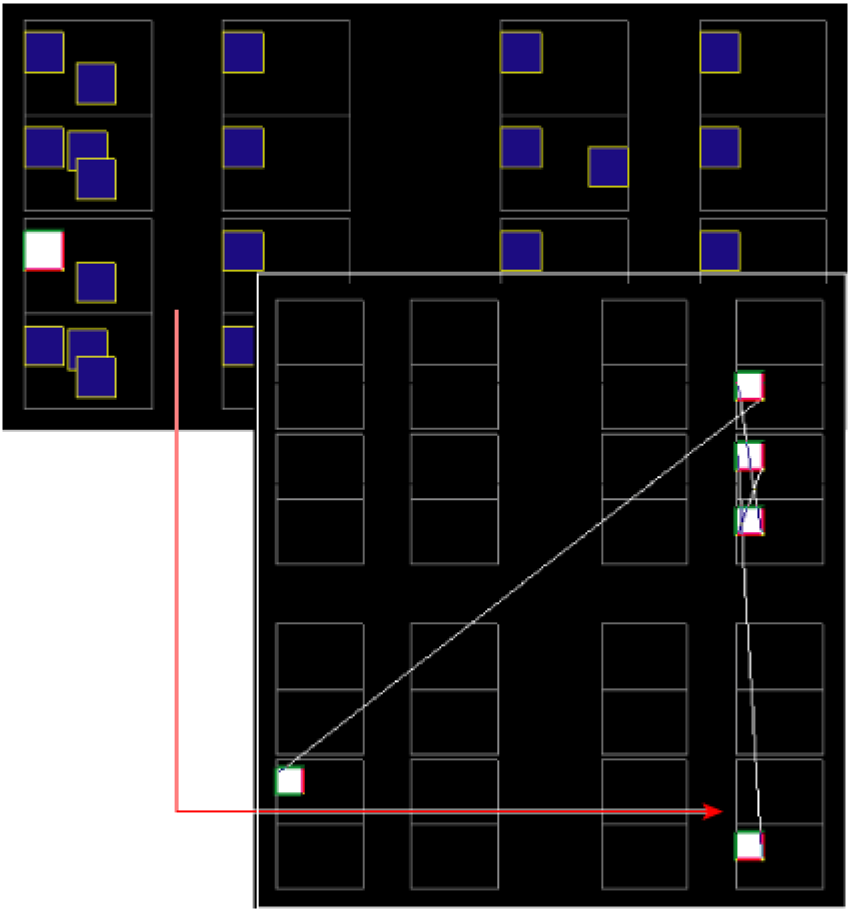
Use the following path commands with the Filter and Nets->Visible commands to isolate and connect the logic that you want to examine.

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.

Expanding Paths Example





Analyzing Timing with the Physical Analyst

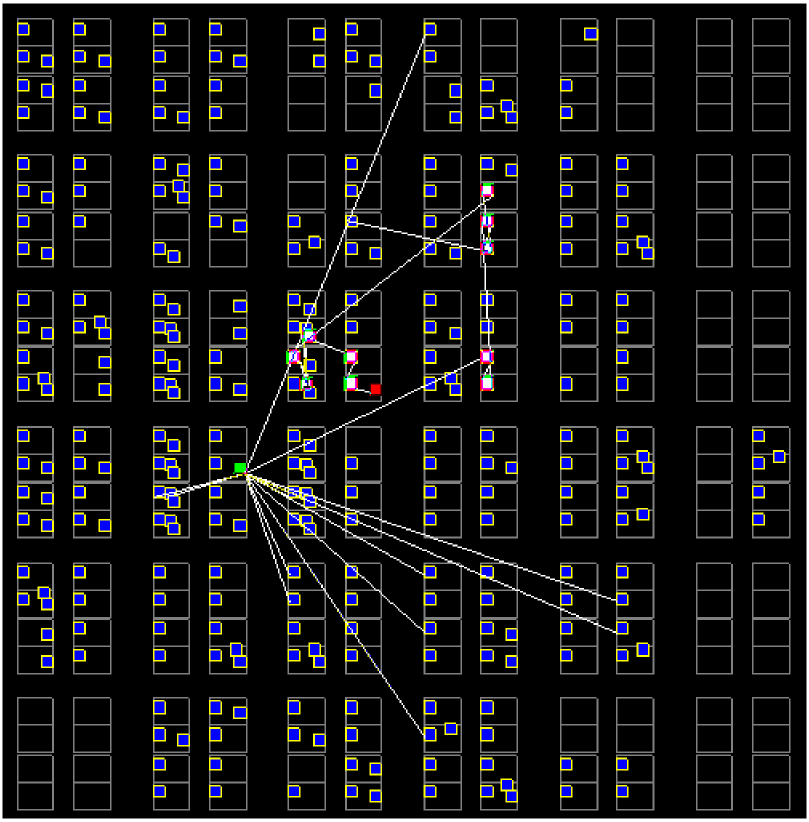
You can use the Physical Analyst functionality to analyze timing. This section describes how to view and use the critical path for further physical synthesis.

- Viewing Critical Paths, on page 5-51
- Tracing Critical Paths Forward and Backwards, on page 5-54

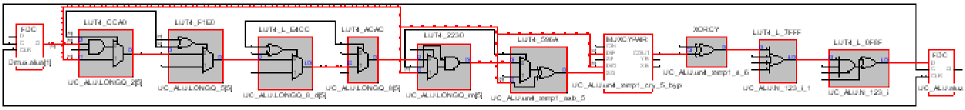
Viewing Critical Paths

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.



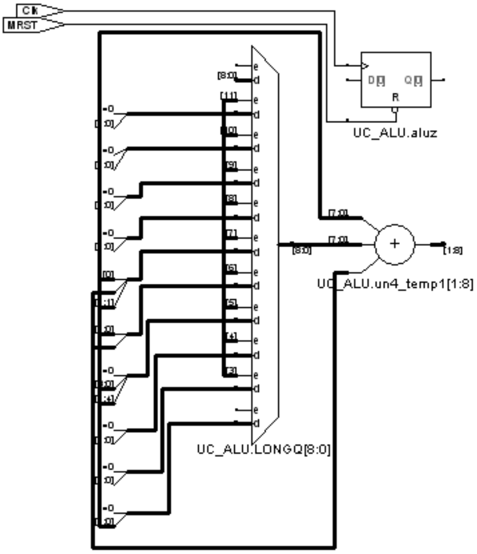
2. Check the Technology view. Click the Filter on Selected Gates icon (![]) to display the critical path.



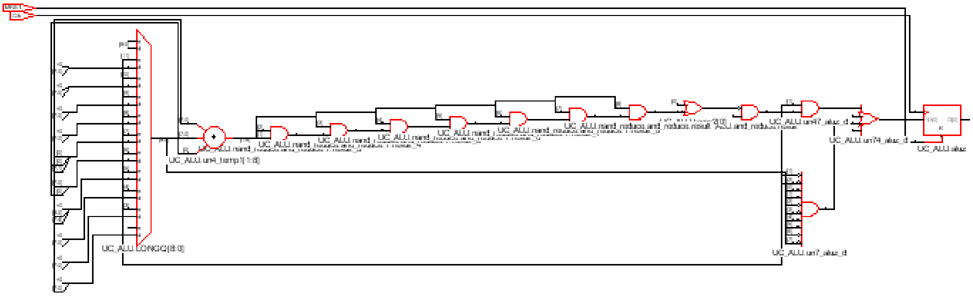
3. You can also cross probe the critical path from the flattened Technology view to the Physical Analyst view by clicking on the Show Critical Path icon (⏱). Then, right-click and select Select All Schematic->Instances. Make sure the Physical Analyst view is open.

4. Check the Physical Analyst view. Critical path instances and nets should be highlighted in this view. See the figure in step 1.

5. In the HDL Analyst view that is already open, click on the Filter on Selected Gates icon ( ). Only the instances and nets belonging to the critical timing path are displayed, as shown below.



6. 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 and Backwards

The Physical Analyst tool also provides the capability to trace a critical path from its starting point to its ending point. You can trace the critical path forward or backwards, either starting from the instance containing the critical start point or starting from the instance containing the critical end point, respectively.

Trace Critical Paths Forward

The following procedure shows you how to trace a critical path forwards.

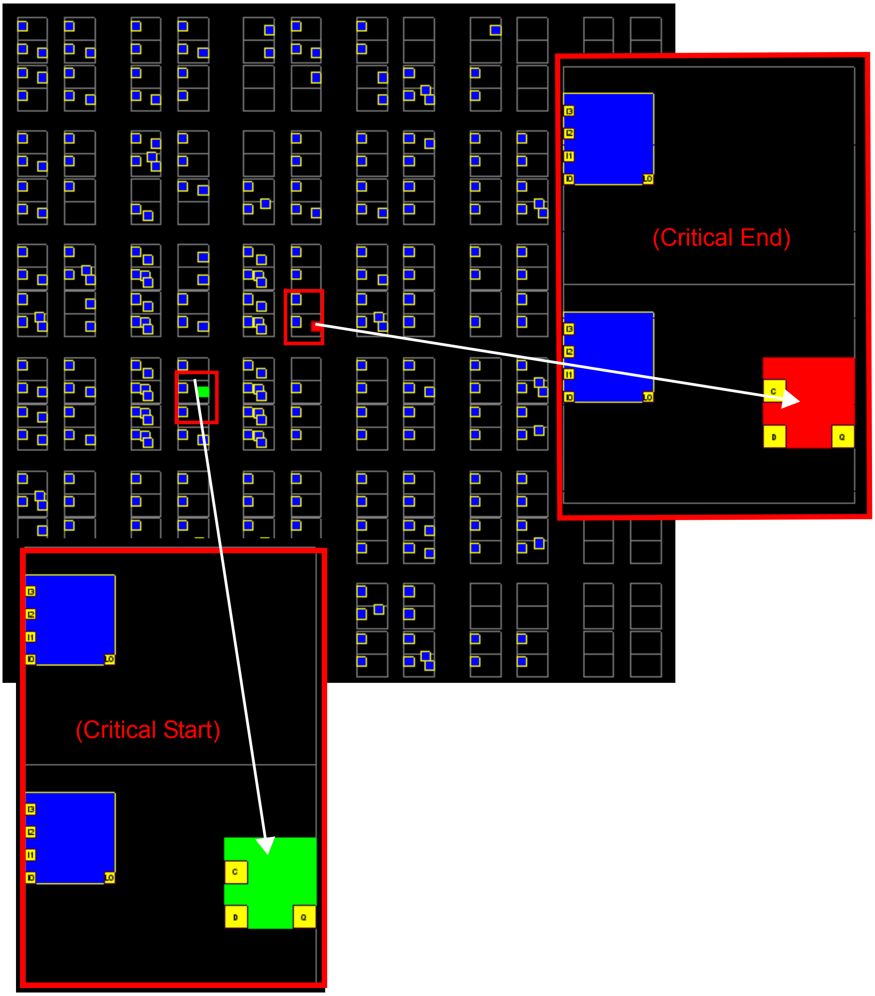
1. To trace a critical path forwards, either:
   - Right-click and select Critical Path->Expand Path Forward from the popup menu
   - Use the F3 shortcut key

   The instance containing the critical path start 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 start point.

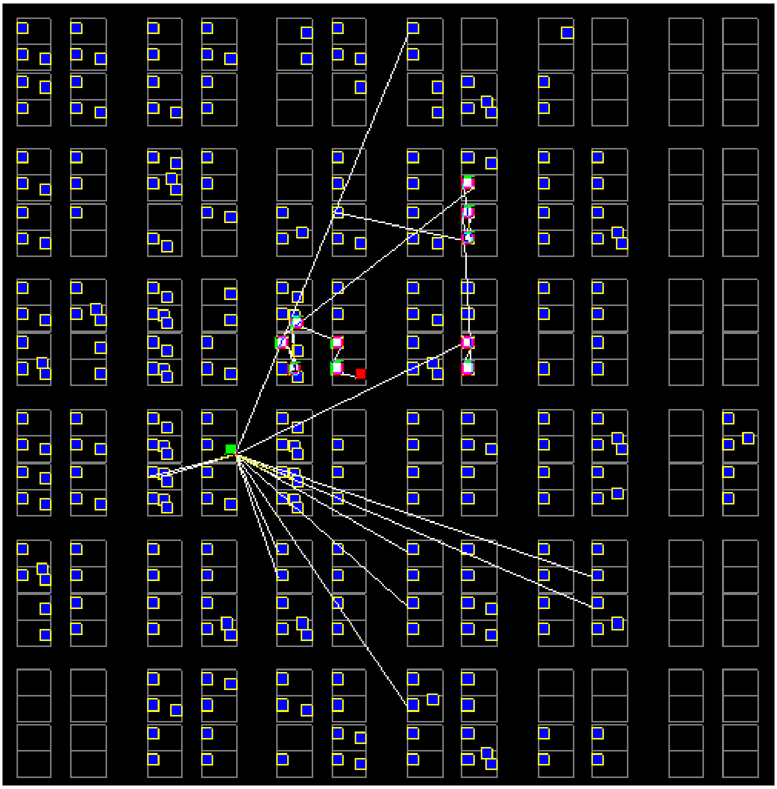
Note: You can also use the Filter Search option of the Find command to locate the Critical path start point. The cell location of the critical path start point is displayed with the color green in the Physical Analyst view.

2. Use one of the critical path forward commands described in step 1 to continue to trace the net to the next instance in its path.

   The next instance containing 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. Continue using the critical path forward command until you reach the end point. The following figure shows you how the critical path is finally displayed.



Trace Critical Paths Backwards

The following procedure shows you how to trace a critical path backwards. See the figures in *Trace Critical Paths Forward* on page 5-54, which also apply to this procedure.

1. To trace a critical path backwards, either:

   – Right-click and select Critical Path->Expand Path Backward from the popup menu

   – Use the Shift+F3 shortcut key

   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.

Note: 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 with the color red in the Physical Analyst view.

2. Use one of the critical path backward commands described in step 1 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. Continue using the Critical Path->Expand Path Backward command until you reach the start point. See the figure in step 3 of *Trace Critical Paths Forward* on page 5-54 to show you how the critical path is finally displayed.

CHAPTER 6

Design Optimization

This chapter covers techniques for optimizing your design using built-in tools or attributes. For vendor-specific optimizations, see Chapter 8, *Vendor-Specific Optimizations*.

It describes the following:

Design Guidelines

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, next
- Area Optimization Tips, on page 6-3
- Timing Optimization Settings, on page 6-4

General Optimization Tips

This section contains general optimization tips that are not directly area or timing-related. For area optimization tips, see *Area Optimization Tips, on page 6-3*. For timing optimization, see *Timing Optimization Settings, on page 6-4*.

- 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.

Area Optimization Tips

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 *Timing Optimization Settings,* on page 6-4. General optimization tips are in *General Optimization Tips,* on page 6-2.

- 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 6-7 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 6-5 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 6-15.
- 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 6-15.
- 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.

Timing Optimization Settings

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 *Area Optimization Tips*, on page 6-3. General optimization tips are in *General Optimization Tips*, on page 6-2.

- Use realistic design constraints, about 10 - 15% of the real goal. Over constraining your design can be counter-productive because you can get poor implementations. Use clock, false path, and multicycle 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 6-7 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 saw warnings about feedback muxes being created for signals when you compiled 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.

Optimizing Results

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:

- Sharing Resources, next
- Setting Fanout Limits, on page 6-7
- Controlling Buffering and Replication, on page 6-8
- Controlling Hierarchy Flattening, on page 6-10
- Preserving Objects from Optimization, on page 6-10
- Preserving Hierarchy, on page 6-12

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 8-181 of the *Reference Manual* for syntax examples.

Verilog

```
module top(out, in, clk_in) /* synthesis syn_sharing = “on” */;
```

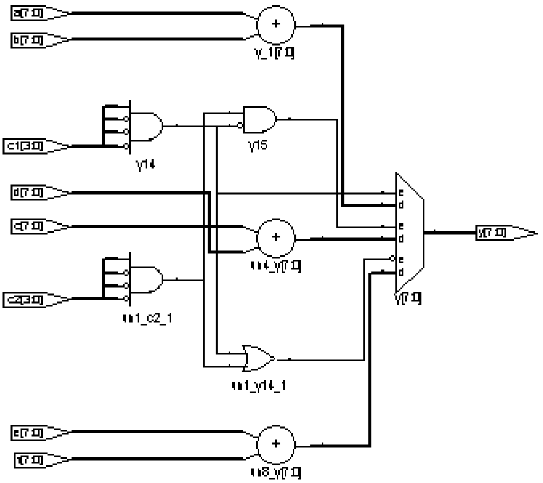
VHDL

```
architecture rtl of top is
  attribute syn_sharing : string;
  attribute syn_sharing of rtl : architecture is “off”;
```

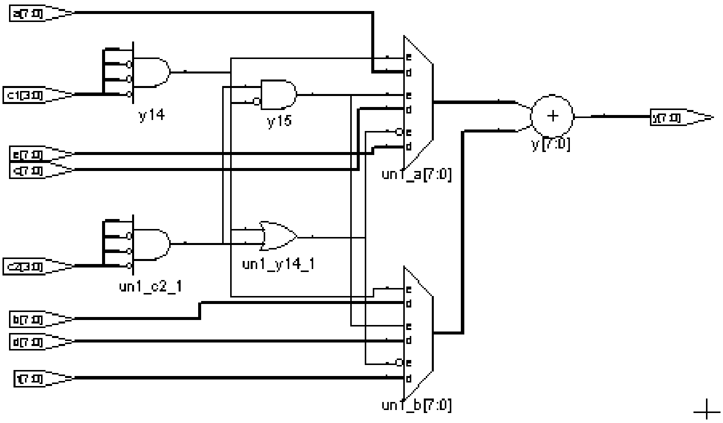
You cannot specify syn_sharing from the SCOPE interface, because it is a compiler directive.

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.



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 6-8 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 6-8.

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 6-8p.

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 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 6-7) and reduce fanout, the software either replicates components or adds buffers. The software reduces fanout on input ports through buffering and reduces fanout on nets driven by registers or combinatorial logic through replication. 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...
syn_maxfan is set on a register output	syn_maxfan is set on input ports in Altera Apex, Actel ProASIC (500K), ProASIC PLUS (PA) and ProASIC3/3E, and QuickLogic pASIC3 designs
syn_replicate is 1	syn_replicate is 0. Note that the syn_replicate attribute must be used in conjunction with the syn_maxfan attribute for Actel families. The syn_replicate attribute is used only to turn off the replication.
	syn_maxfan is set on a port/net that is driven by a port or I/O pad
	The net driver has a syn_keep or syn_preserve 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.

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 6-12.

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	flatten
Remove the current level of hierarchy without affecting the lower levels	remove
Remove the current level of hierarchy and the lower levels	flatten, remove
Flatten the current level (if needed for optimization)	soft

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 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	syn_keep on wire or reg (Verilog), or signal (VHDL). For Actel designs (except 500K and PA), use alspreserve as well as syn_keep.	Keeps net for simulation, a different synthesis implementation, or for passing to the place-and-route tool.
Nets for probing	syn_probe 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	syn_keep on input wire or signal of shared registers	Preserves duplicate driver cells, prevents sharing
Sequential components	syn_preserve on reg or module (Verilog), signal or architecture (VHDL)	Preserves logic of constant-driven registers, keeps registers for simulation, prevents sharing
FSMs	syn_preserve 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	syn_noprune on module or component (Verilog), architecture or instance (VHDL)	Keeps instance for analysis, preserves instances with unused outputs

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 6-10.

1. Attach the syn_hier attribute to the module or architecture you want to preserve. You can also add the attribute in SCOPE. If you use SCOPE to enter the attribute, make sure to use the v: syntax.

2. Set the attribute value:

To...	**Value...**
Preserve the interface but allow cell packing across the boundary	firm
Preserve the interface with no exceptions (Actel, Altera, and Xilinx only)	hard
Preserve the interface and contents with no exceptions (Actel (except PA, 500K, and ProASIC3/3E), 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.

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, next
- Defining State Machines in VHDL, on page 6-14
- Specifying FSMs with Attributes and Directives, on page 6-15

For information about the attributes used to define state machines, see *Running the FSM Compiler on Individual FSMs*, on page 6-20.

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 6-14 and *Specifying FSMs with Attributes and Directives*, on page 6-15.

- 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;
  ```

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 6-13 and *Specifying FSMs with Attributes and Directives*, on page 6-15.

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 *Using the Symbolic FSM Compiler*, on page 6-17), 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 6-13 and *Defining State Machines in VHDL*, on page 6-14.

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	syn_state_machine=1
Prevent state machines from being extracted and optimized	syn_state_machine=0
Prevent the state machine from being optimized away	syn_preserve=1

   For information about how to add attributes, see *Adding Attributes and Directives*, on page 3-66.

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...	**syn_encoding Value**	**Explanation**
Area is important	sequential	One of the smallest encoding styles.
Speed is important	onehot	Usually the fastest style and suited to most FPGA styles.
Recovery from an invalid state is important	safe, with another style. For example: `/* synthesis syn_encoding = “safe, onehot” */`	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 safe to reset the state machine.
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 sequential, more than one bit can change at a time; for gray, only one bit changes at a time, but more than one bit can be hot.
There are a large number of flip-flops	onehot	Fastest style, because each state variable has one bit set, and only one bit of the state register changes at a time.

3. If you are using VHDL, you have two choices for defining encoding:
   - Use syn_encoding as described above, and enable the FSM compiler.
   - Use syn_enum_encoding to define the states (sequential, onehot, gray, and safe) and disable the FSM compiler. If you do not disable the FSM compiler, the syn_enum_encoding values are not implemented. This is because the FSM compiler, a mapper operation, overrides syn_enum_encoding, which is a compiler directive.

     Use this method for user-defined FSM encoding. For example:

     ```
     attribute syn_enum_encoding of state_type : type is "001 010 101";
     ```

Using the Symbolic FSM Compiler

The Symbolic FSM Compiler is an advanced state machine optimizer, which 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 uses the state machines extracted by the FSM Compiler when it explores different encoding styles. The FSM Explorer option is only available in the Synplify Pro and Synplify Premier tools.

For more information, see the following:

- Choosing When to Use the FSM Compiler, on page 6-17, next
- Running the FSM Compiler on the Whole Design, on page 6-18
- Running the FSM Compiler on Individual FSMs, on page 6-20
- Specifying FSMs with Attributes and Directives, on page 6-15

Choosing When to Use the FSM Compiler

The FSM Compiler and the FSM Explorer are automatic tools for state machines, but you can also specify FSMs manually with attributes. For more information about the FSM Explorer and FSM attributes, see *Using FSM Explorer*, on page 6-22, *Adding Attributes and Directives*, on page 3-66 and *Specifying FSMs with Attributes and Directives*, on page 6-15.

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 6-25.

- 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 *Using FSM Explorer,* on page 6-22.

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 New Impl or Impl 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 6-15.

   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 have specified encoding styles, it automatically selects the encoding style based on the number of states.

Number of States	Encoding Style
Up to 4	sequential
5-24	onehot
> 24	gray

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 6-25.
- 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 6-15. 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 New Impl or Impl 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 6-15. 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 6-25.

Using FSM Explorer

The FSM Explorer option is available only in the Synplify Pro and Synplify Premier tools. The Symbolic FSM Explorer is a specialized state machine optimizer that explores different encoding styles before selecting the best style. It uses the FSM Compiler to recognize and extract state machines. If the FSM Compiler has not been run, it runs it automatically. For information about the FSM Compiler, see *Using the Symbolic FSM Compiler,* on page 6-17.

This section discusses the following subtopics:

- Deciding When to Use the FSM Explorer, next
- Running the FSM Explorer, on page 6-23

Deciding When to Use the FSM Explorer

The FSM Explorer and the FSM Compiler are automatic tools for encoding state machines. 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.

In addition to the two automatic tools, you can always specify state machine encoding manually with attributes. For more information about the attributes, see *Specifying FSMs with Attributes and Directives,* on page 6-15.

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 true;
```

   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 6-15 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 New Impl or Impl 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.

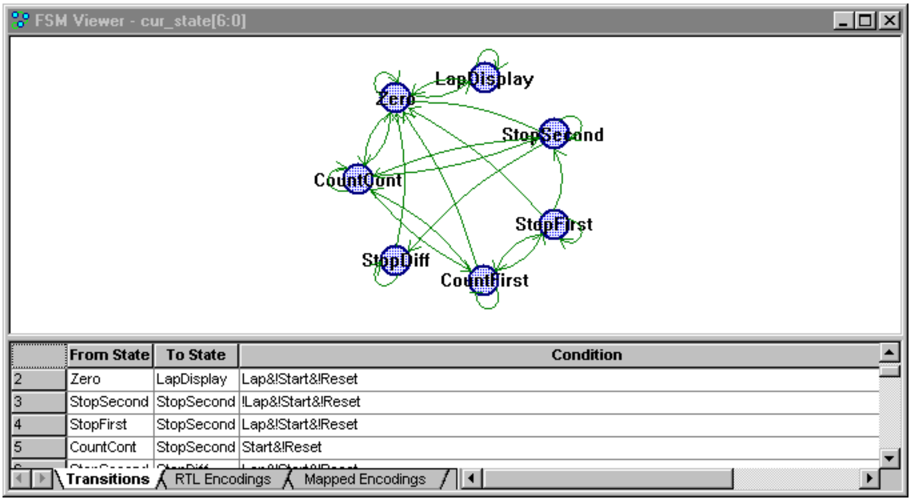
   For information about the FSM viewer, see *Using the FSM Viewer,* on page 6-25.

Using the FSM Viewer

The FSM Explorer 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 *Using the Symbolic FSM Compiler*, on page 6-17 and *Using FSM Explorer*, on page 6-22, 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.

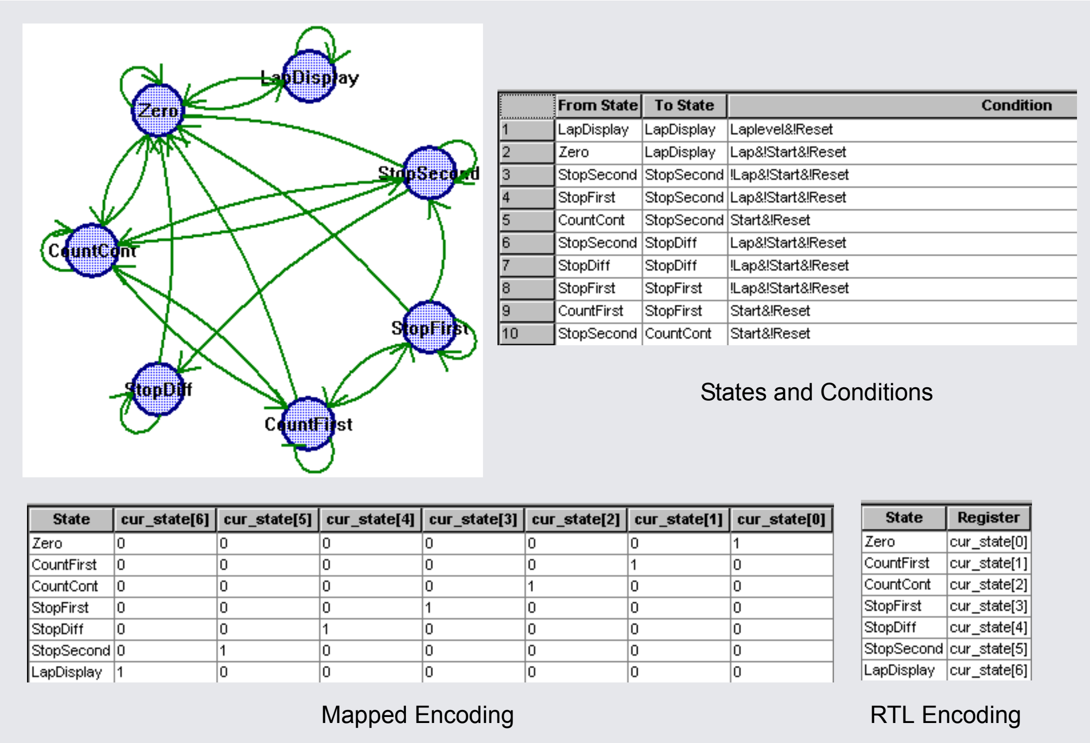


	From State	To State	Condition
2	Zero	LapDisplay	Lap&!Start&!Reset
3	StopSecond	StopSecond	!Lap&!Start&!Reset
4	StopFirst	StopSecond	Lap&!Start&!Reset
5	CountCont	StopSecond	Start&!Reset
6	StopSecond	StopDiff	Lap&!Start&!Reset

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.



	From State	To State	Condition
1	LapDisplay	LapDisplay	Laplevel&!Reset
2	Zero	LapDisplay	Lap&!Start&!Reset
3	StopSecond	StopSecond	!Lap&!Start&!Reset
4	StopFirst	StopSecond	Lap&!Start&!Reset
5	CountCont	StopSecond	Start&!Reset
6	StopSecond	StopDiff	Lap&!Start&!Reset
7	StopDiff	StopDiff	!Lap&!Start&!Reset
8	StopFirst	StopFirst	!Lap&!Start&!Reset
9	CountFirst	StopFirst	Start&!Reset
10	StopSecond	CountCont	Start&!Reset

States and Conditions

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

Mapped Encoding

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

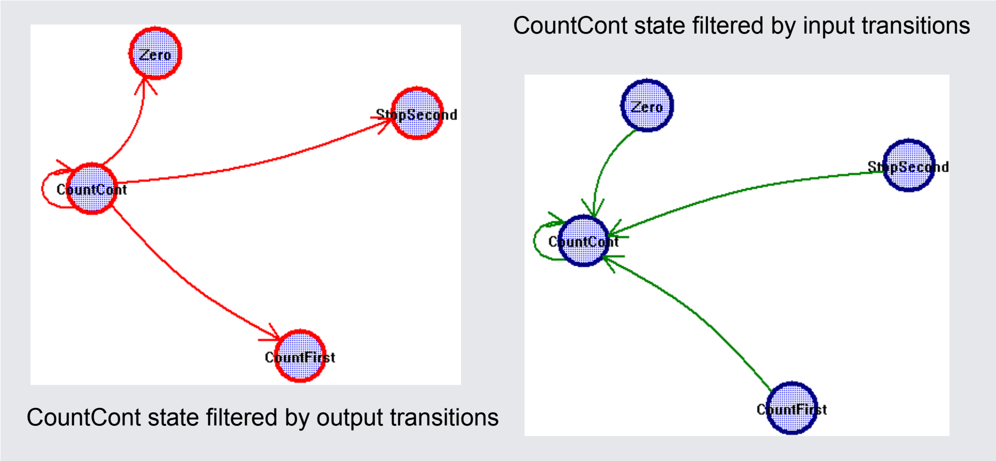
RTL Encoding

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.



CountCont state filtered by input transitions

CountCont state filtered by output transitions

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
```

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, next
- Instantiating Black Boxes and I/Os in VHDL, on page 6-32
- Adding Black Box Timing Constraints, on page 6-34
- Adding Other Black Box Attributes, on page 6-38

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 6-99.

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 abut instantiating black boxes in VHDL, see *Instantiating Black Boxes and I/Os in VHDL*, on page 6-32. Additional information about black boxes can be found in *Working with Gated Clocks*, on page 6-99, *Instantiating CoreGen Cores*, on page 8-29, and *Instantiating Virtex PCI Cores*, on page 8-30. 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"
```

   - 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 6-34 and *Adding Other Black Box Attributes*, on page 6-38.

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 abut instantiating black boxes in VHDL, see *Instantiating Black Boxes and I/Os in Verilog*, on page 6-30.

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 be 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: my_bbox port map (
   Q => q,
   D => data_core,
   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.

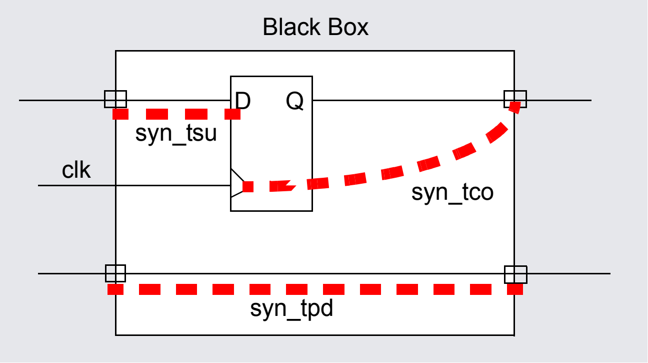
```
begin
data_pad: mybuf port map (
   O => data_core,
   I => data);
```

4. Add timing constraints and attributes. See *Adding Black Box Timing Constraints*, on page 6-34, *Gated Clocks for Black Boxes*, on page 6-108, and *Adding Other Black Box Attributes*, on page 6-38. 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 6-38 and *Gated Clocks for Black Boxes*, on page 6-108. 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 6-30 or *Instantiating Black Boxes and I/Os in VHDL*, on page 6-32.

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_tco11 : string;
      attribute syn_tco12 : string;
   ```

   – Define the constraints in either of these ways:

VHDL syntax	attribute *attribute_name<n>* : "*att_value*"
Verilog-style notation	attribute *attribute_name<n>* of *bbox_name* : component is "*att_value*"

   The following table shows the appropriate syntax for att_value. See the *Reference Manual* for complete syntax information.

Attribute	Value Syntax
syn_tsu<*n*>	*bundle* **->** [**!**]*clock* **=** *value*
syn_tco<*n*>	[**!**]*clock* **->** *bundle* **=** *value*
syn_tpd<*n*>	*bundle* **->** *bundle* **=** *value*

- <*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.

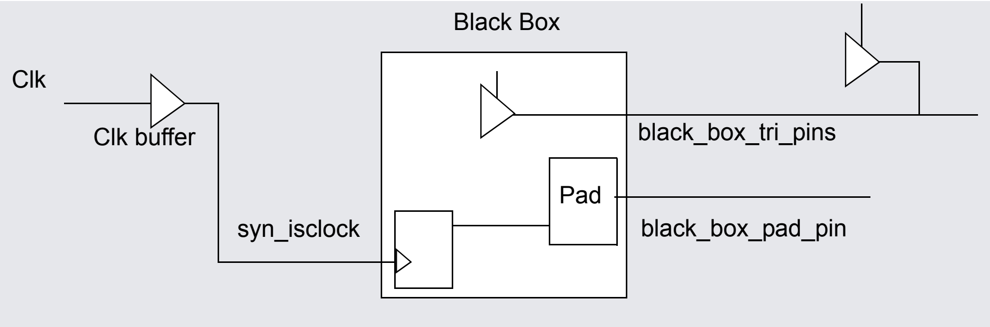
   The resulting constraint file contains syntax like this:

   `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 *Gated Clocks for Black Boxes,* on page 6-108. 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.

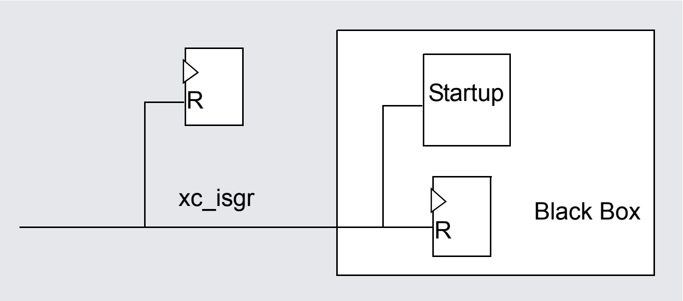
   Depending on the technology, different clock resources are inserted. In Xilinx, the software inserts BUFG, for Actel it inserts CLKBUF, and for QuickLogic, it inserts Q_CKPAD.

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, Lattice, 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 XC4000architectures, use the xc_isgr directive.



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 Altera or Xilinx technologies, you can use or the related technique of retiming to improve performance. See *Retiming*, on page 6-44 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, next
- Pipelining the Design, on page 6-41

Prerequisites for Pipelining

The pipelining feature is only available in the Synplify Pro and Synplify Premier tools.

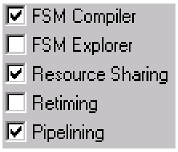
- Currently, pipelining is only supported for certain Altera and Xilinx technologies.
- In Xilinx Virtex designs, you can pipeline ROMs and multipliers. In Altera designs, you can pipeline multipliers, but not ROMs.
- For Xilinx Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-4 devices, you can 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

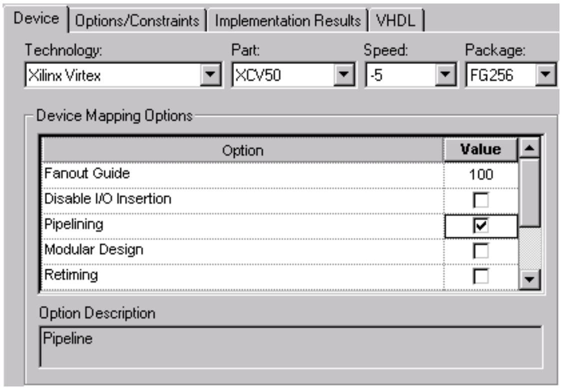
The following procedure shows you techniques for pipelining.

1. Make sure the design meets the criteria described in *Prerequisites for Pipelining*, on page 6-40.

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.



☑ FSM Compiler
☐ FSM Explorer
☑ Resource Sharing
☐ Retiming
☑ Pipelining



Technology: Xilinx Virtex
Part: XCV50
Speed: -5
Package: FG256

Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	☐
Pipelining	☑
Modular Design	☐
Retiming	☐

Option Description

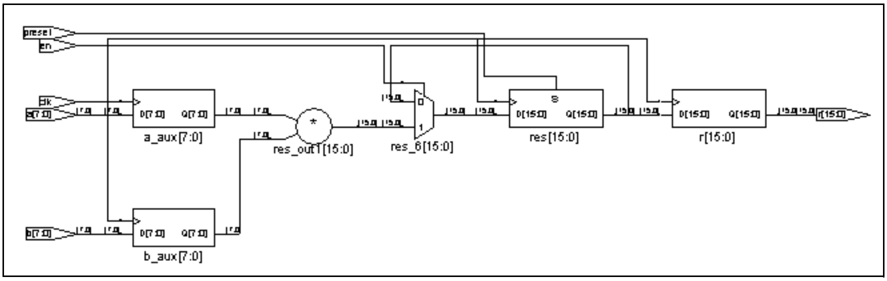
Pipeline

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.

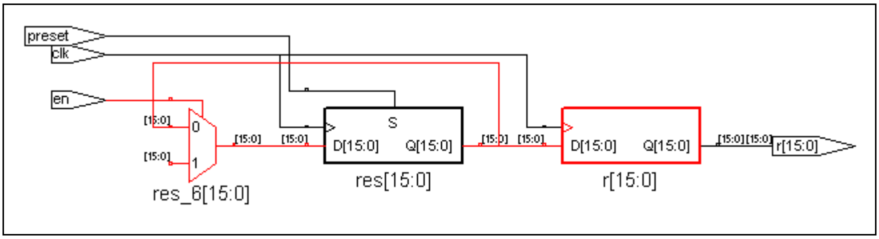
3. To check whether individual registers are suitable for pipelining, do the following:

   – 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.



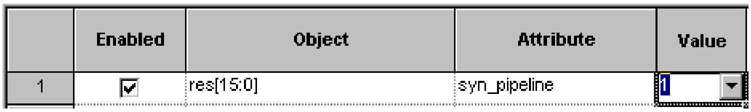
4. To enable pipelining on selected registers, use either of the following techniques:

   – 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	Attribute	Value
1	☑	res[15:0]	syn_pipeline	1

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 only available in the Synplify Pro and Synplify Premier tools. Retiming is a powerful technique for improving the timing performance of sequential circuits without having to modify the 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 only 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 detailed information about using retiming.

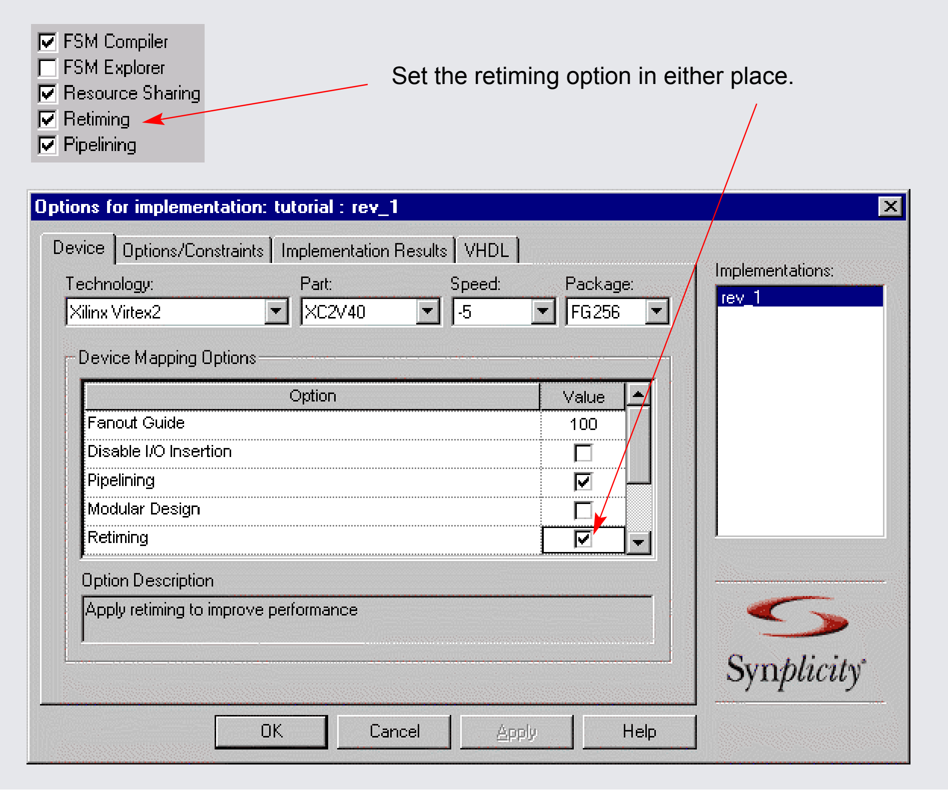
- Controlling Retiming, next
- Retiming Example, on page 6-46
- Retiming Report, on page 6-48
- How Retiming Works, on page 6-48

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.



☑ FSM Compiler
□ FSM Explorer
☑ Resource Sharing
☑ Retiming
☑ Pipelining

Set the retiming option in either place.

Options for implementation: tutorial : rev_1

Technology: Xilinx Virtex2
Part: XC2V40
Speed: -5
Package: FG256

Implementations: rev_1

Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	□
Pipelining	☑
Modular Design	□
Retiming	☑

Option Description

Apply retiming to improve performance

For Altera and Xilinx designs, retiming is a superset of pipelining, so when you select Retiming, you automatically select Pipelining. See *Pipelining*, on page 6-40 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 6-48 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 flops without sets or resets (FD or FDE) during retiming, set syn_preserve=1 or syn_allow_retiming=0 on these flops.

   - To force flops to be packed in I/O pads, set syn_useioff=1 as a global attribute. This will prevent the 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 6-48 for details.

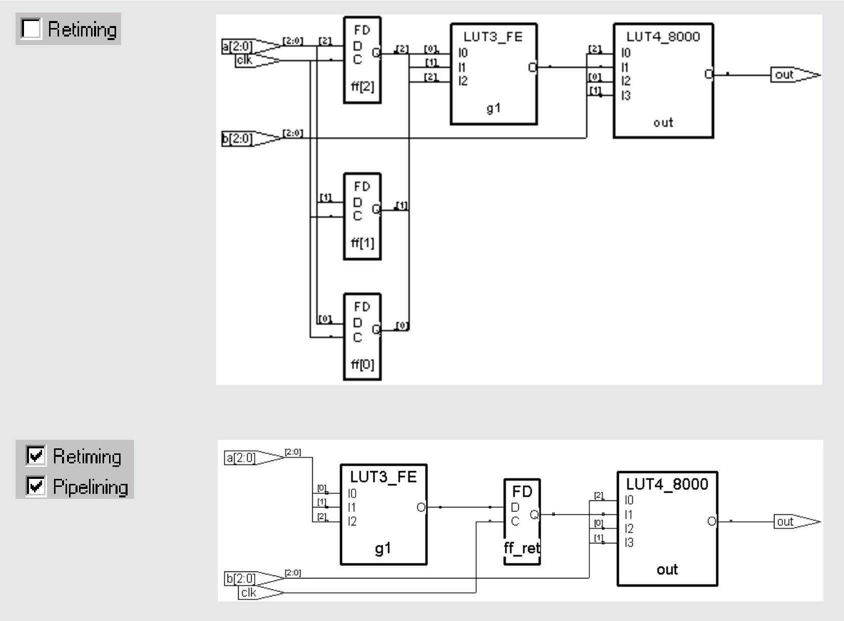
5. Click Run to start synthesis.

   After the LUTs are mapped, the software moves registers to optimize timing. See *Retiming Example,* on page 6-46 for an example. The software honors other attributes you set, like syn_preserve, syn_useioff, and syn_ramstyle. See *How Retiming Works,* on page 6-48 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 6-48 for more information about the report.

Retiming Example

The following example shows a design with retiming disabled and enabled.



☐ Retiming

☑ Retiming
☑ Pipelining

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 Tech view, because there may not be a one-to-one correspondence between the registers in these two views after retiming. A single register in the RTL view might now correspond to multiple registers in the Technology view.

- Retiming affects or is affected by these attributes and constraints:

Attribute/Constraint	Effect
False path constraint	Does not retime flip-flops with different false path constraints. Retimed registers affect timing constraints.
Multicycle constraint	Does not retime flip-flops with different multicycle constraints. Retimed registers affect timing constraints.
Register constraint	Does not maintain define_reg_input_delay and define_reg_output_delay constraints. Retimed registers affect timing constraints.
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 syn_reference_clock 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. It 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.

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 6-52.

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*.

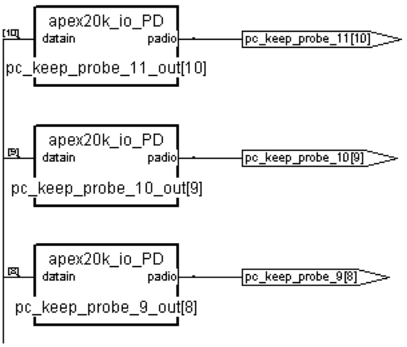
   To define probes for part of a bus, specify where you want to attach the probes; for example, if you specify reg [1:0] in the previous code, the software only inserts two probes.
3. For VHDL source code, add the syn_probe attribute as follows:

   ```
   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 property syn_probe, value 1, to net pc[0]
Adding property syn_probe, value 1, to net pc[1]
Adding property syn_probe, value 1, to net pc[2]
Adding property syn_probe, value 1, to net pc[3]
....
@N|Added probe pc_keep_probe_1[0] on pc_keep[0] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_1[0]
@N|Added probe pc_keep_probe_2[1] on pc_keep[1] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_2[1]
@N|Added probe pc_keep_probe_3[2] on pc_keep[2] in
eight_bit_uc
```

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 6-51.

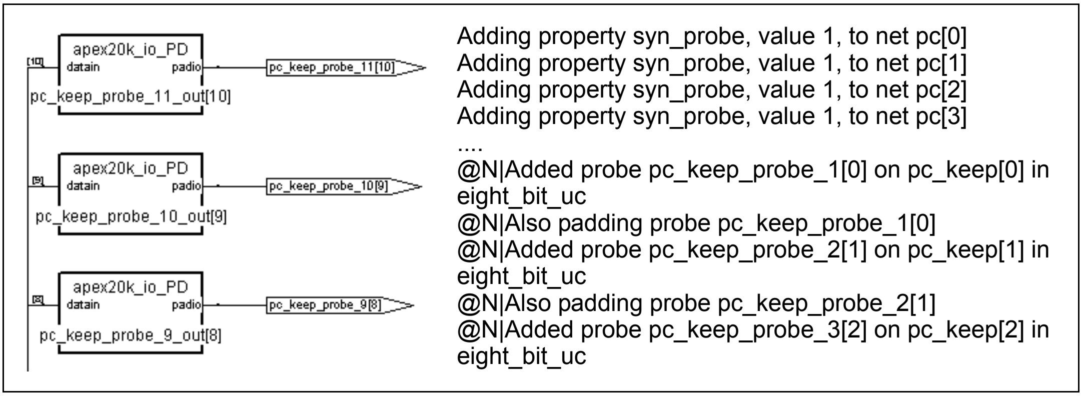
1. Open the SCOPE window and click Attributes.
2. Push down as necessary in an RTL view, and select the net for which you want to insert a probe point.

   Do not insert probes for output or bidirectional signals. If you do, you see warning messages in the log file.
3. Do the following to add the attribute:
   - Drag the net into a SCOPE cell.
   - Add the prefix n: to the net name in the SCOPE window. If you are adding a probe to a lower-level module, the name is created by concatenating the names of the hierarchical instances.
   - If you want to attach probes to part but not all of a bus, make the change in the Object column. For example, if you enter n:UC_ALU.longq[4:0] instead of n:UC_ALU.longq[8:0], the software only inserts probes where specified.

- Select syn_probe in the Attribute column, and type 1 in the Value column.
- Add the constraint file to the project list.

4. Rerun synthesis.
5. Open a Technology view and check the probe wires that have been inserted. You can use the Ports tab of the Find form to locate the probes.

   The software adds I/O pads for the probes. The following figure shows some of the pads in the Technology view and the log file entries.



```
Adding property syn_probe, value 1, to net pc[0]
Adding property syn_probe, value 1, to net pc[1]
Adding property syn_probe, value 1, to net pc[2]
Adding property syn_probe, value 1, to net pc[3]
....
@N|Added probe pc_keep_probe_1[0] on pc_keep[0] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_1[0]
@N|Added probe pc_keep_probe_2[1] on pc_keep[1] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_2[1]
@N|Added probe pc_keep_probe_3[2] on pc_keep[2] in
eight_bit_uc
```

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 vs. Instantiation, next
- Coding RAMs for Inference, on page 6-55
- Specifying RAM Implementation Styles, on page 6-59
- Implementing Altera RAMs Automatically, on page 6-61
- Implementing Xilinx RAMs Automatically, on page 6-64
- Implementing Altera RAMs: FLEX and APEX, on page 6-67
- Implementing Altera RAMs: Stratix Multi-Port RAMs, on page 6-69
- Inferring Xilinx Block RAMs Using Registered Addresses, on page 6-70
- Inferring Xilinx Block RAMs Using Registered Output, on page 6-73
- Setting Xilinx RAM Initialization Values, on page 6-78
- Mapping Xilinx ROM to Block RAM, on page 6-79

Inference vs. 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	Instantiation
Advantages	**Advantages**
Portable coding style Automatic timing-driven synthesis No additional tool dependencies	Most efficient use of the RAM primitives of a specific technology Supports all kinds of RAMs

Inference in Synthesis	Instantiation
Limitations	**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	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.

Coding RAMs for Inference

Read through the limitations before you start. See *Inference vs. Instantiation,* on page 6-54 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 6-59, *Implementing Altera RAMs Automatically,* on page 6-61, and *Implementing Xilinx RAMs Automatically,* on page 6-64.

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:

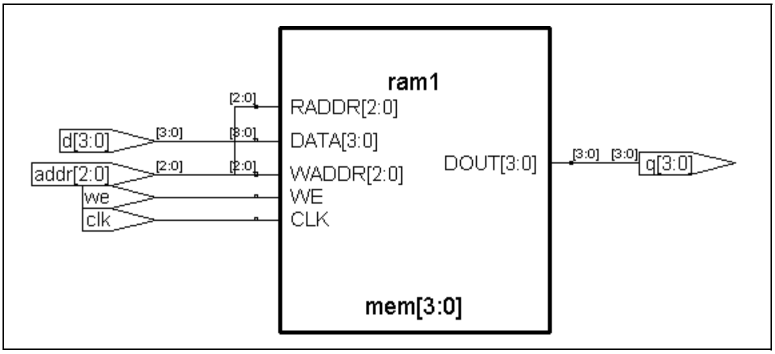
   – To infer a RAM, structure the code as an indexed array or a case structure. Code it as a two-dimensional array (VHDL) or memory (Verilog) with writes to one process.

   – 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.

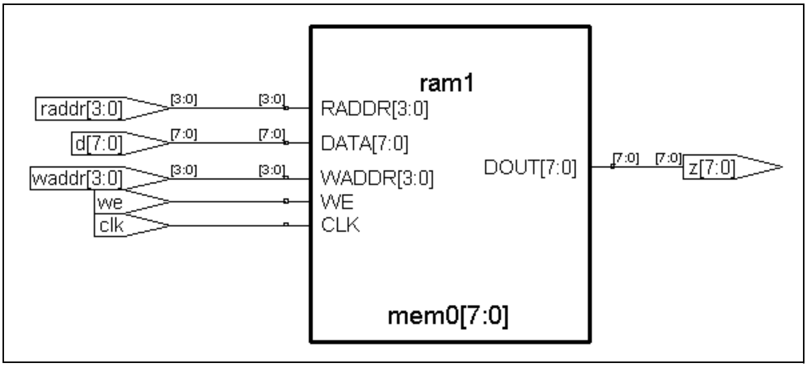
3. For a single-port RAM, make the address for indexing the write-to the same as the address for the read-from. The following code and figure illustrate how the software infers a single-port RAM.

```
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, we, addr) 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 6-61 and *Implementing Xilinx RAMs Automatically*, on page 6-64.

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 6-61 and *Implementing Xilinx RAMs Automatically*, on page 6-64.

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 is 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 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 vs. Instantiation,* on page 6-54 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 implemention you want, see *Specifying RAM Implementation Styles,* on page 6-59, *Implementing Altera RAMs Automatically,* on page 6-61, and *Implementing Xilinx RAMs Automatically,* on page 6-64.

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 6-61 and *Implementing Xilinx RAMs Automatically*, on page 6-64 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 (Atmel, Cypress), 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, you can set syn_ramstyle to select_ram. If you do not set syn_ramstyle

explicitly, the software automatically uses this value, because it is the default.

Implementing Altera 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 6-59 or instantiate LPMs instead of using RAMs.

1. Follow the guidelines described for RAM inference by the compiler (*Coding RAMs for Inference,* on page 6-55).

   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 RAMs: FLEX and APEX,* on page 6-67.

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.

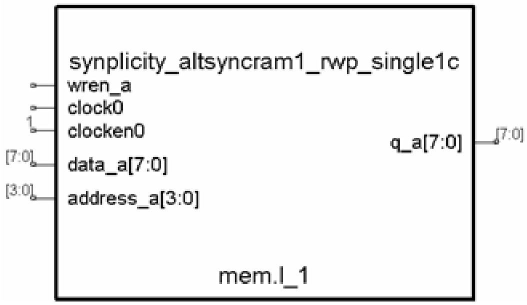
   During synthesis, the mapper maps Altera Stratix RAM to ALTSYNCRAM in the following modes:

Single-port	One address bus
Dual-port	One address bus (where old data cannot be obtained in single-port mode), or 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.

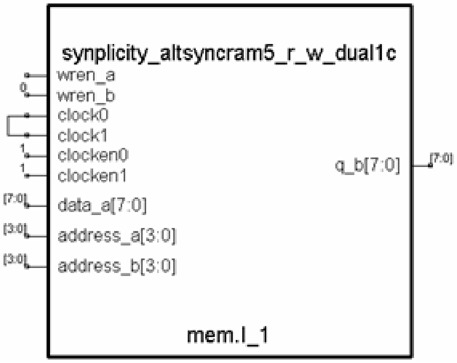


5. To implement Stratix dual-port RAMs, make sure of the following:
   - The code is written so that the hardware exactly matches the RTL behavior. For example, if your code allows simultaneous reads and writes to the same address, it can result in a mismatch between the RTL and hardware behaviors. In 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 *Stratix Dual-Port RAM Code Examples,* on page B-42 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 *Stratix Dual-Port RAM Code Examples*, on page B-42 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 RAMs: Stratix Multi-Port RAMs,* on page 6-69.

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 6-59.

1. Follow the guidelines described for RAM inference by the compiler (*Coding RAMs for Inference,* on page 6-55).

   The Xilinx mapper does not implement any RAMs that are not first inferred by the compiler.

2. To automatically implement distributed RAM, do the following:

   – Make sure the RAM size is at least 2K.

   – 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 6-70).

   – 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 6-73).
- 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 6-73).
   - 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 (data_in goes to data_out)	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3 Automotive, and Spartan-3E	Block SelectRAM+ (single-port or dual-port) Distributed RAM
	Virtex, Virtex-E, Spartan-II, Spartan-IIE, and Spartan-IIE Automotive	Block SelectRAM+ (single-port or dual-port) Distributed RAM
Readfirst (memory goes to data_out)	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3 Automotive, and Spartan-3E	Block SelectRAM+ (single-port) Distributed RAM
	Virtex, Virtex-E, Spartan-II, Spartan-IIE, and Spartan-IIE Automotive	Distributed RAM
Nochange (data_out is unchanged)	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3 Automotive, and Spartan-3E	Block SelectRAM+ (single-port) Distributed RAM
	Virtex, Virtex-E, Spartan-II, Spartan-IIE, and Spartan-IIE 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 *Coding RAMs for Inference*, on page 6-55 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 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 RAMs: FLEX and APEX

The alternative to inferring RAMs in an Altera design is to instantiate LPMs. See *Inference vs. Instantiation*, on page 6-54 and *Working with LPMs*, on page 6-87.

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 *Coding RAMs for Inference*, on page 6-55.
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-portRAMs 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 6-59 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 APEX20K APEX20KE	Single-port or dual-port RAMs with asynchronous READs	ALTDPRAM

Implementing Altera RAMs: 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 *Coding RAMs for Inference,* on page 6-55.

   For multi-port RAMs, the compiler infers an *n*ram 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; *n*ram 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 *n*ram 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 nrams 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 Xilinx Block RAMs Using Registered Addresses

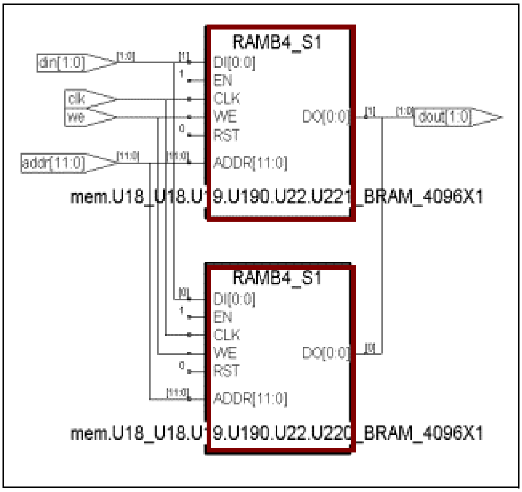
There are two ways to infer block RAMs in Xilinx Virtex designs: using registered addresses and using registered output. For information about the latter, see *Inferring Xilinx Block RAMs Using Registered Output*, on page 6-73. The following procedure shows you how to set up your code with an explicit 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]
```



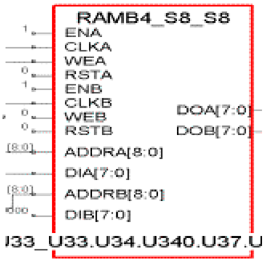
- To forward-annotate initialization values, use the Xilinx INIT property, as described in *Setting Xilinx RAM Initialization Values,* on page 6-78.

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]
```



+ Glue logic to resolve read/write discrepancies

– To forward-annotate initialization values, see *Setting Xilinx RAM Initialization Values,* on page 6-78.

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 6-59 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 6-59 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 6-70). For information about forward-annotating initialization values, see *Setting Xilinx RAM Initialization Values*, on page 6-78.

This information is organized into these subtopics:

- Advantages of Using Registered Output, on page 6-73
- Block RAM Mapping for Virtex-II Write Modes, on page 6-73
- Xilinx Single-Port Example with Registered Output, on page 6-75
- Xilinx Single-Output Dual-Port Example with Registered Output, on page 6-77

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 6-75 and *Xilinx Single-Output Dual-Port Example with Registered Output*, on page 6-77 for examples.

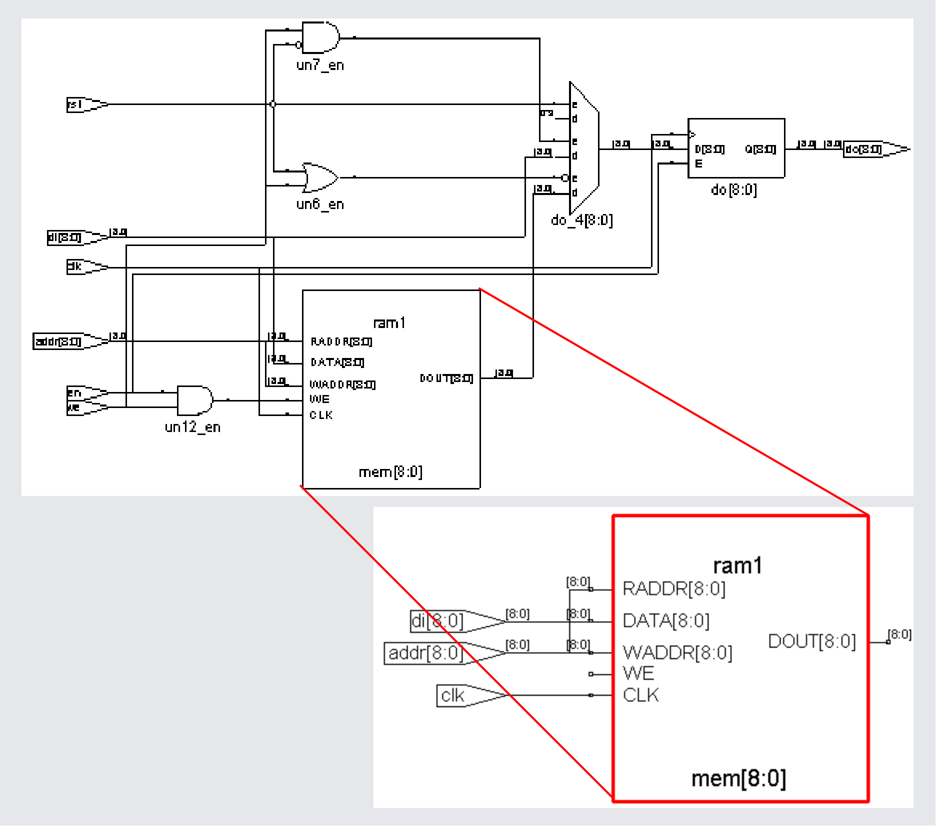
	Virtex	**Virtex-E**	**Virtex-II**	**Virtex-II Pro**
	WRITEFIRST Mode			
With enable and reset, enable takes precedence	SP	SP	SP	SP
With enable and reset, reset takes precedence	SP	SP	SP	SP
Without enable	SP	SP	SP	SP
Without reset	SP	SP	SP	SP
Without enable or reset	SP	SP	SP	SP
	READFIRST Mode			
With enable and reset, enable takes precedence	Select RAM	Select RAM	SP	SP
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
	NOCHANGE Mode			
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 *Setting Xilinx RAM Initialization Values*, on page 6-78.



```
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;
```

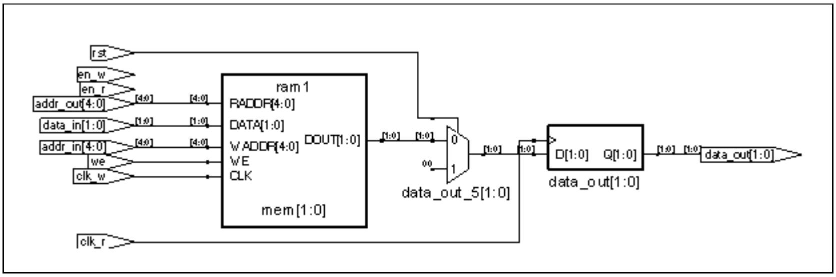
Xilinx Single-Output Dual-Port Example with Registered Output

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 *Setting Xilinx RAM Initialization Values*, on page 6-78.

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;
```



Setting Xilinx RAM Initialization Values

You can forward-annotate RAM initialization values using the Xilinx INIT property.

1. Add the INIT property. 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.

   – In VHDL 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*";

   – In Verilog, attach the INIT property to the instance as shown:

RAM	/* synthesis INIT = "*value*" */
Block RAM	/* synthesis INIT_xx = "*value*" */

2. For RAM, specify a hex value for the INIT statement as shown in these examples:

Verilog	RAM16X1S RAM1(...) /* synthesis INIT = "0000" */;
VHDL	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.

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:

Asynchronous	Synchronous
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.

Inferring Shift Registers

The software infers shift registers for Xilinx Virtex and Altera Stratix architectures. Use the following procedure.

1. Set up the HDL code for the sequential shift components. See *Shift Register Examples*, on page 6-82 for examples.

   Note the following for Xilinx shift registers:

   – The new component represents a set of three or more registers that can be shifted left (from a low address to a higher address).

   – The contents of only one register can be seen at a time, based on the read address.

   – 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.

2. If needed, set the implementation style with the syn_srlstyle attribute. If you do not want the components automatically mapped to shift registers, set the value to registers.

syn_srlstyle Value	Implemented as...
registers	registers
select_srl	Xilinx SRL16 primitives
no_extractff_srl	Xilinx SRL16 primitives without output flip-flops
altshift_tap	Altera Altshift_tap components

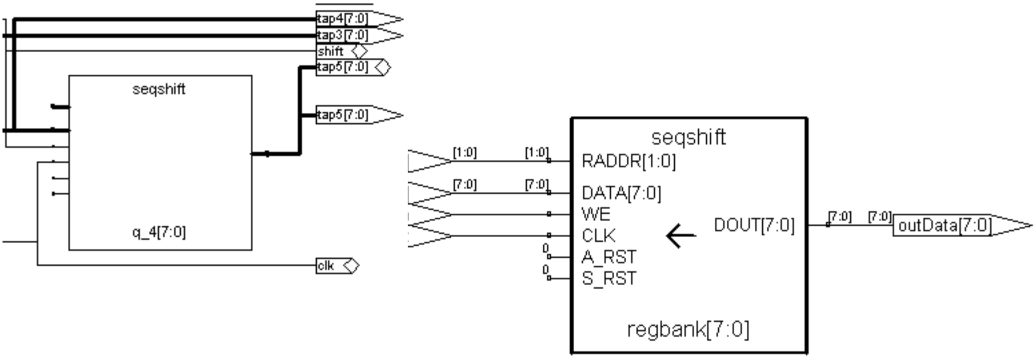
   You can set the value globally or on individual modules or registers.

3. For Altera shift registers, use attributes to control how the registers are packed:

To...	Attach...
Prevent a register from being packed into shift registers	syn_useioff or syn_noprune to the register. You can also use syn_srlstyle with a value of registers.
Prevent two registers from being packed into the same shift registers	syn_keep 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	syn_srlstyle with different group names for the registers you want to separate (syn_srlstyle= altshift_tap, *group_name*)

4. 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.

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 DOWNTO 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
```

Forward Annotation of Initial Values

Initial values for RAMs and sequential shift components are forward annotated to the netlist. The compiler currently generates netlist (`.srs`) files with seqshift, ram1, ram2, and nram components. If initial values are specified in the Verilog code, the synthesis tool attaches an attribute to the component in the `.srs` file.

For Verilog designs, a separate `data` file which contains the initial values must be created and placed in the project directory. For more information on initial values, see *Initial Values in Verilog*, on page 9-28.

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.

	Black Box Method	**Verilog Library/VHDL Prepared Component Method**
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 LPMs as Black Boxes (Altera), on page 6-88, next
- Instantiating LPMs as Black Boxes (Cypress), on page 6-92
- Instantiating LPMs Using VHDL Prepared Components, on page 6-94
- Instantiating LPMs Using a Verilog Library (Altera), on page 6-97

Instantiating LPMs as Black Boxes (Altera)

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 LPMs Using a Verilog Library (Altera),* on page 6-97 or *Instantiating LPMs Using VHDL Prepared Components,* on page 6-94. For information about using Clearbox in Synplify Pro Stratix designs, see *Implementing Megafunctions with Clearbox,* on page 8-16.

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 6-88 and *LPM Megafunction Example (VHDL),* on page 6-90.

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 6-88 and *LPM Megafunction Example (VHDL),* on page 6-90.

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.widthad_a = 5,
      altsyncram_component.numwords_a = 32,
      altsyncram_component.width_b = 8,
      altsyncram_component.widthad_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 LPMs as Black Boxes (Cypress)

You can instantiate Cypress LPMs as black boxes if you define the LPM parameters as comments. The following procedure shows you how to do this. Alternatively, you can use the method described in *Instantiating LPMs Using VHDL Prepared Components*, on page 6-94.

1. Define a black box for the LPM using syn_black_box. For details, see *Defining Black Boxes for Synthesis*, on page 6-30.
2. Assign LPM-specific attributes like LPM_TYPE and LPM_WIDTH. The LPM_TYPE attribute is required and must be set to the exact name of the LPM.

   These attributes are not used for synthesis but are passed to the place-and-route tools. For a full list of available ports and parameters, see the Cypress documentation. For code examples, see *Verilog LPM Example (Cypress)*, on page 6-92 and *VHDL LPM Example (Cypress)*, on page 6-93.
3. Instantiate the LPM in the higher-level module as shown in the following code examples.

Verilog LPM Example (Cypress)

The following Verilog example shows a MADD_SUB component in an adder:

```
module my_madd_sub(dataa, datab, cin, add_sub, result, cout,
overflow)
/* synthesis syn_black_box
   lpm_width = 4
   lpm_representation = "LPM_UNSIGNED"
   lpm_direction = "LPM_NO_TYP"
   lpm_hint = "SPEED"
   lpm_type = "MADD_SUB"
*/;
output [3:0] result;
output cout;
output overflow;
input [3:0] dataa;
input [3:0] datab;
input cin;
input add_sub;
endmodule
```

```
module adder(r, a, b);
output [3:0] r;
input [3:0] a, b;
my_madd_sub inst0(.dataa(a), .datab(b), .cin(0), .add_sub(1),
.result(r), .cout(), .overflow());
endmodule
```

VHDL LPM Example (Cypress)

This is a VHDL example:

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

entity adder is port (
      dataA : in std_logic_vector(7 downto 0);
      dataB : in std_logic_vector(7 downto 0);
      sum : out std_logic_vector(7 downto 0));
end adder;

architecture archadder of adder is
   component madd_sub
   port   (dataa: in std_logic_vector(7 downto 0);
      datab : in std_logic_vector(7 downto 0);
      cin : in std_logic := '0';
      add_sub: in std_logic := '1';
      result: out std_logic_vector(7 downto 0);
      cout : out std_logic;
      overflow: out std_logic);
   end component;

   attribute black_box : boolean;
   attribute black_box of madd_sub : component is true;
   attribute lpm_width : positive;
   attribute lpm_width of madd_sub : component is 8;
   attribute lpm_representation : string;
   attribute lpm_representation of madd_sub : component is
            "LPM_UNSIGNED";
   attribute lpm_direction : string;
   attribute lpm_directionof madd_sub : component is "LPM_NO_TYP";
   attribute lpm_type : string;
   attribute lpm_type of madd_sub: component is "madd_sub";
   attribute lpm_hint : string;
   attribute lpm_hint of madd_sub: component is "SPEED";
```

```
begin
U1: Madd_sub   -- Configured as an adder
   port map (
      dataA => dataA,
      dataB => dataB,
      --    cin => zero,
      --    add_sub => one,
      result => sum,
      cout => open,
      overflow => open
      );
end archadder;
```

Instantiating LPMs Using VHDL Prepared Components

The prepared components method is the simplest to use, but it does not cover all available LPMs. For other methods, see *Instantiating LPMs as Black Boxes (Cypress), on page 6-92*, *Instantiating LPMs as Black Boxes (Altera), on page 6-88*), or *Instantiating LPMs Using a Verilog Library (Altera), on page 6-97* (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`, and the Cypress prepared components are in `cp_lpmpkg`.

2. Instantiate the prepared component. For a list of Cypress prepared components, see *List of Prepared LPM Components (VHDL), on page D-4* in the *Reference Manual*. For Altera LPMs, see *Instantiating LPMs in VHDL, on page B-76* in the *Reference Manual*.

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 (Cypress)

The following example shows you how to implement a MADD_SUB component in the design of an adder.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library cypress;
use cypress.lpmpkg.all;

entity adder is port (
   dataA : in std_logic_vector(7 downto 0);
   dataB : in std_logic_vector(7 downto 0);
   sum : out std_logic_vector(7 downto 0));
end adder;

architecture archadder of adder is
begin

U1: MADD_SUB -- Configured as an adder
   generic map (
         lpm_width => 8,
         lpm_representation => lpm_unsigned,
         lpm_direction => lpm_add,
         lpm_hint => speed
         );
   port map (
         dataA => dataA,
         dataB => dataB,
         --    cin => zero,
         --    add_sub => one,
         result => sum,
         cout => open,
         overflow => open
         );
end archadder;
```

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_WIDTHAD => 4,
                   LPM_TYPE => "LPM_RAM_DQ")
      port map (data => data,
                   address => address,
                   we => we,
                   inclock => clock,
                   outclock => clock,
                   q => q);
end arch1;
```

Instantiating LPMs Using a Verilog Library (Altera)

For Altera LPMs, you can also instantiate LPMs from a Verilog library. For other methods of instantiating LPMs, see *Instantiating LPMs as Black Boxes (Altera)*, on page 6-88 and *Instantiating LPMs Using VHDL Prepared Components*, on page 6-94.

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_widthad = 1;
parameter numwords = 1<<lpm_widthad;
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_widthad-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, rdaddr;
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),
   .wraddress(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 8-20.

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 and Xilinx technology families. The information is organized into these sections:

- The Synplicity Approach to Gated Clocks, next
- Synthesizing a Gated Clock Design, on page 6-101
- Prerequisites for Gated Clock Conversion, on page 6-103
- Gated Clock Conversion Report, on page 6-105
- Gated Clocks for Black Boxes, on page 6-108
- Restrictions to Using Fix Gated Clocks, on page 6-110
- Generated-Clock Optimization, on page 6-111

The Synplicity Approach to Gated Clocks

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.

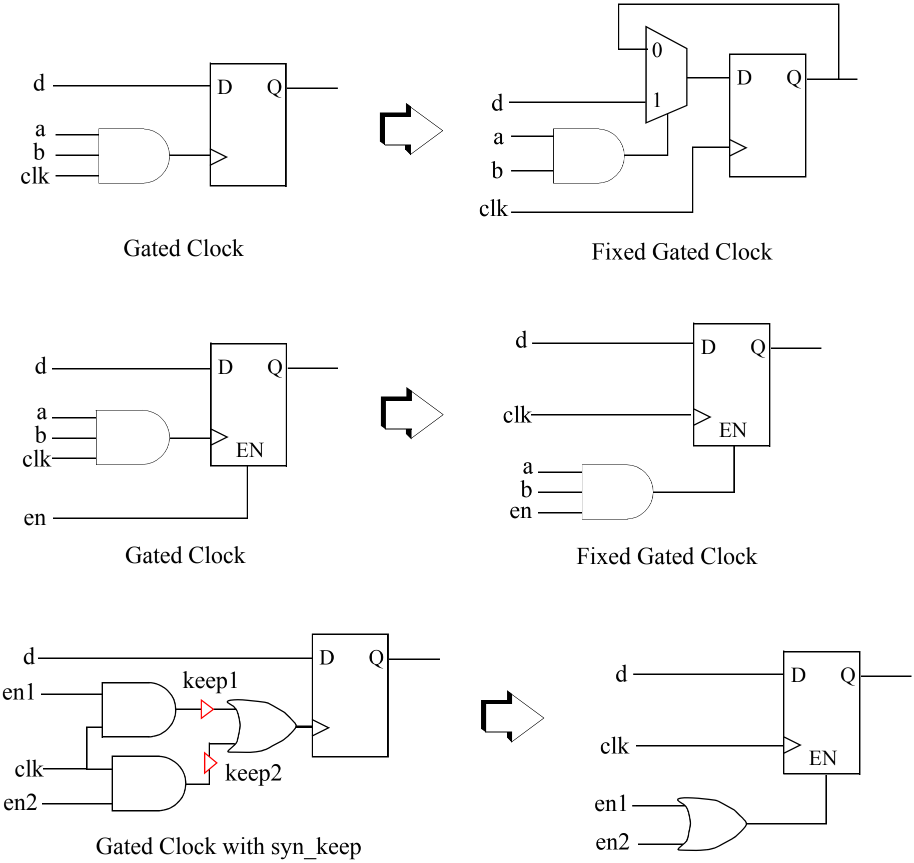
If you use the dedicated FPGA global clock trees instead, you free up routing resources and expedite placement and routing. Dedicated FPGA clock trees are routed to every sequential device on the die and are designed with low skew to avoid hold-time violations. Using these global clock trees allows the 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.



Gated Clock

Fixed Gated Clock

Gated Clock

Fixed Gated Clock

Gated Clock with syn_keep

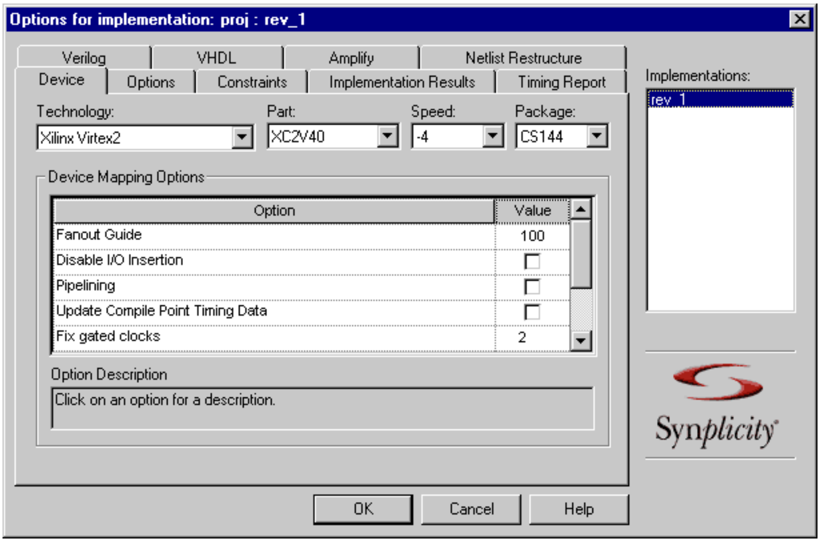
Fixed Gated Clock

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 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 6-103 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 *Gated Clocks for Black Boxes*, on page 6-108 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.



Options for implementation: proj : rev_1

Technology: Xilinx Virtex2
Part: XC2V40
Speed: -4
Package: CS144

Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	□
Pipelining	□
Update Compile Point Timing Data	□
Fix gated clocks	2

Option Description
Click on an option for a description.

Implementations:
rev_1

- 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	The default. Does not report any gated clock conversions.
2	Only reports sequential elements that could not be converted.
3	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 to Using Fix Gated Clocks*, on page 6-110 for additional information.

6. Check the results in the Gated Clock Report section of the log file. See *Gated Clock Conversion Report*, on page 6-105 for an example of this report.

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. `define_clock -name {clk} -freq 10.000 -clockgroup default_clkgroup`
Supported primitives	The sequential primitive clocked by the gated clock must be supported by Synplicity. Synplicity supports 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 *Gated Clocks for Black Boxes,* on page 6-108.
Correct logic format	See *Correct Logic Format,* on page 6-103 for an example of the correct logic format.

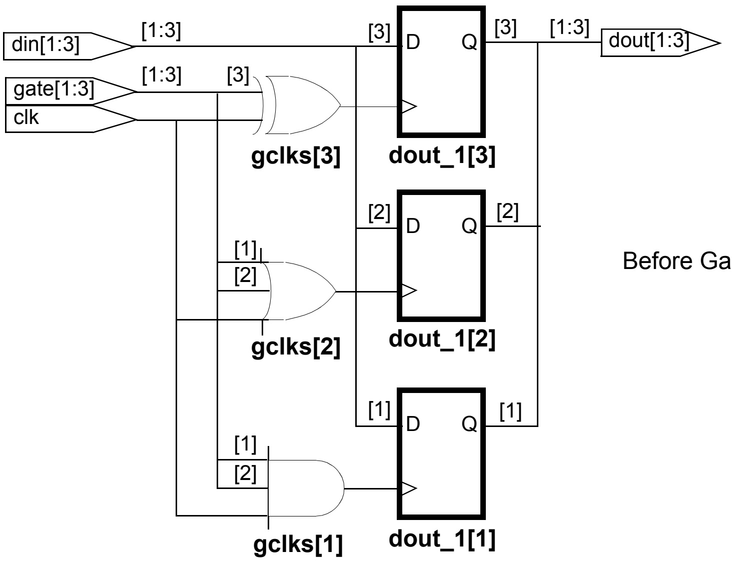
Correct Logic Format

Specifically, the combinational logic for the gated clock must satisfy the following two conditions to have the correct format:

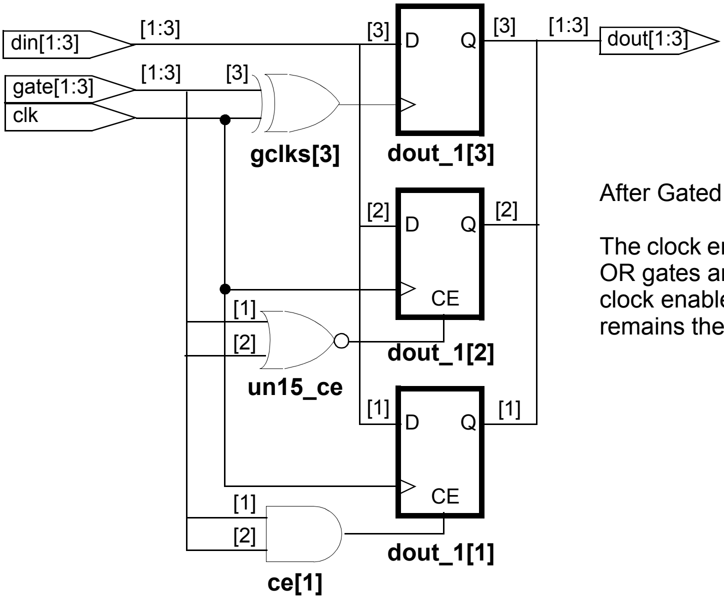
- For at least one set of gating input values, the value output for the gated clock must be constant and not change as the base clock changes.
- For at least one value of the base clock, changes in the gating input must not change the value output for the gated clock.

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 Cock option enabled, clock enables for the AND gate and OR gate are converted, but the exclusive-OR gate shown in the second figure is not converted. The following table explains.

AND gate gclks[1]	If either gate[1] or gate[2] is "0," then gclks[1] is "0," independent of the value of clk which satisfies the first condition. Also, if clk is "0," then gclks[1] is "0," independent of the values of gate[1] and gate[2] which satisfies the second condition. Because gclks[1] satisfies both conditions, it is successfully converted to the clock-enable format.
OR gate gclks[2]	If either gate[1] or gate[2] is "1," then gclks[2] is "1" independent of the value of clk which satisfies the first condition. Also, if clk is "1," then gclks[2] is "1" independent of the value of gate[1] or gate[2] which satisfies the second condition. Because gclks[2] satisfies both conditions, it is successfully converted to the clock-enable format.
Exclusive-OR gate gclks[3]	Irrespective of the value of gate[3], gclks[3] continues to toggle. The exclusive-OR function causes gclks[3] to fail both conditions which prevents gclks[3] from being converted.



Before Gated Clock Conversion



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.

Gated Clock Conversion Report

The value of the Fix Gated Clocks option determines how the conversions in the log file are reported:

Value	Effect
1	The default. Does not report any gated clock conversions.
2	Only reports sequential elements that could not be converted.
3	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 6-103 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
=======================================================
```

Fix 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 syn_gatedclk_data_in and/or syn_gatedclk_data_out 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	The gated clock conversion code cannot determine which clock to use because of one of the following: • There is no user clock driving the sequential element through the gating logic. • There are multiple user-defined clocks driving the gating logic.
Gating structure creates Improper gating logic	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: • For certain combinations of the gating signals, the gated clock signal must be capable of being disabled • For the remaining combinations of the gating signals, the gated clock signal equals either the clock signal or its inverted value • 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
Instance has no clock pin	The sequential gate does not have a clock pin.
Latch clock driven by an OR gate	The latch is gated by an OR-gate or an OR-gate equivalent and cannot be converted.
Library cell is not marked as sequential	The library cell has been marked as non sequential, with the property syn_force_seq_prim 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).

Error Message	Explanation
Not in chip	The clock driver is in another FPGA, not in the FPGA in which the sequential element is present.
Property *dontfixgatedclock* found	There is a syn_dontfixgatedclock 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 nable to determine the reason why gated clock conversion is failing. Contact Synplicity Support.
User asserted syn_keep found on gated clock logic	There is a user-asserted syn_keep on one of the gates in the gating logic or one of the nets found in the gating logic. This prevented gated clock conversion.

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

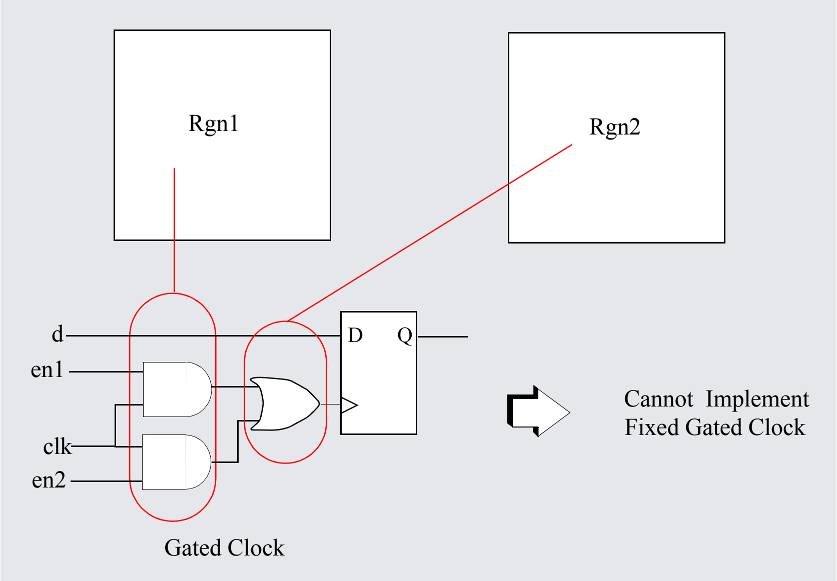
```
library synplify;
use synplify.attributes.all;

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;
```

Restrictions to Using Fix 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 *The Synplicity Approach to Gated Clocks*, on page 6-99.
- The Fix Gated Clocks option cannot be implemented for instantiated and inferred RAMs in Altera technologies.
- The Fix Gated Clocks option cannot be implemented for inferred counters in Altera technologies.
- 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.

Note: 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.



Generated-Clock Optimization

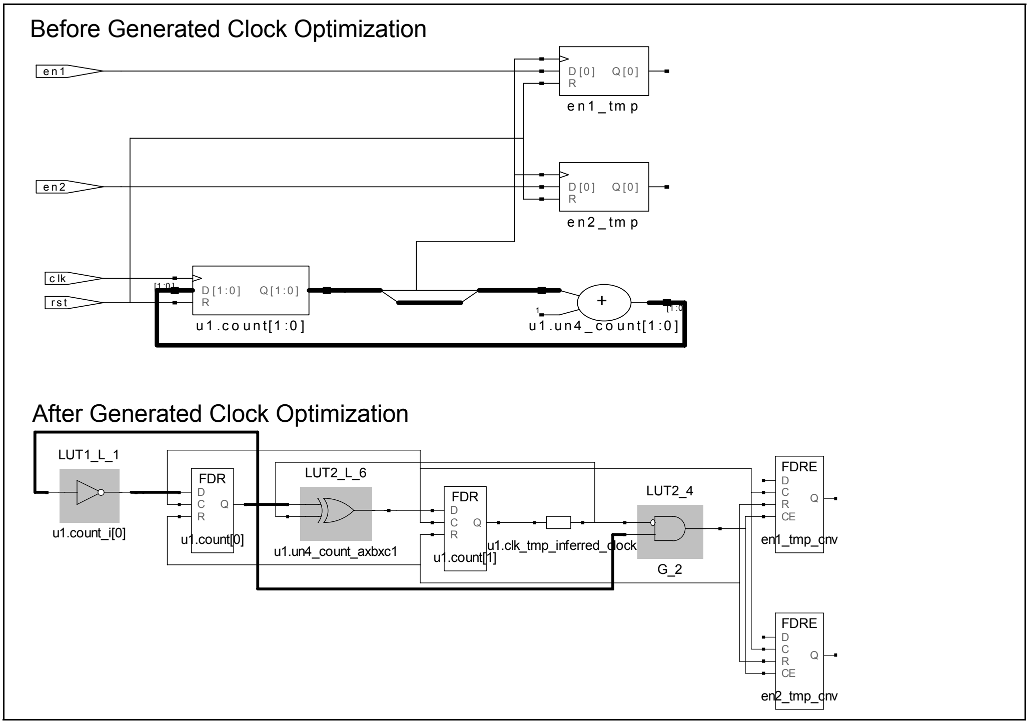
For Altera and Xilinx families, generated-clock optimization is included with the Fix Gated Clocks option. When the option is enabled, the generated clock logic is replaced with logic that uses the initial clock with an enable. With generated-clock optimization, the original circuit functionality is preserved and performance is improved by reducing clock skew.

Generated-Clock Optimization Example

The following code is optimized for a generated clock:

```
module ram(data0,data1,waddr0,waddr1,we0,we1,
         clk0,clk1,q0,q1);

parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;

input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;

reg [addr_width-1:0] reg_addr0, reg_addr1, reg_addr2;
reg [d_width-1:0] mem [mem_depth-1:0] /* synthesis syn_ramstyle =
         "no_rw_check" */ ;

assignq0 = mem[reg_addr0];
assignq1 = mem[reg_addr1];

always @(posedge clk0)
begin
   reg_addr0 <= waddr0;
   if (we0)
      mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
   reg_addr1 <= waddr1;
   if (we1)
      mem[waddr1] <= data1;
end

endmodule
```

The following figure, shows a circuit before and after optimization for generated clocks.


Before Generated Clock Optimization
en1
en2
clk
rst
D[0] Q[0]
R
en1_tmp
D[0] Q[0]
R
en2_tmp
[1:0]
D[1:0] Q[1:0]
R
u1.count[1:0]
+
1
[1:0]
u1.un4_count[1:0]
After Generated Clock Optimization
LUT1_L_1
u1.count_i[0]
FDR
D
C Q
R
u1.count[0]
LUT2_L_6
u1.un4_count_axbxc1
FDR
D
C Q
R
u1.count[1]
u1.clk_tmp_inferred_clock
LUT2_4
G_2
FDRE
D
C Q
R
CE
en1_tmp_cnv
FDRE
D
C Q
R
CE
en2_tmp_cnv

Enabling Generated-Clock Optimization

Generated-clock optimization is enabled by entering a non-zero value in the Fix Gated Clock Value field in the Device tab of the Options for implementation dialog box. The following table describes the options.

Fix Gated Clock Value	Description
0	Disable generated-clock optimization.
1	Perform optimization with no messages.
2	Perform optimization and report unoptimized sequential elements.
3	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 FF1 and FF2 in the previous figure, cannot have an active set or reset.

   For example, if FF1 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.

CHAPTER 7

Design Planning and Optimizations

Using the Synplify Premier Design Planner tool, you create a design plan to physically constrain portions of a design to specific regions on a device. Netlist restructure files usually contain primitives that have been bit sliced or modules that have been zippered. The design plan (.sfp) as well as netlist restructure files are used during optimization to improve the overall design performance. It is important to place physical constraints carefully. This section presents the following topics:

You should be familiar with the recommended Synplify Premier design flows and the Design Planner tool to use these design planning tips effectively.

Using the Design Planner

This section discusses the following topics:

- Creating a Design Plan on page 7-2
- Cutting, Copying, and Pasting in the Design Planner on page 7-5

For more information about the Design Planner, see the *Synplicity FPGA Synthesis Reference Manual*.

Creating a Design Plan

After the design is compiled, you create a design plan by doing the following:

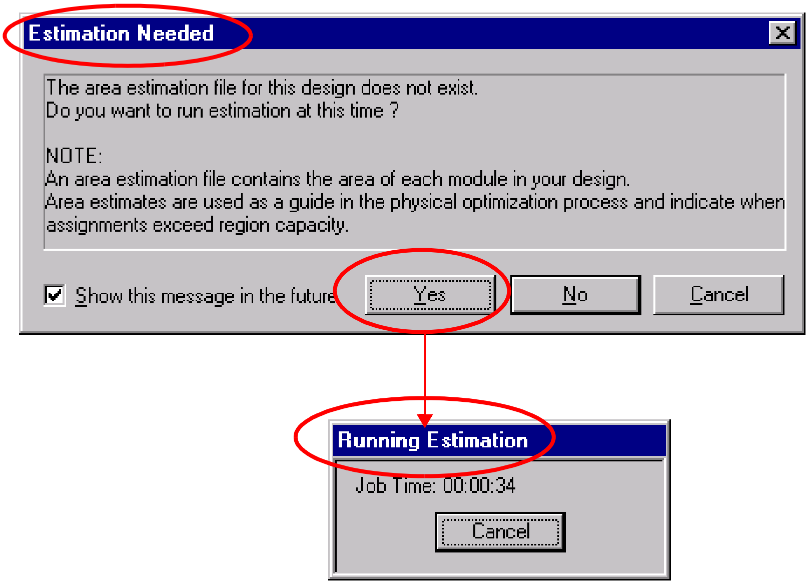
1. Click the New Design Plan icon (▦) in the Project view. Alternatively, you can also create a new design plan file using File->New from the Project menu.

2. 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 now.

   – If you click No, the Design Planner is displayed. See Figure 7-2 on page 7-4.

   – If you click Yes, the Running Estimation dialog pops up in the upper-left corner of the window displaying the runtime of the job. Once estimation is completed, the Design Planner opens. See Figure 7-1 on page 7-3.

Note: The No area estimate warning check box must be selected on the Preferences dialog box to invoke this estimation needed message. Access the Preferences dialog box from the Tools->Design Planner Preferences menu and select the Assignments tab.



The area estimation file for this design does not exist.
Do you want to run estimation at this time ?

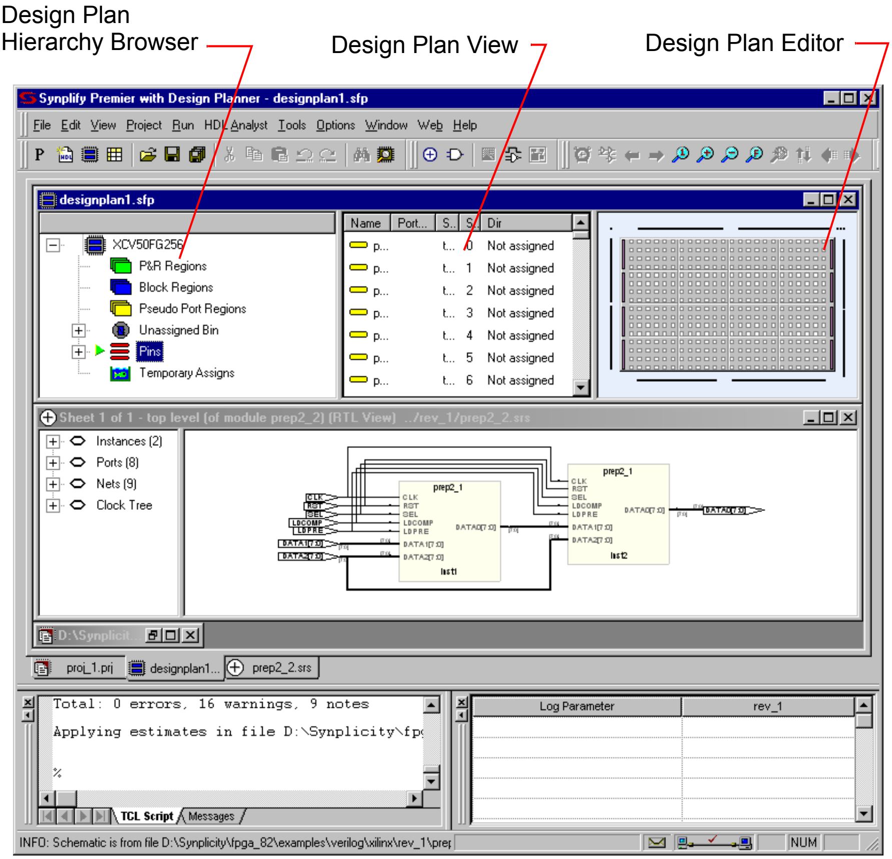
NOTE:
An area estimation file contains the area of each module in your design.
Area estimates are used as a guide in the physical optimization process and indicate when assignments exceed region capacity.

☑ Show this message in the future

Job Time: 00:00:34

Figure 7-1: Estimation Needed and Running Estimation Dialog Boxes

The following figure shows the Design Planner and RTL views.



Name	Port...	S..	S	Dir
p...		t...	0	Not assigned
p...		t...	1	Not assigned
p...		t...	2	Not assigned
p...		t...	3	Not assigned
p...		t...	4	Not assigned
p...		t...	5	Not assigned
p...		t...	6	Not assigned

```
Total: 0 errors, 16 warnings, 9 notes
Applying estimates in file D:\Synplicity\fp(
%
```

INFO: Schematic is from file D:\Synplicity\fpga_82\examples\verilog\xilinx\rev_1\prep

Figure 7-2: Design Planner

The Synplify Premier Design Planner consists of an RTL view and three sub-views:

- Design Plan Hierarchy Browser - displays hierarchical arrangement of the unassigned design modules and pins for a specified device.
- Design Plan View - provides information about module assignment in the device rows and regions.
- Design Plan Editor - shows the physical layout of the device and the placement of the constraints.

See *The Design Planner View* on page 2-25 of the Reference Manual for information on the Design Planner user interface.

Cutting, Copying, and Pasting in the Design Planner

You can use the cut, copy, and paste functions in the Design Plan Editor and Design Plan Hierarchy Browser views. In conjunction with the HDL Analyst copy function, you use these functions instead of drag and drop.

Cut, copy, and paste works only on assignments (such as modules, primitives, and nets). You cannot cut or copy regions using the Design Planner tool and, you cannot paste to multiple regions.

The following subsections provide details on using cut, copy, and paste.

Assign Module or Primitive from Analyst

1. Select the module or primitive in HDL Analyst and copy (Ctrl-c).
2. Select the destination region and paste (Ctrl-v).

Assign a Net to I/O Block from Analyst (Xilinx)

1. Select the net in HDL Analyst and copy.
2. Select the I/O block region and paste.

Assign a Module or Primitive from Hierarchy Browser Unassigned Bin

1. Select the module or primitive in the hierarchy browser and copy.
2. Select the destination region and paste.

Replicate a Module or Primitive using the Hierarchy Browser

1. Select the module or primitive within the region using the hierarchy browser and copy.
2. Select the destination region and paste.

   This displays the Replication dialog box.

Move an assignment Using the Hierarchy Browser View

1. Select the module or primitive within the region using the hierarchy browser and cut (Ctrl-x).

2. Select the destination region and paste.

Pin Assignments

This section discusses the following general guidelines for displaying and assigning pin assignments for design planning:

- Methods for Specifying Pin Assignments on page 7-6
- Specifying Pins Using the Design Plan Editor on page 7-7
- Implementing Pin Assignments on page 7-11
- Storing Temporary Pin Assignments on page 7-14
- Displaying Rats Nesting on page 7-15
- Assigning Clock Pins on page 7-17

Methods for Specifying Pin Assignments

You can specify I/O pin locks and pin assignments using any one of the following features of the Synplify Premier tool:

- Add pins to a constraint (.sdc) file using the SCOPE editor.
- Convert locked I/O pins to a constraint (.sdc) file using Run->Translate Constraints in the Project view. See *Converting Pin Location Constraint Files in the Synplify Premier Tool* on page 3-43 for details.
- Using the Design Plan Editor in Design Planner to assign pins. See *Specifying Pins Using the Design Plan Editor* on page 7-7 for details.
- Importing pin assignments from an Altera .pin or a Xilinx .pad constraint file to a design plan (.sfp) file using Tools->Import Info in the Design Planner. See *Importing Pin Location Files* on page 7-10.

Specifying Pins Using the Design Plan Editor

The Design Plan Editor, the floorplan and region view in the Design Planner, allows you to view and assign external ports and internal nets of the design to I/O pins on the device. The Design Plan Editor is available for Altera and Xilinx devices.

Pin Assignment Options

The Design Plan Editor includes options to:

- Expand the pin view

  To do this, select View->Expanded Pin View menu from the Project menu, or use Ctrl-e to toggle the expanded pin view on (enabled) or off (disabled) in the Design Plan Editor.

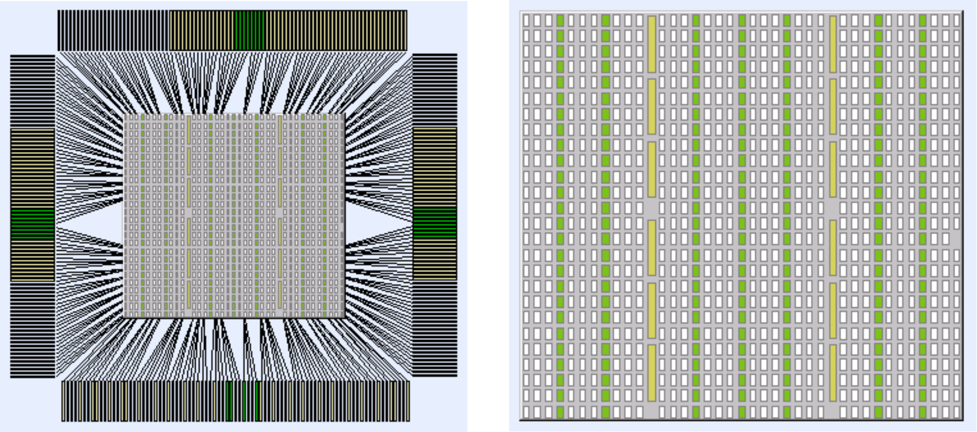


Figure 7-3: Expanded Pin View - Enabled and Disabled (Altera Device)

- Adjust the pin view

  Select View->Adjust Pin View... from the Project menu, the Adjust Pin View dialog box appears.

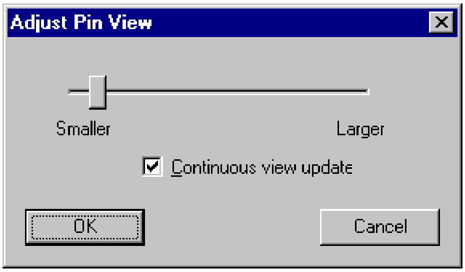


Figure 7-4: Adjust Pin View Dialog Box

To adjust the pin view:

– Move the slider to either a smaller or larger view of the pins.

  By default the Continuous view update option is enabled. This option dynamically displays the pin size changes in the Design Plan Editor as you drag the slider. If disabled, the pin size display updates only after you release the mouse button.

– Click OK to save your new pin view setting or Cancel to restore your original pin view setting.

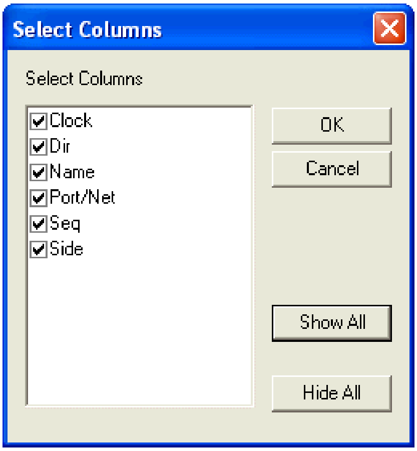
- Display the device I/O pin names

  To display the I/O pin names in the three panes of the Synplify Premier Design Planner, perform the following:

  – In the Design Plan Hierarchy Browser, select the expand icon next to Pins to display all the available I/O pins. When you select a pin in the Design Plan Hierarchy Browser, its corresponding pin location is highlighted in the Design Plan Editor.

  – In the Design Plan Hierarchy Browser, select the Pins folder to list the pins in the Design Plan view. When you select a pin in the Design Plan view, its corresponding pin location is highlighted in the Design Plan Editor.

  – In the Design Plan view, right-click and select Show/Hide columns, then select the columns (Clock, Name, Side, Seq, Dir, or Port/Net) to display from the Select Columns dialog box.

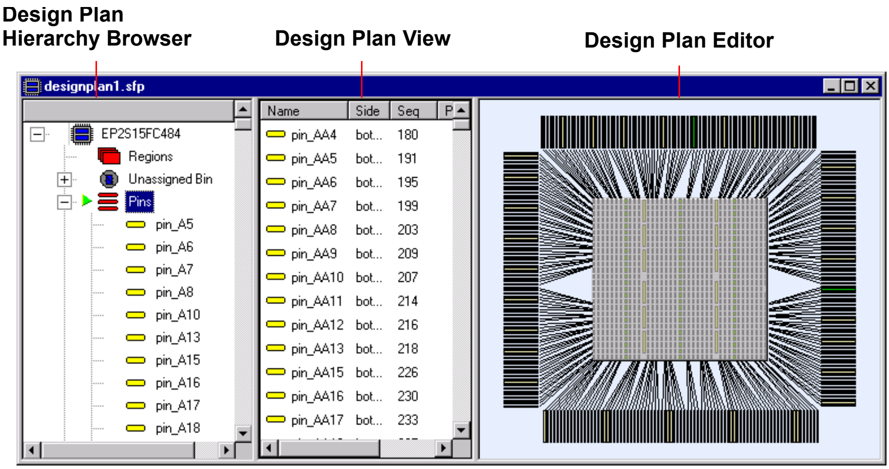


☑ Clock
☑ Dir
☑ Name
☑ Port/Net
☑ Seq
☑ Side

Figure 7-5: Show/Hide Columns for I/O Pins in the Design Plan View

– In the Design Plan Editor, the pin number is displayed when you place the cursor over an I/O pin location.

The following figure shows an example of I/O pins displayed for an Altera device in all three views of the Design Planner.



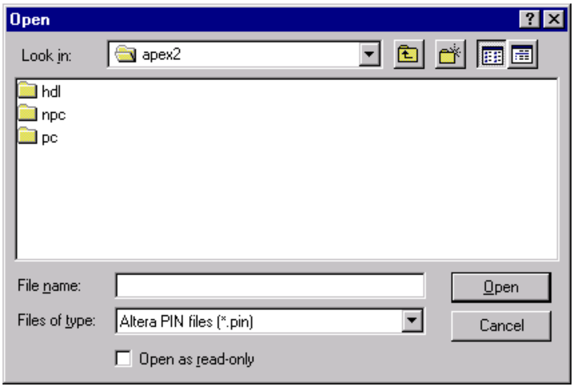
Name	Side	Seq
pin_AA4	bot...	180
pin_AA5	bot...	191
pin_AA6	bot...	195
pin_AA7	bot...	199
pin_AA8	bot...	203
pin_AA9	bot...	209
pin_AA10	bot...	207
pin_AA11	bot...	214
pin_AA12	bot...	216
pin_AA13	bot...	218
pin_AA15	bot...	226
pin_AA16	bot...	230
pin_AA17	bot...	233

Figure 7-6: Design Planner View of I/O Pins (Altera Device)

Importing Pin Location Files

In the Synplify Premier Design Planner, you can use a utility tool to import pin location information from an Altera .pin or Xilinx .pad file into the .sfp file. To use this utility:

1. Select Tools->Import Pin Info from the Project Menu. The Open dialog box appears.



File name:

Files of type: Altera PIN files (*.pin)

□ Open as read-only

Figure 7-7: Import Pin Information Dialog Box

2. Select either an Altera.pin or a Xilinx .pad file and click Open.

   The imported pin assignments are displayed and highlighted in the Design Planner view.

Pin Assignment Conflicts

Since there are several methods of assigning I/O pins, the following are examples of pin assignment conflicts you might encounter:

- Imported information from a .pin or .pad file with different device packages and parts.
- The .sdc file might contain I/O pin locks that conflict with the pin locks specified in the .sfp file.

Note: The SCOPE constraint file (.sdc) typically takes precedence over the Design Plan file (.sfp) when conflicts exist after pin assignments. It is highly recommended you avoid creating these mismatches in the `.sdc` and `.sfp` files.

- If pin locks are specified using the back end place-and-route tool that are added into the `.sfp` file, potential pin lock conflicts may occur. When conflicts exist, an appropriate warning message appears in the `.srr` log file and is displayed in the Messages window.
- Design rule checks are implemented for multiple assignments to the same I/O pins or ports.

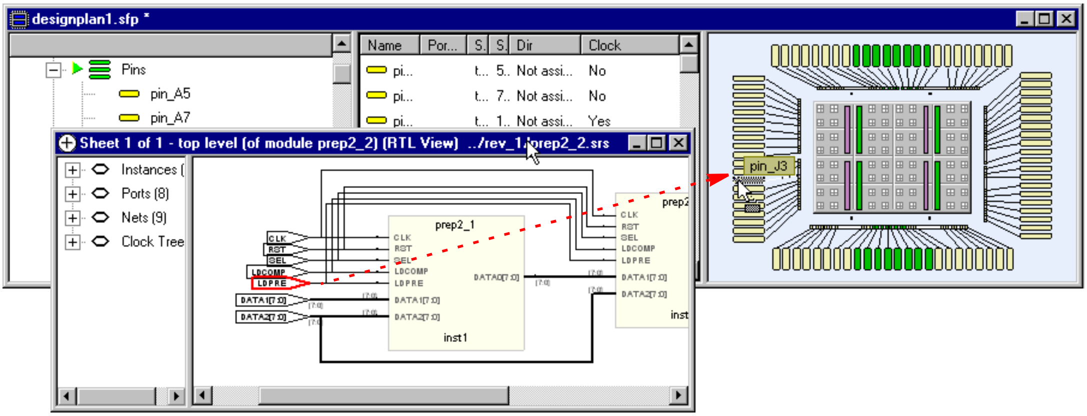
Note: Pin assignment information and region information are included in the same `.sfp` file.

Implementing Pin Assignments

This section describes how to assign I/O pins with the following conditions:

- Assign a single port

  To do this, drag a port from the HDL Analyst RTL view and drop it to the pin location on the device in the Design Plan Editor.



designplan1.sfp *

Name	Por...	S.	S.	Dir	Clock
pi...		t...	5..	Not assi...	No
pi...		t...	7..	Not assi...	No
pi...		t...	1..	Not assi...	Yes

Sheet 1 of 1 - top level (of module prep2_2) (RTL View) ../rev_1/prep2_2.srs

Figure 7-8: Drag a Port to a Pin Location in the Design Plan Editor

Once the pin is assigned, you can reassign its location in the Design Plan Editor by dragging and dropping it to another pin location.

- Assign a bus port (group of signals)

  To do this, drag a bus port from the HDL Analyst 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.

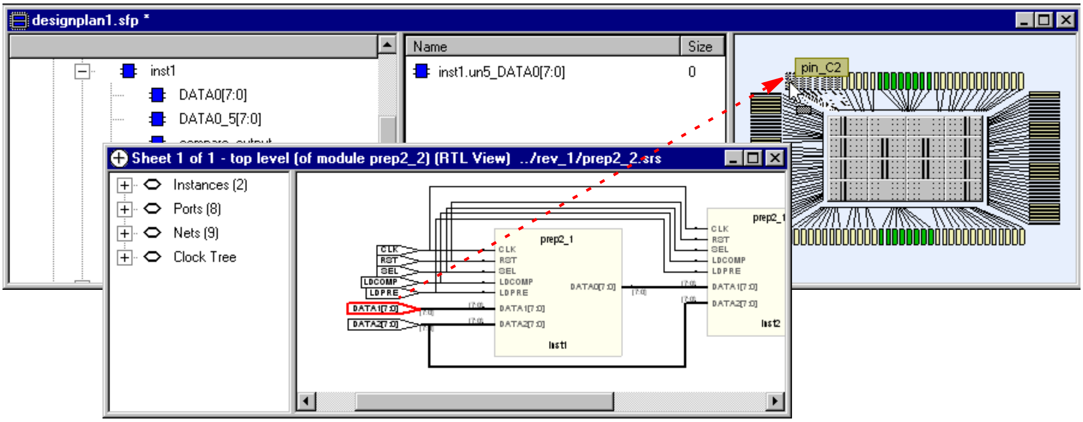
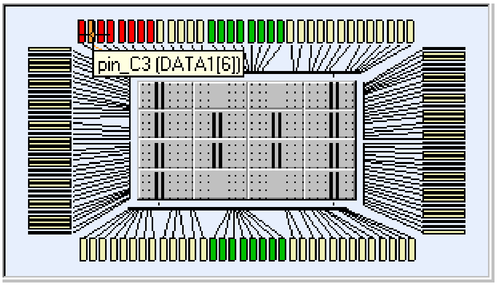


Figure 7-9: Drag a Bus Port to a Pin Location in the Design Plan Editor

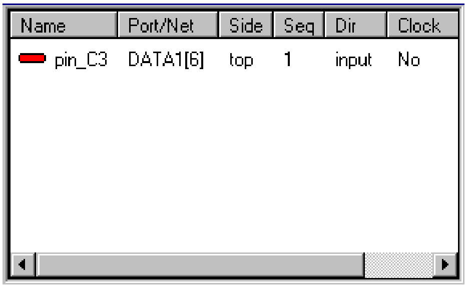
Pins located on the left and right sides of the device are allocated from bottom to top. Pins located on the top and bottom of the device are allocated from left to right. Pins that are occupied will be skipped. All devices allocate pins using this convention.

I/O Pin Display

In the Design Plan Editor, selected assigned pins appear orange, deselected assigned pins appear red, and selected unassigned pins appear blue. When your cursor is above the assigned pin, the pin number along with the port or net assignment is displayed.

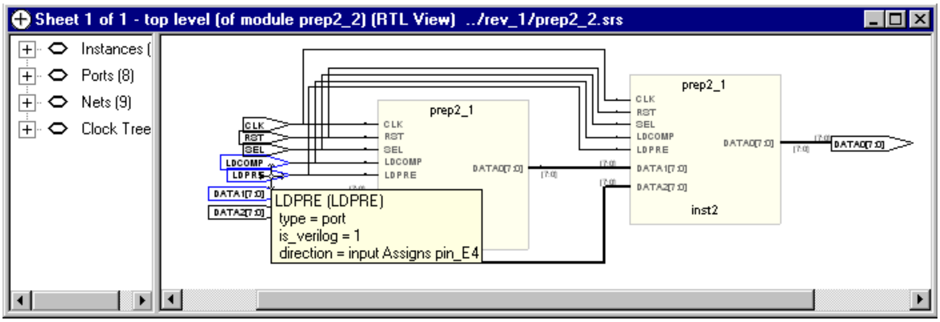


In the Design Plan view, assigned I/O pins display pin directions and the port or net assignments. If these columns are not visible, right-click in the Design Plan view and select Show/Hide columns.



Name	Port/Net	Side	Seq	Dir	Clock
🟥 pin_C3	DATA1[6]	top	1	input	No

In the RTL view, assigned ports appear blue. When your cursor is above the assigned port, a tool tip appears showing the pin assignment information.



Sheet 1 of 1 - top level (of module prep2_2) (RTL View) ../rev_1/prep2_2.srs

LDPRE (LDPRE)
type = port
is_verilog = 1
direction = input Assigns pin_E4

Figure 7-10: Tooltip Displays a Pin Assignment in the RTL View

Note: You can also drag the port from the HDL Analyst->RTL view and drop it to the pin location in either the Hierarchy Browser, or else, the Design Plan view of the Synplify Premier Design Planner.

- Reverse pin assignments

  Reversing pin assignments allows you to assign pins both in a clockwise or counter-clockwise direction.

  To reverse pin assignments, select any set of pins in any view of the Synplify Premier Design Planner and right-click, then select Reverse Pin Assignments from the pop-up menu. The reversed pin assignments are displayed in the Design Planner views.

Storing Temporary Pin Assignments

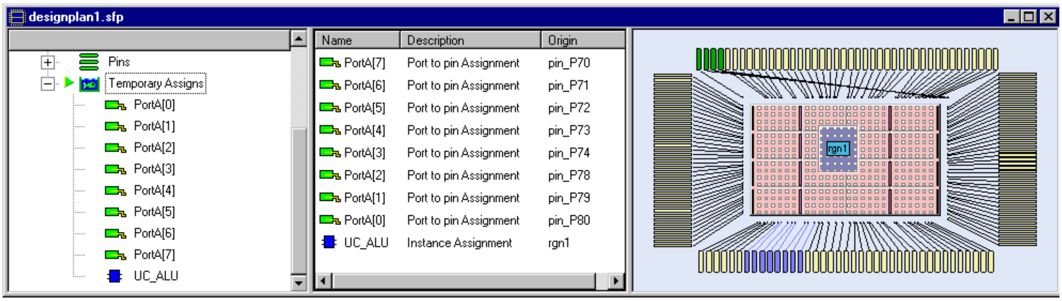
A Temporary Assigns icon ( ) is displayed in the Hierarchy Browser which allows you to temporarily store pin assignments. For example, you can rearrange (reorder) pin assignments for nets or ports. To do this, you must first move these pins to Temporary Assigns and then assign them to the desired pin locations.

The Temporary Assigns option supports the following capabilities:

- You *can* drag and drop pin assignments from the Design Plan Editor to Temporary Assigns. However, you *cannot* drag and drop assignments from the HDL Analyst RTL view to Temporary Assigns.
- You can drag and drop pin assignments from the Temporary Assigns container to the new pin or region location in the Design Plan Editor.
- If you want to return an assignment from the Temporary Assigns container to its original placement:
  - Select the assignment in the Temporary Assigns.
  - Then, right-click and select Reassign from the pop-up menu.
- To empty the Temporary Assigns container:
  - Select the Temporary Assigns icon.
  - Then, right-click and select Empty from the menu.

- To sort pin assignments by description, name, or origin in the Design Plan View, do the following:
  - Display the Description and/or Origin header by right-clicking and selecting Show Columns->Description/Origin from the menu.
  - Click on the Column heading in the Design Plan View to sort.
- You can Edit->Undo or Edit->Redo operations in the Temporary Assigns container.
- If an assignment in the Temporary Assigns container subsequently gets reassigned or unassigned from the RTL view, then the assignment is automatically removed from Temporary Assigns.

The following figure shows the use of Temporary Assigns in the Synplify Premier Design Planner.



Name	Description	Origin
PortA[7]	Port to pin Assignment	pin_P70
PortA[6]	Port to pin Assignment	pin_P71
PortA[5]	Port to pin Assignment	pin_P72
PortA[4]	Port to pin Assignment	pin_P73
PortA[3]	Port to pin Assignment	pin_P74
PortA[2]	Port to pin Assignment	pin_P78
PortA[1]	Port to pin Assignment	pin_P79
PortA[0]	Port to pin Assignment	pin_P80
UC_ALU	Instance Assignment	rgn1

Figure 7-11: Design Planner with Temporary Assigns

Displaying Rats Nesting

After I/O pins are assigned, you can enable rats nesting to show connectivity between the I/O pads and the assigned logic for regions on the device.

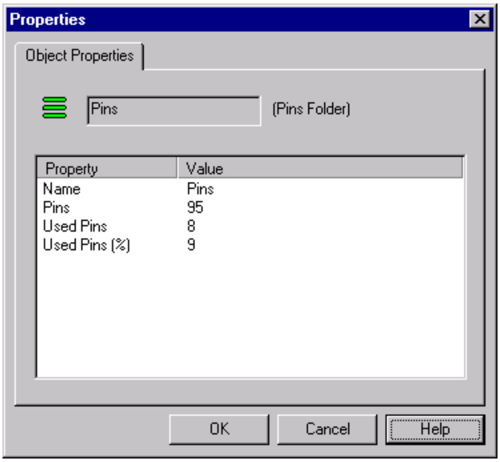
- To enable rats nesting for all regions of a design, right-click in the Design Plan Editor and select Rats Nest->Show from the pop-up menu.
- To enable rats nesting for a region, select a region and right-click in the Design Plan Editor. Select Rats Nest->Show Selected from the menu
- To disable rats nesting, right-click in the Design Plan Editor and select Rats Nest->Hide.

You can also select View->Rats Nest from the Project menu, then choose the Show, Hide, or Show Selected command.

Pin Assignment Statistics

After assigning I/O pins, you can view your assignments with the following tools:

- In the Design Plan Hierarchy Browser, right click on the Pins folder 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.



Pins (Pins Folder)

Property	Value
Name	Pins
Pins	95
Used Pins	8
Used Pins (%)	9

- When you select assigned port(s) in the HDL Analyst RTL view, their corresponding pin(s) are highlighted in the Synplify Premier Design Planner views. Likewise, you can also select assigned pin(s) from any Design Planner view and their corresponding port(s) will be highlighted in the HDL Analyst RTL view.

Note: You can also crossprobe ports from the HDL Analyst RTL view to their pin assignments in the Hierarchy Browser and Design Plan views.

- When you select an internal net with a pin assignment from the HDL Analyst RTL view, its corresponding pin will be highlighted in the

Synplify Premier Design Planner views. Likewise, you can also select this assigned pin from any Design Planner view, and its corresponding internal net will be highlighted in the HDL Analyst RTL view.

Assigning Clock Pins

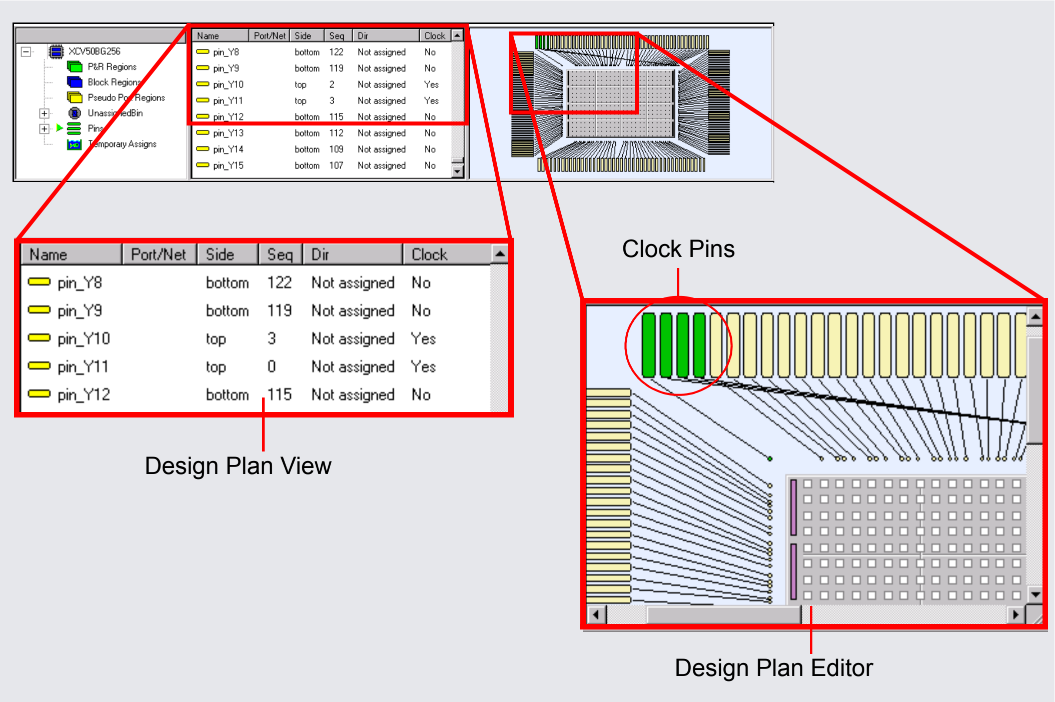
The Design Plan Editor allows you to view and assign clock pins on the device.

Clock pins support the following capabilities:

- Clock pins are available for Altera and Xilinx devices.
- Clock pins are displayed in the color green to differentiate them from signal I/O pins in the Design Plan Editor.
- You can drag and drop a signal pin to a clock pin. When you do so, a message asks you to confirm the assignment to ensure that the correct signal gets assigned to the clock pin.
- You cannot drag and drop a bus (group of signals) to a clock pin.
- You can drag and drop a bus to an I/O pin near a clock pin(s). The clock pin(s) will be skipped when assigning this bus to the I/O pins.

Refer to *Specifying Pins Using the Design Plan Editor* on page 7-7 for information about how to use the Synplify Premier Design Planner pin assignment capabilities. In the Design Planner, you can similarly assign clock pins as you do I/O pins. The color of the pin changes after you assign a clock pin.

In the Design Plan view, you can enable or disable the Clock column on the Select Columns dialog box which displays whether or not a pin is a clock (Yes or No). See the following figure.



Name	Port/Net	Side	Seq	Dir	Clock
pin_Y8		bottom	122	Not assigned	No
pin_Y9		bottom	119	Not assigned	No
pin_Y10		top	2	Not assigned	Yes
pin_Y11		top	3	Not assigned	Yes
pin_Y12		bottom	115	Not assigned	No
pin_Y13		bottom	112	Not assigned	No
pin_Y14		bottom	109	Not assigned	No
pin_Y15		bottom	107	Not assigned	No

Name	Port/Net	Side	Seq	Dir	Clock
pin_Y8		bottom	122	Not assigned	No
pin_Y9		bottom	119	Not assigned	No
pin_Y10		top	3	Not assigned	Yes
pin_Y11		top	0	Not assigned	Yes
pin_Y12		bottom	115	Not assigned	No

Clock Pins

Design Plan View

Design Plan Editor

Figure 7-12: Design Plan Editor Showing Clock Pins (Xilinx Device)

Working with Regions

This section discusses the following general guidelines for placing and editing regions in the Design Planner before running physical synthesis:

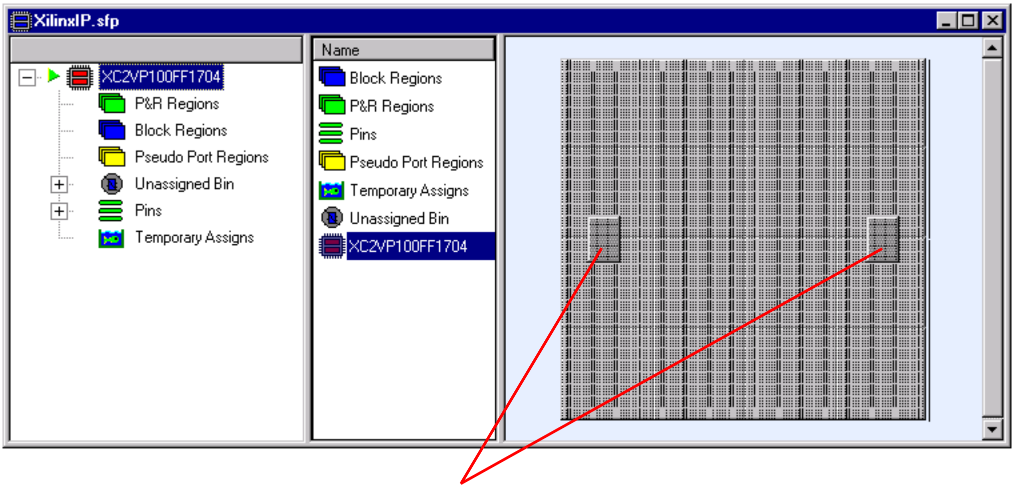
- Viewing Intellectual Property (IP) Core Areas on page 7-19
- Placing Regions on page 7-20
- Moving and Sizing Regions on page 7-21
- Replicating Logic Manually on page 7-23
- Assigning Register to Pin-Locked I/O Paths to Regions on page 7-24
- Checking Synplify Premier Utilization on page 7-25

Viewing Intellectual Property (IP) Core Areas

You can view dedicated areas on the device reserved for IP core from the design plan editor UI of the Design Planner. The IP core area appears as a gray box on the device in the Design Plan Editor. Xilinx Virtex-II Pro devices can contain up to four IP core areas depending upon the part specified for the device.

IP core areas support the following capabilities:

- IP core areas are available on Xilinx Virtex-II Pro (Virtex2p - Implementation options name) devices.
- A tool tip is displayed when the cursor is placed over an IP core area in the Design Plan Editor.
- Do *not* create regions that overlap or are contained within an IP core area.



IP Core

For more information on IPs, see *Handling Xilinx IPs (Design Planner)* on page 9-35.

Placing Regions

Where you place a region is dependent on how the data flows in your design and where the pins are locked.

For Altera, you can determine where the target place-and-route tool places the critical path after you run with no constraints. Determine where the critical path is placed using the Design Plan Editor. This can be a good starting place for you to determine what row to begin with when placing the critical path on the logic device using the Synplify Premier Design Planner tool.

For Xilinx, the size and location of a region can be easily modified, so a rough estimate of the region is usually sufficient. However, a starting point for determining where to place the region is to gather information from the Xilinx floorplanner. Run placement and routing without constraints, then use the floorplanner to determine where the critical path logic is placed. From this information, you can begin by creating the region in the same general area on the logic device using the Design Planner tool.

For Altera, you can also overlap regions. When overlapping regions, be aware that the Synplify Premier Design Planner software treats overlapping regions no differently than regions that do not overlap.

For Xilinx, it is *not* recommended that you overlap regions for some designs. The Xilinx place-and-route tool cannot always place these designs, and therefore, can potentially create an error. But keep in mind, Synplify Premier Design Planner software can still support regions that overlap.

Moving and Sizing Regions

The following enhancements provide user support when you create, move, or resize regions.

- WYSIWYG region boundaries
- Cursor arrow keys region manipulation
- Preserving logic and memory resources

WYSIWYG Region Boundaries

The Design Planner tool shows you region boundaries when a create, move, or resize region operation is performed. Region boundaries are adjusted to fit around logic and memory contained in the region.

Moving Regions

You can move regions using either the cursor arrow keys or the mouse button.

Using the Cursor Arrow Keys

To move a region using the cursor arrow keys:

1. Select the region to highlight the rectangle representing this region.
2. Use the cursor arrow keys (left, right, up, or down) to reposition the region on the device.

Using the Mouse Button

To move a region using the mouse button:

1. Select the region to highlight the rectangle representing this region.
2. Press the left mouse button while dragging the region to the desired position on the device.

Note: Logic and memory resources are preserved when you move a region, using either the mouse button or the cursor arrow keys.

Resizing Regions

You can resize regions using either the cursor arrow keys or the mouse button. To resize a region:

Using the Cursor Arrow Keys

To resize a region using the cursor arrow keys:

1. Select the region to highlight the rectangle representing this region.
2. Press and hold the Ctrl and Shift keys simultaneously.
3. Press the cursor arrow key (left, right, up, or down) in the direction you want the region resized.

   An initial resizing arrow appears along the edge of the region.
4. While continuing to hold the Ctrl and Shift keys, use the cursor key to resize the region in the direction of the arrows.
5. Release the Shift key. You can no longer resize the region.

Using the Mouse Button

To resize a region using the mouse button:

1. Select the region to highlight the rectangle representing this region.
2. 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.

Note: Logic and memory resources are *not* preserved when you resize a region, using either the cursor arrow keys or the mouse button.

Preserving Logic and Memory Resources

When regions contain both logic and memory resources, you can preserve these resources when you move a region. For example, Xilinx devices can preserve the number of CLBs and BRAMs in a region. Altera devices can preserve the number of LABs and ESBs in a region.

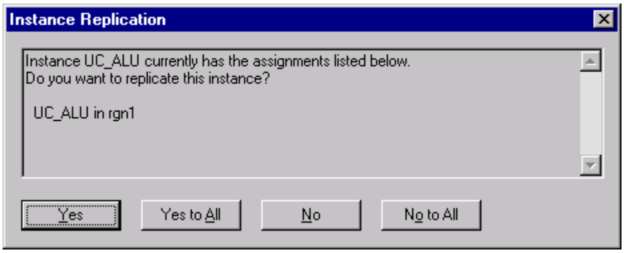
To *enable* resource preservation, hold the Shift key while you move a region using either the mouse button or cursor arrow keys. When you resize a region using either the mouse button or the cursor keys, you *cannot* preserve resources. The default is not to preserve resources.

Replicating Logic Manually

You need to replicate logic manually when an instance in a region fans out to instances in several other regions. This is done to avoid the routing delay between the regions, especially if the regions are not placed close together. After replication, each region contains a local copy of the instance. You can copy (`Ctrl-c`) logic to be replicated and paste (`Ctrl-v`) replicated logic in the Design Planner.

Note: If you replicate an instance to a region and the instance does not drive any logic in the region, then the software does not create a copy of the instance in the region. Therefore, when you look at the RTL netlist of the region, the replica of the instance does not appear.

When you assign the same instance logic from the HDL Analyst RTL view to different regions created in the Design Plan Editor, an Instance Replication dialog box appears as shown in the following figure. Confirm whether or not you want to replicate the selected logic instance to the specified region. You can choose to confirm replication for each instance separately (Yes or No) or simultaneously (Yes to All or No to All) for multiple instances.



Instance UC_ALU currently has the assignments listed below.
Do you want to replicate this instance?

UC_ALU in rgn1

Figure 7-13: Instance Replication Dialog Box

Assigning Register to Pin-Locked I/O Paths to Regions

If a path is critical from a register to a pin-locked I/O, make sure that you assign the critical path to a region that is close to the pins where the I/O is locked. Physically constraining logic close the to locked pins minimizes routing delays.

Checking Synplify Premier Utilization

As a general rule, follow the utilization guidelines defined in the following sections for device and region utilization.

Device Utilization

Device utilization above 90% can lead to longer timing closure. If your design uses over 90% of the device and if the design contains several finite state machines, try using the sequential encoding style, (instead of one-hot) to possibly provide more space on the device.

Region Utilization

Area utilization of a region should not be more than 80% to allow for Synplify Premier Design Planner area estimations and for any additional room 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 the routing process, the place-and-route tool will error out. Make sure the utilization estimates are up-to-date (using Run->Estimate Area in the Project view and right-click->Estimate Regions after making any changes to the design plan) and that utilization does not exceed 80% before going on to the placement and routing phase. After synthesis, you can also check the utilization in the `.srr` file to get a better estimate for the design.

Using Process-Level Hierarchy

The presence of process-level hierarchy can affect design performance either 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 is intended to treat designs with and without process-level hierarchy the same way. However, the presence of process-level hierarchy changes names of instances extensively and this affects the mappers as well as the back end place-and-route tools.

Bit Slicing

The following topics describe bit slicing in the Synplify Premier tool.

- About Bit Slicing on page 7-26
- Using Bit Slicing on page 7-26
- Bit Slice Examples on page 7-29
- Bit Slicing Guidelines on page 7-32

About Bit Slicing

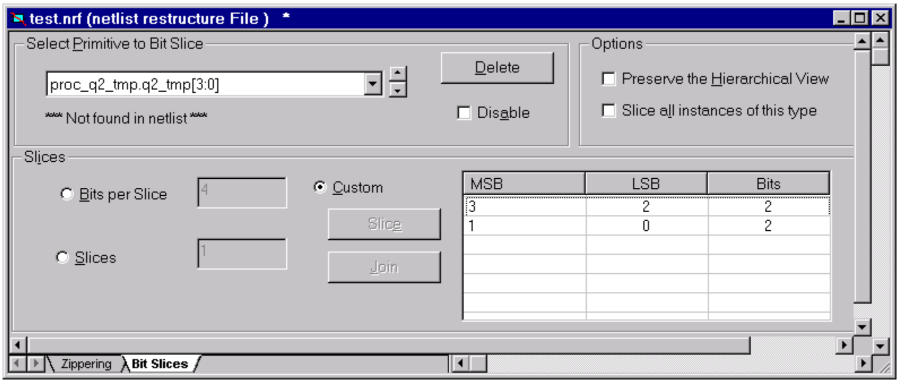
You can use bit slicing whenever a primitive is too large to fit into a region or when a finer granularity of placement control is needed for a wide signal. Bit slicing allows you to break up larger primitives into a number of smaller primitives where each can be placed into separate regions, as appropriate.

The logical division of primitive outputs is defined by a `slice_primitive` command read from a netlist restructure (`.nrf`) file. Bit slicing references this file and uses the netlist filter to control the logical division of the element into the user-defined number of primitives. A graphical user interface simplifies the editing of the `.nrf` files for bit slicing (existing `.tcl` files can be used directly by the netlist filter or renamed with a `.nrf` extension for viewing in the graphical user interface).

Using Bit Slicing

To use the Project environment for bit slicing:

1. In the Synplify Premier project window, open a new (File->New->Netlist Restructure File) or existing `.nrf` file, then click on the Bit Slices tab.



test.nrf (netlist restructure File) *

Select Primitive to Bit Slice: proc_q2_tmp.q2_tmp[3:0]

*** Not found in netlist ***

☐ Disable

Options

☐ Preserve the Hierarchical View

☐ Slice all instances of this type

Slices

○ Bits per Slice 4

○ Slices 1

● Custom

MSB	LSB	Bits
3	2	2
1	0	2

Figure 7-14: Bit Slicing

2. Type in or drag and drop the instance to slice from the RTL view onto this tab.

To slice an instance by a specified number of bits per slice or by a specified number of slices:

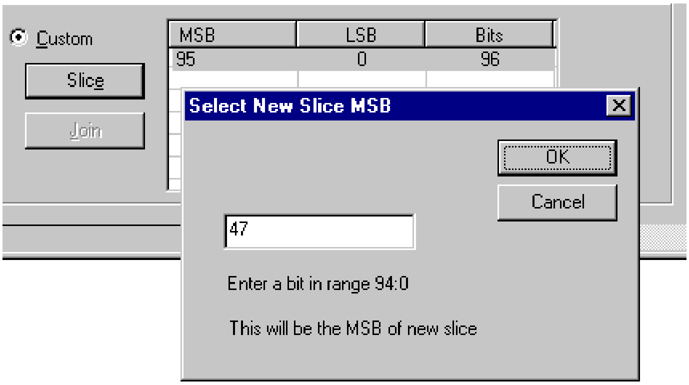
1. Enter a value for Bits per Slice or Slices clicking the corresponding button and entering a value in the adjacent field. If you enter a Bits per Slice value, an instance is allocated for each group of bits with any remaining bits allocated to the last instance. When using a Slices value, the bits are divided equally among the specified number of instances with the last instance assigned any partial number.
2. Save the file.
3. Close the RTL view and redisplay the Project view. The netlist restructure folder displays in the Project view.
4. Display the Options for Implementation dialog box (click Impl Options or select Project->Implementation Options) and click on 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.
5. 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.

Custom Slicing

A custom slice setting is available for defining slices of varying widths. To use the custom setting:

1. Click on the Custom button. This enables the MSB/LSB table.
2. Select the entry in the table, then click on the Slice button.

   This displays the Select New Slice MSB.



● Custom

MSB	LSB	Bits
95	0	96

47

Enter a bit in range 94:0

This will be the MSB of new slice

Figure 7-15: Select New Slice MSB

3. Either click OK to slice the number of bits into two or enter the starting MSB for the second (least significant) slice.
4. Continue to select entries in the table and click Slice to redisplay the Select New Slice MSB popup menu (see *Slicing into Predefined Primitives* on page 7-30).
5. Save the file.
6. Close the RTL view and redisplay the Project view.
7. Display the Options for Implementation dialog box (click Impl Options or select Project->Implementation Options) and select the Netlist Restructure tab. Make sure that the netlist restructure file is checked.
8. Recompile the design. The sections of the sliced element can be individually assigned.

Bit Slice Options

To enable either of the Bit Slice options, check the corresponding box in the Options section on the Bit Slices tab.

- Preserve the Hierarchical View–preserves the hierarchical view (the effects of bit slicing are only visible at the next level down in the hierarchy).
- Slice all instances of this type–global application of the bit slice definition to all same-type instances in the netlist

Note: If you use zippering on a module before bit slicing a primitive within the module, the post zippering name of the module instance must be used in the bit slicing command. For example, after using zippering on a module, run Compile Only (F7) and 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.

- Select a group of bits, right-click and select Properties. A dialog box displays the bit slicing properties of the primitive. Click OK to dismiss this dialog box. Also, see *Zippering Guidelines* on page 7-41.

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.

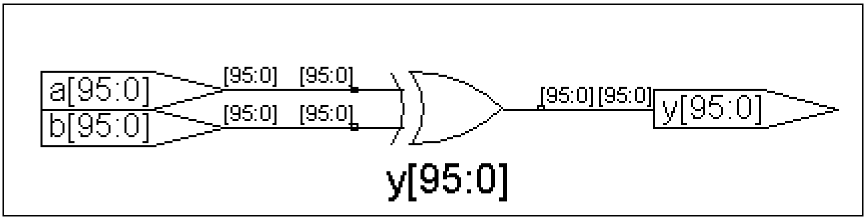
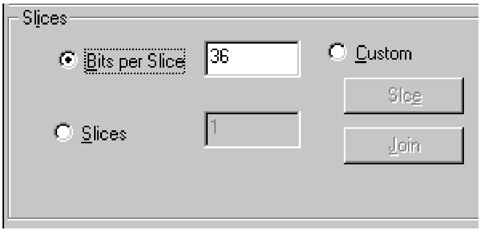


Figure 7-16: Unsliced 96-bit primitive

Slicing into Primitives of Equal Size

In this example, the Bits per Slice value is set to 36.



Slices

● Bits per Slice 36

○ Slices 1

○ Custom

Figure 7-17: Setting the Bits per Slice

The Bits per Slice setting divides the output of the y[95:0] primitive into three individual primitives. The first two primitives each contain the requested 36 bits; the last primitive contains the remaining 24 bits (y[95:72]). The RTL Device view for this bit slicing example is shown in the following figure.

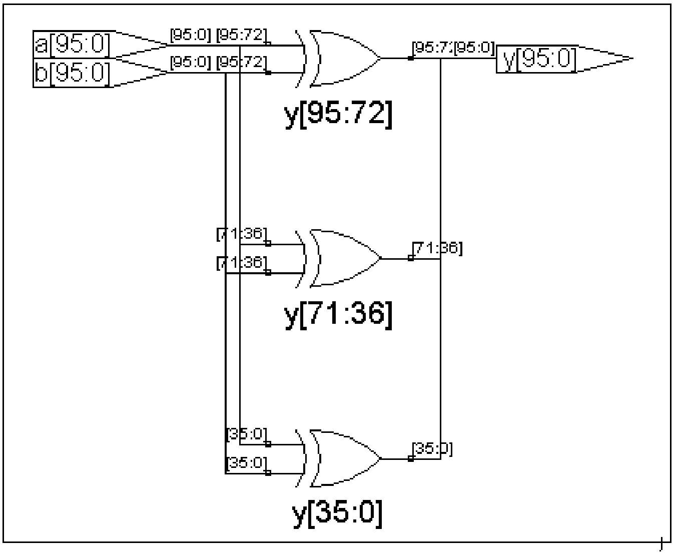
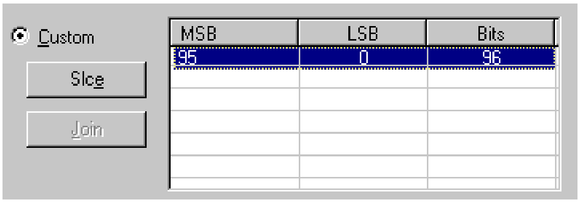


Figure 7-18: Bit Slicing Into Primitives of Equal Widths

Slicing into Predefined Primitives

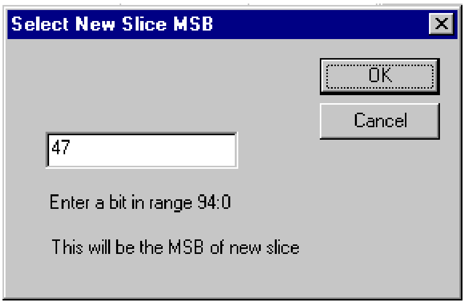
In this example, the Custom setting is used to slice the primitive into three individual primitives of predefined widths of 48, 32, and 16. When the Custom radio button is selected, the adjacent table is enabled. Clicking on the top (and only) entry enables the Slice button as shown in the following figure.



MSB	LSB	Bits
95	0	96

Figure 7-19: Setting the Bits per Slice

Clicking the Slice button prompts you to accept the displayed MSB for the new (next) slice or to enter another MSB for the slice.



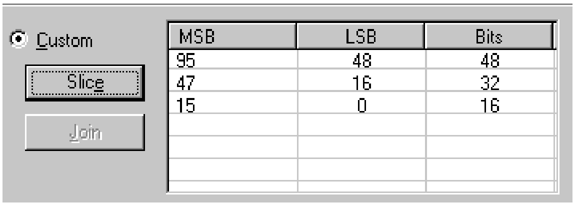
47

Enter a bit in range 94:0

This will be the MSB of new slice

Figure 7-20: Creating the First Slice

For this example, click OK to create an initial bit slice of 48. 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. Select the second entry in the table and click Slice. You are again prompted to accept the displayed MSB for the new slice or to enter another MSB for the slice. Enter 15 (the MSB for the third slice) and click OK. The table is updated as shown in the following figure.



MSB	LSB	Bits
95	48	48
47	16	32
15	0	16

Figure 7-21: Final Table Values

You can merge two (or more) adjacent slice definitions in the table by selecting the entries with the Ctrl key and clicking Join. Using this feature allows you to essentially undo an entry with an incorrect width.

Save the `.nrf` file, make sure that the filename is checked on the Netlist Restructure tab, and run Compile Only (press F7) on the design. The RTL Device view for this bit slicing example is shown in the following figure.

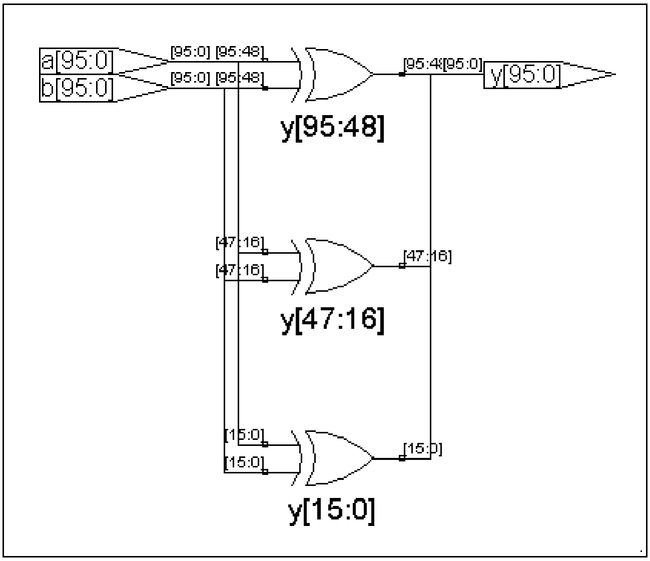


Figure 7-22: Bit Slicing into Primitives of Varying Widths

Bit Slicing Guidelines

For bit slicing, you can only divide bus primitives of the following types:

buf

inv

tristate

or

xor

and

register

mux

latch

See `slice_primitive` in the *Tcl Commands and Scripts* chapter of the *Synplicity FPGA Reference Manual* for information on using the bit-slicing command in a `.nrf` file or in a `.tcl` script.

Zippering

You can use zippering whenever a logic block is too large to fit into a region. Zippering allows you to break up a block into a number of smaller instances where each can be placed into separate regions, as appropriate.

This help section consists of the following Zippering topics:

- Using Zippering
- Analyzing a Design for Zippering
- Zippering Example
- Zippering Guidelines

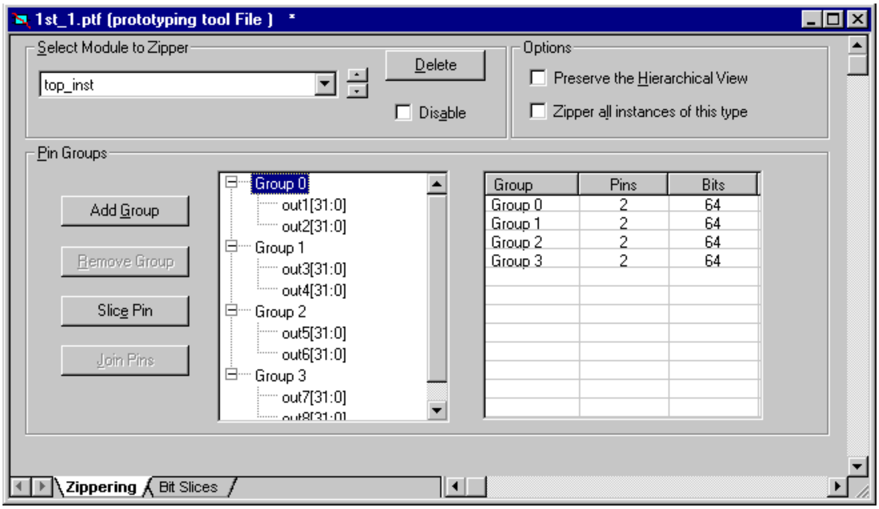
Zippering works by allowing the outputs of a block to be divided into groups. Once divided, a “cone-of-logic” is traced down through the hierarchy to the input pins to create instances containing only the requisite logic. While calculating the cone of logic, logic replication occurs based on the number of inputs in the cone. The individual instances can then be assigned to different regions.

Using Zippering

Use the `zipper_inst_hier` command in the `.nrf` or `.tcl` file to define where to logically divide a module by identifying groups of output signals. Zippering references the file and uses the netlist filter to control the logical division of the module. A graphical user interface simplifies the creation and editing of `.nrf` files (existing `.tcl` files can be used directly or renamed with a `.nrf` extension for viewing in the graphical user interface).

To use the Project environment for zippering:

1. In the Synplify Premier project view, create a new file (File->New->Netlist Restructure File) or existing `.nrf` file, then click on the Zippering tab.



Select Module to Zipper: top_inst

□ Disable

Options

□ Preserve the Hierarchical View

□ Zipper all instances of this type

Pin Groups

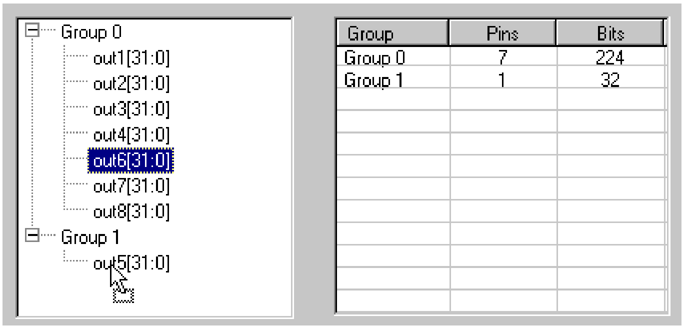
- Group 0
  - out1[31:0]
  - out2[31:0]
- Group 1
  - out3[31:0]
  - out4[31:0]
- Group 2
  - out5[31:0]
  - out6[31:0]
- Group 3
  - out7[31:0]
  - out8[31:0]

Group	Pins	Bits
Group 0	2	64
Group 1	2	64
Group 2	2	64
Group 3	2	64

Figure 7-23: Zippering

2. Drag and drop the block to be zippered from the RTL view to the UI.
3. Click on the “+” sign to expand the Group 0.

   This displays the output nets.
4. Click on Add Group to add an empty group to the Pin Groups window. If necessary, continue to click on Add Group to add additional groups (each group defined represents a zippered section).
5. Click on a net in group 0 and drag the net to the new group. You can use the Ctrl and Shift keys to select more that one net.



- Group 0
  - out1[31:0]
  - out2[31:0]
  - out3[31:0]
  - out4[31:0]
  - out6[31:0]
  - out7[31:0]
  - out8[31:0]
- Group 1
  - out5[31:0]

Group	Pins	Bits
Group 0	7	224
Group 1	1	32

Figure 7-24: Assigning Net to Group

6. Continue to arrange the groups by dragging nets to the individual groups. Note that you can slice a net (see *Slicing a Bus Net* on page 7-35) by selecting the net and clicking Slice Pin.

7. When all of the nets are arranged in groups, save the file.

8. Close the RTL view and redisplay the Project view.

9. Display the Options for Implementation dialog box (click Impl Options or select Project->Implementation Options) and select the Netlist Restructure tab. Make sure that the netlist restructure file is checked.

10. Run Compile Only (F7) on your design with the netlist restructure file and open the partition. The sections of the zippered element can be individually assigned.

11. The additional logic replication performed during zippering increases the total area of the design. To update area estimates, run area estimation (press F9) after zippering.

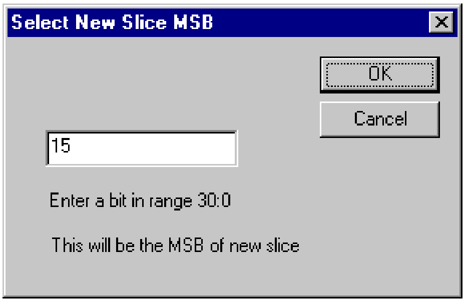
    Take care when using zippering since non-optimal zippering can cause extensive replication which can significantly increase design size.

Slicing a Bus Net

When necessary, bus nets can be individually sliced and divided into separate groups. To slice a bus net:

1. Select the bus net to slice from the group in the Pin Groups window.

2. Click Slice Pin. Accept the displayed MSB for the new (next) slice or enter another MSB for the slice.



Select New Slice MSB

15

Enter a bit in range 30:0

This will be the MSB of new slice

Figure 7-25: Creating the First Slice

3. Click OK to create a slice equal to half the width of the initial bus net or enter an MSB for the new (second) slice and click OK. 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 essentially undo the split by selecting the nets using the Ctrl or Shift key and clicking Join Pins.

Analyzing a Design for Zippering

Zippering requires careful examination of your design to logically divide the block into an optimal number of instances and logical signal boundaries. As a simple example, consider the large block shown in Figure 7-26 on page 7-36.

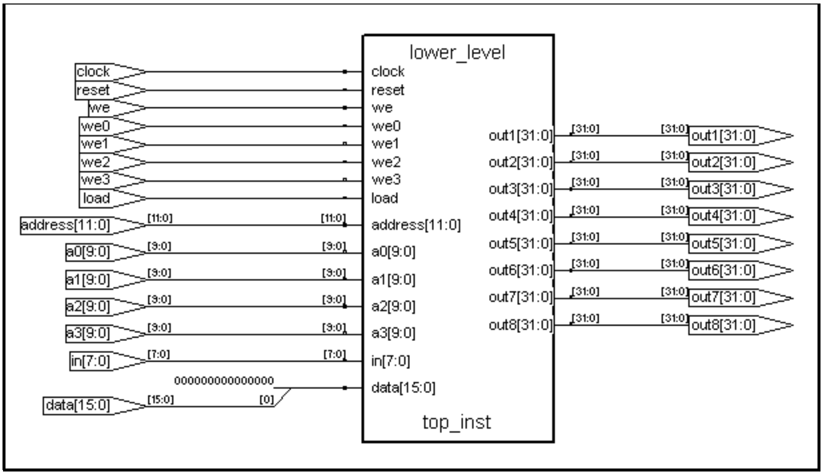


Figure 7-26: RTL View of top_inst Hierarchical Block

The hierarchical block in Figure 7-26 requires 325 I/O pins including 256 output pins for the eight output buses and 69 input pins. In a design, the number of I/O pins required for this block may be too large to fit into one region. Looking down a level into the block's hierarchy (see Figure 7-27 on page 7-37), if you were to logically divide the block into two instances as shown by the broken line, each instance would require a smaller number of output pins for the split buses and some number of input pins with some redundancy (common logic). These two instances could then be assigned to different regions.

With zippering, you only need to specify the outputs and corresponding instance. The “cone of logic” is used to trace outputs back to their inputs and only logic necessary to control the specified outputs is included.

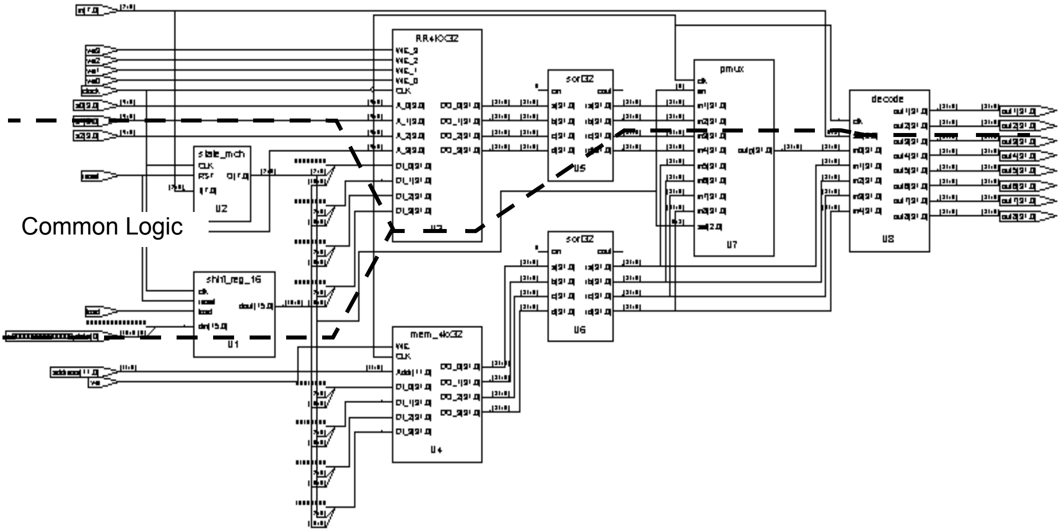


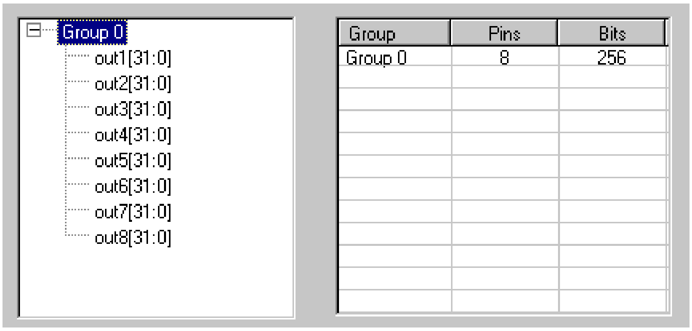
Figure 7-27: RTL View: top_inst One Level Down in Hierarchy

Zippering Example

The following example illustrates zippering a 256-output block shown in Figure 7-27 on page 7-37 into two instances. In this example, the outputs from module top_inst are divided between two instances: out1, out2, out3, and out4 in one instance and out5, out6, out7, and out8 in the other instance.

To zipper the example design:

1. Open a new or existing `.nrf`.
2. Drag and drop top_inst from the RTL view to the UI.
3. Click on the “+” sign to Expand Group 0 in the Pin Groups window.

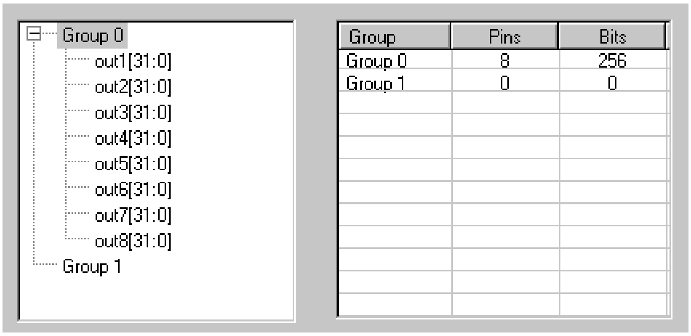


Group 0
- out1[31:0]
- out2[31:0]
- out3[31:0]
- out4[31:0]
- out5[31:0]
- out6[31:0]
- out7[31:0]
- out8[31:0]

Group	Pins	Bits
Group 0	8	256

Figure 7-28: Expand Group 0

4. Click Add Group to add an empty group to the Pin Groups window as shown in the following figure.



Group 0
- out1[31:0]
- out2[31:0]
- out3[31:0]
- out4[31:0]
- out5[31:0]
- out6[31:0]
- out7[31:0]
- out8[31:0]

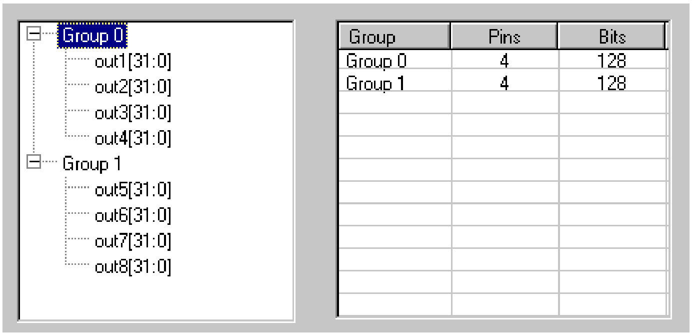
Group 1

Group	Pins	Bits
Group 0	8	256
Group 1	0	0

Figure 7-29: Add a Second Group

5. Select net out5[31:0] in Group 0 and drag it to Group 1. Expand Group 1 (click the + sign) to show the new assignment.

6. In order, drag and drop nets out6[31:0], out7[31:0], and out8[31:0] from Group 0 to Group 1. The groups are shown in the following figure.



- Group 0
  - out1[31:0]
  - out2[31:0]
  - out3[31:0]
  - out4[31:0]
- Group 1
  - out5[31:0]
  - out6[31:0]
  - out7[31:0]
  - out8[31:0]

Group	Pins	Bits
Group 0	4	128
Group 1	4	128

Figure 7-30: Defining the Groups

7. Save the file.

8. Close the RTL view and redisplay the Project view.

9. Display the Options for Implementation dialog box (click Impl Options or select Project->Implementation Options) and select the Netlist Restructure tab. Make sure that the netlist restructure file is checked.

10. Run Compile Only (F7) on the design.

The results are shown in the following RTL view.

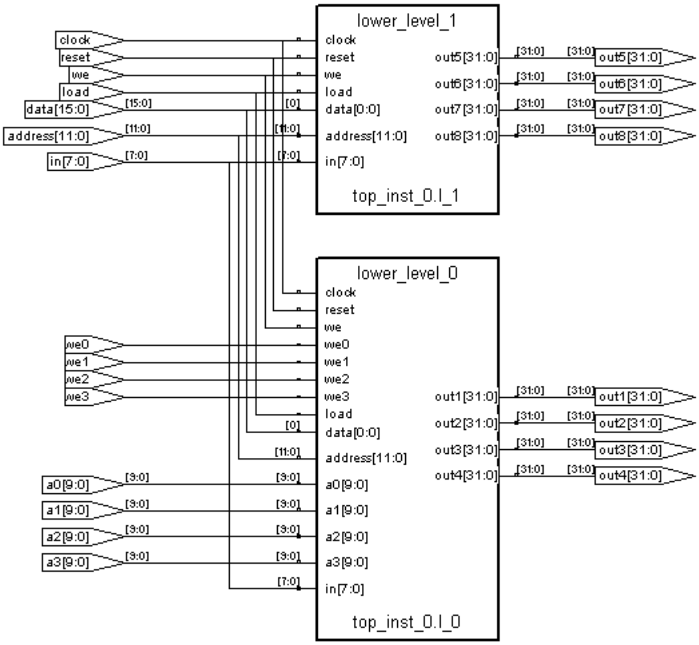


Figure 7-31: Zippered Module

As shown in Figure 7-31, the original block is split into two instances. The first instance is named top_inst_0.1_1 and contains the out5 through out8 buses, and the second instance is named top_inst_0.1_0 and contains the out1 through out4 buses. Looking at the I/O pin requirements, the first instance requires 153 I/O pins, and the second instance requires 197 I/O pins.

Zippering Guidelines

For zippering:

- You can combine zippering and bit slicing in a single `.nrf`. Bit slicing commands are automatically placed ahead of the zippering commands in the file so that as the file is read, line-by-line, all of the primitives are sliced before any outputs are zippered.
- If you zipper a block before bit slicing a primitive in the block, the post zippering name of the instance must be used in the bit slicing command. For example, after zippering a block, run Compile Only (F7), open the RTL view, and push down into the new hierarchical block containing the primitive. Drag and drop the primitive to bit slice onto the Bit Slices tab of the UI.
- Zippering usually increases overall area utilization which can increase dramatically with the random selection of output groups.
- Zippering can be done at any level in the hierarchy above the leaf level; the full hierarchical instance name must be specified.
- After zippering, individual instances may not include all of the contents of the original instance.
- Zippering through an existing `.tcl` file is supported; add the `.tcl` file with the Add File button or select the Netlist Restructure tab of the Option for Implementation dialog box and click Add Restructure File.
- 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).
- Select a group of instance pins, right-click and select Properties. A dialog box displays the zippering pin group properties of the module. Click OK to dismiss this dialog box. Also, see *Bit Slice Options* on page 7-29.

See `zipper_inst_hier` in the *Tcl Commands and Scripts* chapter of the *Synplicity FPGA Reference Manual* for information on using the zippering command in a `.nrf` or in a `.tcl` script.

CHAPTER 8

Vendor-Specific Optimizations

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 6, *Design Optimization*.

This chapter describes the following:

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 8-2
- Specifying Locations for Actel Bus Ports, on page 8-3
- Specifying Macro and Register Placement, on page 8-3
- Passing Technology Properties, on page 8-4
- Specifying Padtype and Port Information, on page 8-5

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 8-3.

1. Start with a design using one of the following vendors and technologies: Actel (except 500K and PA), Altera FLEX10K, ACEX, or APEX families, Xilinx Virtex or Spartan-3 families, Lattice, or QuickLogic.
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 8-3.
   - 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 (except 500K and PA)	`alspin` {*pin_number*}
Altera APEX	`altera_chip_pin_lc` {*pin_number*}

Altera FLEX10K/ACEX	`altera_chip_pin_lc {@`*`pin_number`*`}`
Lattice	`loc {`*`pin_number`*`}`
QuickLogic	`ql_placement {`*`pin_number`*`}`
Xilinx	`xc_loc {`*`pin_number`*`}.` See *Controlling Placement with RLOCs,* on page 8-35 for details about relative placement.

Specifying Locations for Actel Bus Ports

You can specify pin locations for Actel bus ports, except the 500K and PA technologies. 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 an 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 8-2 and bus pin placement information described in *Specifying Locations for Actel Bus Ports,* on page 8-3.

For...	**Use...**
Relative placement of Actel macros and IP blocks	alsloc Attribute define_attribute {u1} alsloc {R15C6}
Placement of Lattice ORCA input or output registers next to I/O pads	din Attribute or dout Attribute **define_attribute {** load **} din ""**

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	orca_props Attribute 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:
	`attribute INIT of RAM1 : label is "0000";`
	or
	`/* 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	orca_padtype Attribute define_attribute {AIN[3]} orca_padtype {IBT}
	QuickLogic	ql_padtype Attribute define_attribute {clk} ql_padtype {CLOCK}
	Xilinx	xc_padtype Attribute define_attribute {a[3:0]} xc_padtype {IBUF_GTLP}
Ports	Altera	altera_io_opendrain Attribute define_attribute {alucout} altera_io_opendrain {1}
	Altera	altera_io_powerup Attribute define_attribute {seg [31:0]} altera_io_powerup {high}
	Xilinx	xc_isgsr Directive define_attribute {bbgsr.gsrin} xc_isgsr {1}
	Xilinx	xc_pullup/xc_pulldown Attribute **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 8-6
- Customizing Netlist Formats, on page 8-7

Targeting Output to Your Vendor

You can generate output targeted to your vendor.

1. To specify the output, click the Impl 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) *_sdc.sdc	Designer Series
Altera Flex and Acex	EDIF (.edf) AHDL (.tdf)	MAX+PLUSII or Quartus II
Altera Apex, Stratix, Statix-II, Stratix-GX, Mercury, Max-II, Excalibur, Cyclone, Cyclone-II	Verilog (.vqm)	Quartus II
Altera Max	EDIF (.edf) AHDL (.tdf)	MAX+PLUSII
Atmel	EDIF (.edf)	Figaro
Cypress	VHDL (.vhn)	Warp
Lattice	EDIF (.edf)	ispExpert
Lattice Mach	EDIF (.edf) or .src	ispExpert
Lattice Orca	EDIF (.edn)	ispLEVER

Vendor	Output Netlist	P&R Tool
QuickLogic	EDIF (.qdf or .edf)	SpDE
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

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

   See *Specifying Result Options,* on page 3-9 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 3-64.

Customizing Netlist Formats

The following table lists some attributes for customizing your Actel, Altera, and Xilinx output netlists:

For...	Use...
Netlist formatting	syn_netlist_hierarchy Attribute (Altera, Xilinx, Actel) define_global_attribute syn_netlist_hierarchy {0}
EDIF formatting	syn_edif_bit_format Attribute (Xilinx) define_global_attribute syn_edif_bit_format {%n<%i>}
	syn_edif_name_length Attribute (Xilinx) define_global_attribute syn_edif_name_length { restricted }
	syn_edif_scalar_format Attribute (Xilinx) define_global_attribute syn_edif_scalar_format {%u}
Bus specification	syn_noarrayports Attribute (Altera, Xilinx, Actel) define_global_attribute syn_noarrayports {1}

Working with 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 8-8
- Using ACTGen Macros, on page 8-9
- Working with Radhard Designs, on page 8-10

For additional Actel-specific information, see *Passing Information to the P&R Tools*, on page 8-2 and *Generating Vendor-Specific Output*, on page 8-6.

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 8-9. For general information about working with black boxes, see *Defining Black Boxes for Synthesis*, on page 6-30.

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`.

`proasic`	ProASIC (500K) and ProASIC PLUS (PA and ProAsic3E) macros
`actel`	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 8-8. For general information about working with black boxes, see *Defining Black Boxes for Synthesis,* on page 6-30.

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

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 cc:

VHDL	Verilog

`library synplify;`
`use synplify.attributes.all;`
`attribute syn_radhardlevel of`
`  behav: architecture is “cc”;`
	`module module_b (a, b, sub,`
	`  clk, rst) /*synthesis`
	`  syn_radhardlevel=”cc”*/;`

   - Make sure that the corresponding Actel macro file (see step 1) is the first file listed in the project.

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_cc, enter the following in the module source file:

VHDL	Verilog

`library synplify;`
`use synplify.attributes.all;`
`attribute syn_radhardlevel of`
`  bl_int: signal is “tmr_cc”`
	`reg [15:0] a1_int, b1_int`
	`  /*synthesis syn_radhardlevel`
	`  = ”tmr_cc”*/;`

Working with 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 *Design Guidelines*, on page 6-2. This section discusses the following topics that are specific to Altera technologies:

- APEX Design Tips, next
- FLEX Design Tips, on page 8-12
- Determining ROM Implementation, on page 8-12
- Working with Altera EABs and ESBs, on page 8-14
- Working with Altera PLLs, on page 8-15
- Implementing Megafunctions with Clearbox, on page 8-16
- Packing I/O Cell Registers, on page 8-18
- Using LPMs in Simulation Flows, on page 8-20
- Working with Quartus II, on page 8-22

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 6-30
- Inferring RAMs, on page 6-54
- Inferring Shift Registers, on page 6-80

- Working with LPMs, on page 6-87
- Passing Information to the P&R Tools, on page 8-2
- Generating Vendor-Specific Output, on page 8-6

APEX Design Tips

Use these techniques when working with APEX designs:

- 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*, on page 8-18 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 Synplicity 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 *Adding Attributes and Directives*, on page 3-66 for information.

Format	Example
Verilog	`reg [3:0] z /* synthesis syn_romstyle="block_rom" */;`

VHDL	`signal z : std_logic_vector(3 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is "block_rom";`

Constraint file syntax	`define_attribute {z_20[3:0]} syn_romstyle {block_rom}`

– 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.

2. 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.

Format	Example
Verilog	`reg [3:0] z /* synthesis syn_romstyle="logic" */;`

VHDL	`signal z : std_logic_vector(3 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is "logic";`

Constraint file syntax	`define_attribute {z_20[3:0]} syn_romstyle {logic}`

- 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.
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 `READ`s. 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.

Working with Altera PLLs

The synthesis software recognizes the Altera PLL component, altpll. The following procedure shows you how to take advantage of this component and use it in your design.

1. Use the Altera megafunction wizard to generate structural VHDL or Verilog files for the Altera PLLs in your design.

2. If you are using VHDL, comment out the LIBRARY and USE clauses in the file generated by the Altera Megawizard tool. The following shows an example of the lines to be commented out:

   ```
   LIBRARY altera_mf;
   USE altera_mf.altera_mf_components.all;
   ```

   You comment out these lines because the altpll component is declared in the Megawizard file before instantiation, so you do not need references to anothe vhd files that contains the component declaration.

   However, if you need the component declaration to be compatible with a particular Quartus version, use the vhd files packaged with the software in the lib/altera directory and named for the Quartus version. For example, the altera_mf42.vhd file is intended for use with Quartus 4.2

3. If you are using Verilog, do not do anything, as the mapper understands the altpll component.

   However, for compatibility with different Quartus versions, altera_mf.v files are packaged with the software in the lib/altera directory. Use the file that corresponds to the Quartus version you intend to use.

4. Instantiate the altpll component in your design.

5. Add the Megawizard Verilog or VHDL files to your project.

6. 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 put in the same clock group.

7. 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 to synthesize. The synthesis software forward-annotates the PLL input constraints.

Implementing Megafunctions with Clearbox

The Synplify Pro synthesis software treats user-instantiated Quartus megafunctions as black boxes, because they do not come with any timing information. This prevents the synthesis tool from making any timing-driven optimizations at the megafunction boundary. For example, the synthesis software does not move the registers of a pipelined LPM_MULT to improve FMAX. Altera Clearbox provides structural information for modules containing the following primitives stratix_lcell, stratix_mac_mult, stratix_mac_out, and stratix_ram_block.

The following procedure shows you how to use Altera Clearbox to implement megafunctions as clear boxes, with timing and resource information that can be used for synthesis. Use this procedure with Stratix, Stratix II, Cyclone, and Cyclone II technologies.

1. Use the Altera megafunction wizard to generate structural VHDL or Verilog files for the megafunctions in your design.

   The Clearbox output has its full body either in VHDL or Verilog with no parameters remaining, other than the parameters remaining on the ATOMs The synthesis software uses this timing and resource information for the megafunctions, but does not synthesize the internals of the megafunctions.

2. If you are using VHDL, it is recommended that you comment out the LIBRARY and USE clauses in the file generated by the Altera Megawizard tool. 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;
   ```

   You comment out these lines because the Altera Megawizard file declares the Clearbox components before instantiating them, so you do not need references to vhd files that contain the component declarations.

The VHDL file generated by Megawizard uses the stratix.vhd declaration. Because of ongoing modifications in Quartus, you might need component declarations that match particular versions of Quartus. These component declarations are packaged with the software in the lib/altera directory and are named for the technology and Quartus version. For example, stratix_41.vhd corresponds to Quartus 4.1.

3. Instantiate the modules in your design.
4. Add the Verilog/VHDL files to your project.
   – If you are using Verilog, make sure to include the `stratix.v` file from the `lib/altera` directory. Make sure to use the version that matches the version of Quartus you are going to use. For example, if you are using Quartus 4.1, make sure you use the stratix_41.v file.

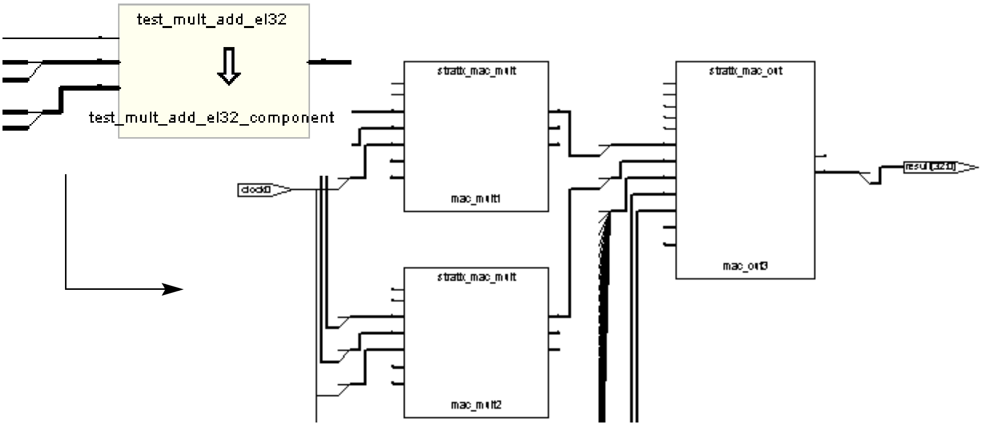
     This file contains the port and parameter definitions of the Clearbox primitives. It is not automatically included because Verilog does not support library statements.
5. Click Implementation Options, set the implementation options, and synthesize the design.
   – On the Device tab, set the target technology to Stratix, Stratix II, Cyclone, or Cyclone II.

   – On the Implementation Results tab, select the appropriate Quartus version. This is important, because this determines the format of the output .vqm file, which varies with different Quartus versions.

   – Click OK.

   – Synthesize as usual.

   The software uses the timing and resource information from the structural Verilog/VHDL files to calculate paths more accurately. Megafunctions are implemented as hierarchical instances, not black boxes. This figure shows you can push into the mult_add megafunction and see the stratix_mac_mult and stratix_mac_out primitives it contains.



The .vqm file generated for Quartus after synthesis only contains the Clearbox module, and does not write out the internals of the module.

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

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

Packing I/O Cell Registers

You can improve input or output path timing in designs by packing registers into I/O cells with the syn_useioff attribute.

With Altera Stratix, when this attribute is enabled, registers are not packed into Multiply/Accumulate (MAC) blocks. The highest to lowest priority is registers, then ports, then global.

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	`module test(d, clk, q) /* synthesis syn_useioff=1 */;`

VHDL	`architecture rtl of test is`
	`attribute syn_useioff : boolean;`
	`attribute syn_useioff of rtl : architecture is true;`

Constraint file syntax	`define_global_attribute syn_useioff 1`

2. To set the attribute locally, set syn_useioff=1 on a port.

Format	Example

Verilog	`module test(d, clk, q);`
	`input [3:0] d;`
	`input clk;`
	`output [3:0] q /* synthesis syn_useioff=1 */;`
	`reg q;`

VHDL	`entity test is`
	`port (d : in std_logic_vector (3 downto 0);`
	`clk : in std_logic;`
	`q : out std_logc_vector (3 downto 0);`
	`attribute syn_useioff : boolean;`
	`attribute syn_useioff of q : signal is true;`
	`end test;`

Constraint file syntax	`define_attribute {p:q[3:0]} syn_useioff 1`

The software packs registers with asynchronous clear pins and asynchronous preset pins for APEX20KE I/O cells. The software can infer the I/O cell if you have a preset or clear, and an embedded flip-flop in the I/O cell.

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 6-87.

Simulation Flows

The simulation flows vary, depending on the method used to instantiate the LPMs. For information about instantiating the LPMs, see *Instantiating LPMs Using VHDL Prepared Components,* on page 6-94, *Instantiating LPMs as Black Boxes (Altera),* on page 6-88, and *Instantiating LPMs Using a Verilog Library (Altera),* on page 6-97. The following table summarizes the differences between the flows:

	Black Box Flow	**Verilog Library/VHDL Prepared Component Flows**
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 Max+PlusII Quartus II 1.0 or earlier	Synplify 7.0 or later Quartus II 1.1 or later

Black Box Method Simulation Flow

Use this flow if you instantiated the 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 with any synthesis release from 7.0 on, if your design contains the supported LPMs.

1. Instantiate the LPMs.
   - For VHDL designs, use the prepared components methods described in *Instantiating LPMs Using VHDL Prepared Components*, on page 6-94 or *Instantiating LPMs as Black Boxes (Altera)*, on page 6-88.
   - For Verilog designs, use the library methods described in *Instantiating LPMs Using a Verilog Library (Altera)*, on page 6-97 or *Instantiating LPMs as Black Boxes (Altera)*, on page 6-88.
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.

Working with Quartus II

The following steps show you how to use the synthesis information to run Quartus II either from the Synplicity synthesis interface or standalone.

1. Set the QUARTUS_ROOTDIR environment variable to point to your Quartus II installation directory.

2. Synthesize your design.

3. Select one of the options from the Option->Quartus menu.

   – Set options by selecting Set Option. You can configure options like cliquing and black boxes. The Quartus II software opens. The synthesized Verilog netlist, forward annotated timing constraints, and pin assignments are placed in a named Quartus project

   – To place and route interactively, select Foreground Compile. This command opens the Quartus 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.

   – To run Quartus II in batch mode, select Run Background Compile. This runs Quartus II in batch mode. View the log file that contains placement and routing data as the design compiles, and obtain a report on the completed placement and routing. Background compilation generates a log file and other files specified with Set Options.

   Using the information in the `<project_name>_cons.tcl` and `<project_name>.tcl` files, the software sets up the Quartus project, compiles it, and reads forward-annotated information from the synthesis run.

4. To run the Quartus II software standalone using the synthesis information, do the following:

   – Start Quartus II.

   – Select `<project_name>_cons.tcl` from the Run Tcl Script menu.

   The software uses the synthesis results to run Quartus II.

Working with 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 8-23
- Using Lattice GSR Resources, on page 8-24
- Inferring Carry Chains in Lattice XPLD Devices, on page 8-25
- Controlling I/O Insertion in Lattice Designs, on page 8-25
- Forward-Annotating Lattice ORCA Constraints, on page 8-26

For additional information about working with Lattice designs, see *Passing Information to the P&R Tools*, on page 8-2 and *Generating Vendor-Specific Output*, on page 8-6.

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	*synplify_install_dir*/lib/lucent/orca*.v Replace the asterisk with either 2, 3, or 4, according to the Orca series you are using
ispXPGA devices	*synplify_install_dir*/lib/lattice/lava1.v
EC/ECP devices	*synplify_install_dir*/lib/lucent/ecp.v
CPLD devices	*synplify_install_dir*/lib/cpld/lattice.v

2. 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	`library lattice;`
	`use lattice.components.all;`

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.
	`library orca*;`
	`use orca*.orcacomp.all;`

ispXPGA Devices	`library lava;`
	`use lava.components.all;`

EC/ECP Devices	`library ec;`
	`use ec.components.all;`
	`library ecp;`
	`use ecp.components.all;`

3. Instantiate the macros from the library as described in *Instantiating Black Boxes and I/Os in Verilog*, on page 6-30 and *Instantiating Black Boxes and I/Os in VHDL*, on page 6-32.

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 EC, ECP, 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 6-5.

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.

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_pad attribute on the module. Set the attribute value to 1. To disable I/O pad insertion at the module level, set the syn_force_pad attribute for the module to 0.

   - To force I/O pad insertion on an individual port, set the syn_force_pad 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_pad attribute, the synthesis design optimizes any unconnected I/O buffers away.

Forward-Annotating Lattice ORCA Constraints

For Lattice Orca, EC, and ECP designs, 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 3-64.

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
define_false_path	-from in1_int -to output
define_false_path	-from in* -to out*

   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 Synplify synthesis tool creates a `.prf` file in the same directory as your result files.

5. Start the Lattice ispLEVER place-and-route tool and run the Map stage. The place-and-route software includes the constraints from the synthesis .prf file when it generates another `.prf` file after mapping.
6. Run the PAR and BIT stages in ispLEVER.

Working with Xilinx Designs

This section contains tips for working with Xilinx designs:

- Designing for Xilinx Architectures, next
- Instantiating CoreGen Cores, on page 8-29
- Packing Registers for I/Os, on page 8-33
- Controlling Placement with RLOCs, on page 8-35
- Using Clock Buffers in Virtex Designs, on page 8-36
- Reoptimizing With EDIF Files, on page 8-39
- Instantiating Special I/O Standard Buffers for Virtex, on page 8-38

For additional Xilinx-specific techniques, see *The Xilinx MultiPoint Synthesis Flow*, on page 10-48, *Using the Xilinx Modular Flow*, on page 10-54, *Working with Gated Clocks*, on page 6-99, *Inferring RAMs*, on page 6-54, and *Inferring Shift Registers*, on page 6-80. 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 *Design Guidelines*, on page 6-2.

- 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.

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 8-30.

1. Define the core as a black box by adding the syn_black_box attribute to the module definition line.

   ```
   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;
   ```

2. 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” */;
   ```

3. Generate timing and resource usage information for synthesis.
   - Use the Xilinx CORE generator to create structural EDIF netlists. For legacy cores, generate a single flat `.edf` netlist file. For newer cores, generate a top-level flat `.edf` netlist file that instantiates `.ndf` files for each hierarchical level in the design.
   - In the synthesis software, add the generated files (`.edf` only for legacy cores; `.edf` and `.ndf` for newer cores) to your project.

4. Instantiate the black box in the module or architecture.

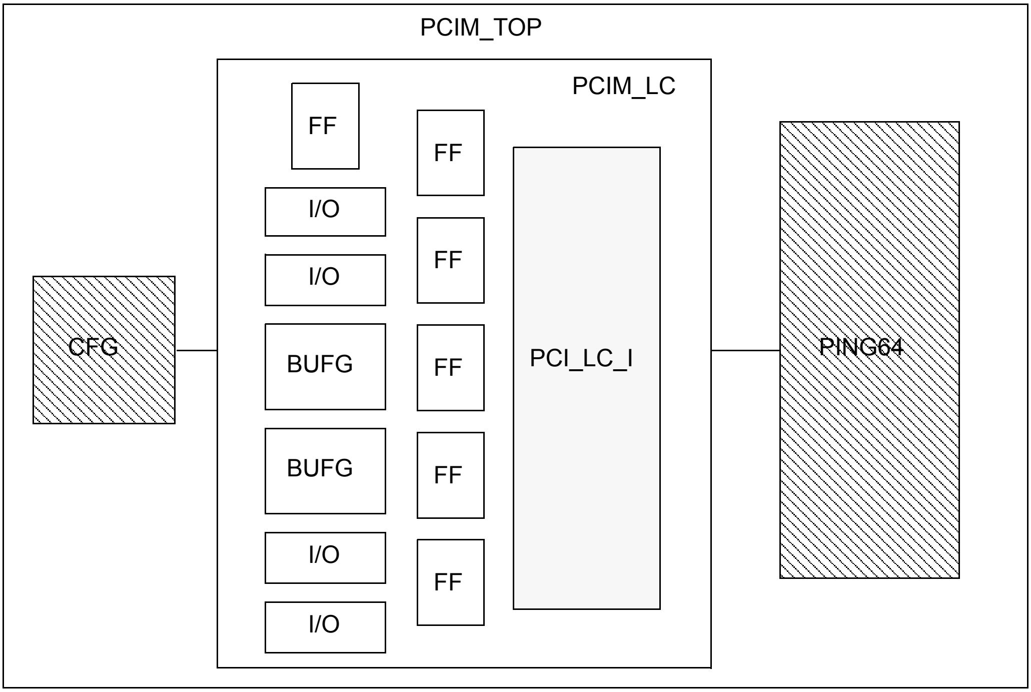
   ```
   ram64x8 r1(din, addr, we, clk, dout);
   ```

5. 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 Synplicity Virtex file (`<`*`install`*`>/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 Synplicity Virtex file (`<`*`install`*`>/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 I/Os

When a register drives an input or output, you might want to pack it in an IOB instead of a CLB. For example, when

- 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 flip-flops 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 *Adding Attributes and Directives,* on page 3-66.

   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 CLBs. 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.

Controlling Placement with 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.

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 hmap	4 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
```

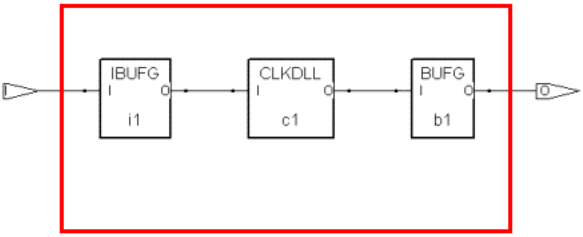
4. Create a top-level design and instantiate your design.

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 *Adding Attributes in the SCOPE Window*, on page 3-68 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
```

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, bidir_in, q_in;
IBUF_AGP i1 (.O(a_in), .I(a) ) ;
IOBUF_CTT i2 (.O(q_in), .IO(bidir) , .I(Q_int), .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_in & b;

endmodule
```

2. To specify the I/O buffers with an attribute, add the attribute in the SCOPE window (refer to *Setting Constraints in the SCOPE Window, on* page 3-18 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 should 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 Impl 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.

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 Synplicity 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.

CHAPTER 9

Design Planning for Vendors

This section provides tips for design planning Altera-specific devices and particular design conditions. Refer to:

- Design Planning with Altera Devices on page 9-2

This section also provides tips for design planning Xilinx-specific devices and particular design conditions. Refer to:

- Design Planning with Xilinx Designs on page 9-8
- Handling Xilinx Critical Paths (Design Planner) on page 9-14
- Handling Xilinx Black Boxes (Design Planner) on page 9-22
- Handling Xilinx Block RAMs (Design Planner) on page 9-24
- Handling Block Multipliers (Design Planner) on page 9-30
- Handling DSP Blocks (Design Planner) on page 9-32
- Handling Xilinx IPs (Design Planner) on page 9-35

Design Planning with Altera Devices

The following topics describe the design planning process for Altera devices.

- Stratix and Cyclone Devices on page 9-2
- Displaying Stratix Devices on page 9-2
- Creating Regions for Stratix Devices on page 9-4

Stratix and Cyclone Devices

The guidelines in this section provide tips and strategies for using the Synplify Premier Design Planner for design planning Altera devices. Design Planner supports the following Altera technologies: STRATIX, STRATIX II, STRATIX-GX, CYCLONE, and CYCLONE-II.

Displaying Stratix Devices

The high-performance Stratix, Stratix-II, and Stratix GX technology includes the following advanced features:

- Abundant memory resources for on-chip storage (512 RAM, 4K RAM, and MRAM)
- High-bandwidth DSP blocks for digital signal processing-intensive applications
- Maximized signal quality and data transfer reliability with differential I/O technology, capable of 840-Mbps performance
- Robust clock management and frequency synthesis for managing on- and off-chip timing to maximize system performance using full-featured, embedded phase-locked loops (PLLs)

Supporting Stratix Devices

Use the Design Planner to view and create regions and assign critical path logic to these regions from the Design Plan Editor. The Stratix device contains:

- A row and column coordinate system. The origin (1,1) is located at the lower-left corner of the device.
- Device features that all align on these row and column boundaries.
- The following component features: LABs (logic blocks), DSPs (digital signal processors), 512 RAMs, 4K RAMs, and MRAMs (512K RAMs). Depending upon the part and package used, the number of these blocks on the device may vary.
- Tool tips displayed as you move the mouse cursor over a feature describing what it represents and its location on the device.

The following figure shows an example of an Altera Stratix device from the Design Plan Editor.

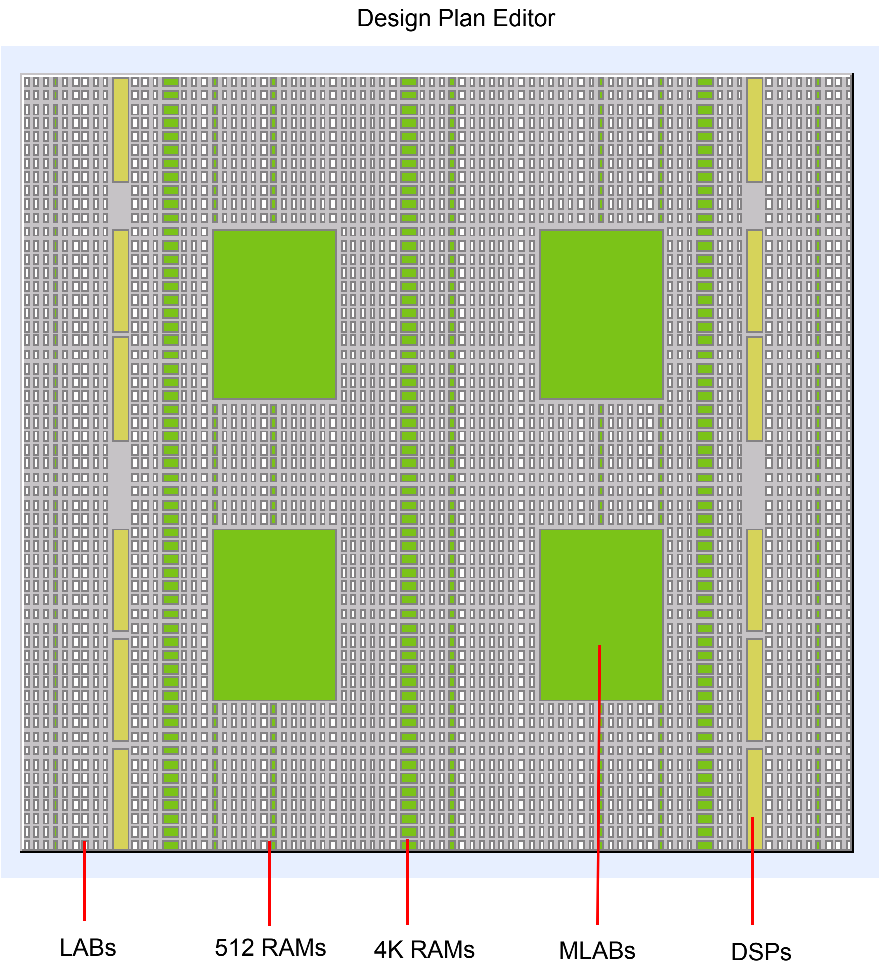


Figure 9-1: Example of an Altera STRATIX Device

Creating Regions for Stratix Devices

You can create a region by drawing a rectangle around any number of LAB, RAM, and DSP structures within the desired location on a Stratix family device. Then you can move or resize the region, as necessary. Refer to *Moving and Sizing Regions* on page 7-21 for more information.

To create a region on the device:

1. Place the cursor over the device in the Design Plan Editor and click the right mouse button to display a popup menu.
2. Select the Add Region option from the popup menu or use `Ctrl-r`.

   The popup menu disappears, but the cursor is initialized to create a region.
3. Press the left mouse button while dragging the cursor across the desired LABs/RAMs/DSPs, if available, in the selected rows/columns of the device, and then release the mouse button.

   A blue rectangle appears displaying the region you created.

Note: You can create regions that contain only LABs, only RAMs, only DSPs, or that overlap any combination of LABs, RAMs, and DSPs, as required. Regions can then be moved or resized. The Design Planner tool prevents you from creating a region that is completely contained inside any of these types of blocks; these regions are invalid.

Whenever you create, move, or resize regions, all operations snap to the row/column grid locations on the device.

Assigning to Regions

When you assign MAC/RAM/ROM blocks to a region on the device, the Design Planner software applies the following conditions, respectively:

- For MACs:
  - Place MAC blocks in a region containing DSP resources. Otherwise, the MAC block is mapped to logic and a warning message is generated in the log file (`.srr`). DSP resource utilization can also be displayed in the Design Plan view after you run Estimate Regions. Refer to *Checking Region Utilization* on page 9-7.
  - When the attribute `syn_multstyle=logic` is applied to the MAC instance, then MAC logic is mapped to logic and a warning message is generated in the log file (`.srr`).

  - When a multiplier is placed in a region without DSP resources and the attribute `syn_multstyle=lpm_mult` is applied to the multiplier, then the multiplier is mapped to the MAC block and a warning message is generated in the log file (`.srr`).
  - Do not place signed and unsigned multipliers in the same DSP block.

Note: Use the Create MAC Hierarchy optimization on the Netlist Restructure tab of the Implementation Options dialog box, to conveniently map MAC configurations together into one MAC block so that this block can be easily assigned to DSP regions for physical synthesis. This option is enabled by default for Stratix devices only.

- For RAMs/ROMs:
  - Place RAM/ROM logic in a region containing RAM/ROM resources. Otherwise, RAM/ROM logic is mapped to logic and a warning message is generated in the log file (`.srr`). Specified RAM/ROM resource utilization can also be displayed in the Design Plan view after you run Estimate Regions. Refer to *Checking Region Utilization* on page 9-7.
  - When RAM/ROM logic is placed in a region with RAM/ROM resources and the attribute `syn_multstyle=logic` is applied to the RAM/ROM instance, then the RAM/ROM is mapped to logic and a warning message is generated in the log file (`.srr`).
  - When RAM/ROM logic is placed in a region without RAM/ROM resources and the attribute `syn_ramstyle=blockram` is applied to the RAM/ROM instance, then the RAM/ROM is mapped to altsyncram and a warning message is generated in the log file (`.srr`).

Note: The RAM will not be inferred if the register driving the address or the output register is not placed in the same region.

Checking Region Utilization

The Design Planner software calculates resource capacity and usage for logic assigned to the region, and displays this information in the Design Plan view. To update and view the area of a region reflecting the actual utilization, perform the following:

1. Right-click on the device in the Design Plan Editor, then select Estimate Regions or Estimate All Regions from the popup menu. As the job runs, the region is greyed-out and you can:
   - View a label placed in the upper-left corner of the region displaying the elapsed time of the job. This label is removed when the estimation job completes. Est Pending appears in the upper-left corner of all other regions waiting for region estimation.
   - View the status in the Tcl Script window.
   - Or, select Run->Job Status.
2. Click on Regions in the Design Plan Hierarchy Browser. This should update information for the rgn in the Design Plan view with statistics for the assigned logic.
3. To choose desired options to report, right-click in the Design Plan view and select Show/Hide Columns from the pull-down menu. You can display region usage for any of the following from the Select Columns dialog box:
   - Area, Area Use, Area Use (%)
   - DSP, DSP Use, DSP Use (%)
   - RamBits, RamBit Use, and RamBit Use (%)

Design Planning with Xilinx Designs

The guidelines in this section provide tips and strategies for using the Synplify Premier Design Planner for design planning Xilinx devices. The Design Planner support the following Xilinx technologies: Virtex-4, Virtex-II Pro, Virtex-II, Virtex-E, Virtex, and Spartan-3. Refer to the following sections:

- Displaying Xilinx Device Resources on page 9-8
- Creating Regions for Xilinx Designs on page 9-12
- Handling Xilinx Critical Paths (Design Planner) on page 9-14
- Handling Xilinx Black Boxes (Design Planner) on page 9-22
- Handling Xilinx Block RAMs (Design Planner) on page 9-24
- Handling Block Multipliers (Design Planner) on page 9-30
- Handling DSP Blocks (Design Planner) on page 9-32
- Handling Xilinx IPs (Design Planner) on page 9-35

Displaying Xilinx Device Resources

The Design Planner tool displays the following resources on the Xilinx device, if relevant:

- Block FIFOs (Virtex-4 devices only)
- Digital Signal Processing Elements (DSPs - Virtex-4 devices only)
- Block RAMs
- Block Multipliers
- Digital Clock Managers (DCMs)
- I/O Banks

The following example displays these resources on a Virtex-4 device. The figure shows the lower-left corner of the device. Notice that I/O banks are located within the device.

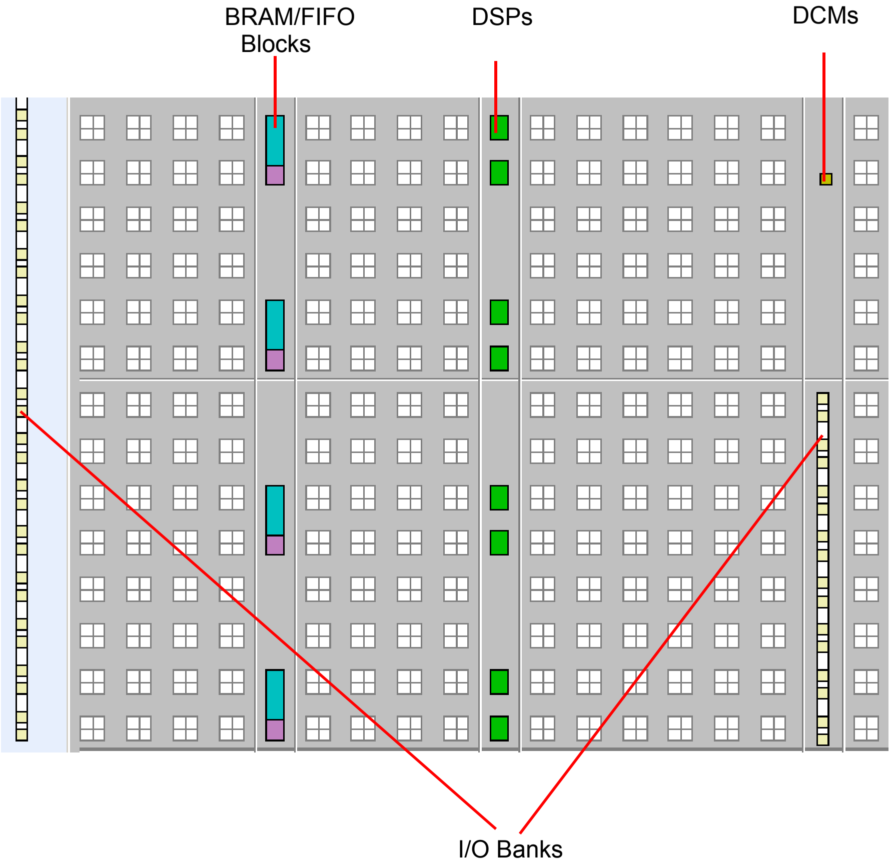


Figure 9-2: Xilinx Virtex-4 Device

The following example displays these resources on a Virtex-II device.

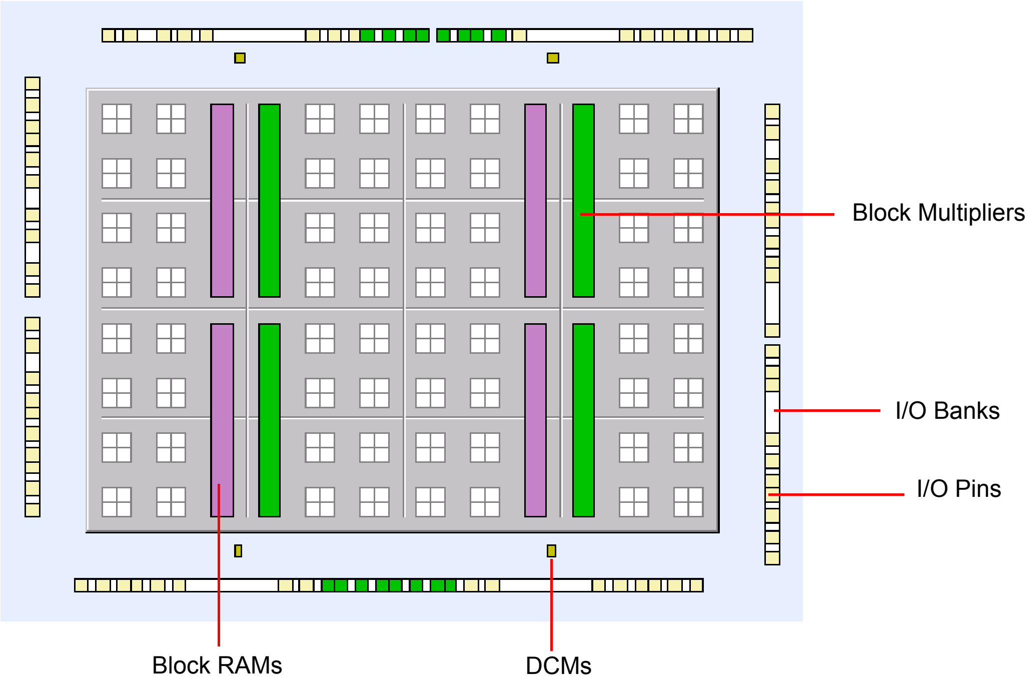


Figure 9-3: Xilinx Virtex-II Device

Block FIFOs

Use the Design Planner to view block FIFOs displayed on the Virtex-4 device in the Design Plan Editor. See Figure 9-2 on page 9-9. In the Virtex-4 architecture, dedicated logic in the block RAM enables you to easily implement synchronous or asynchronous FIFOs. This eliminates the need for additional CLB logic for counter, comparator, or status flag generation, and uses just one block RAM resource per FIFO.

DSPs

Use the Design Planner to view digital signal processing blocks (DSP48) displayed on the Virtex-4 device in the Design Plan Editor. These application specific module blocks provide a programmable mix of logic, memory, I/O processors, clock management, and digital signal processing. For DSP support, refer to *Handling DSP Blocks (Design Planner)* on page 9-32.

Block RAMs

Use the Design Planner to view block RAMs displayed on the device in the Design Plan Editor. For block RAM support, refer to *Handling Xilinx Block RAMs (Design Planner)* on page 9-24.

Block Multipliers

Use the Design Planner to view block Mults displayed on the device which appear as rectangles adjacent to the block RAM resources in the Design Plan Editor. See Figure 9-3 on page 9-10. For Block Mult support, refer to *Handling Critical Paths with Large Multiplexers* on page 9-21.

I/O Banks

Use the Design Planner to view I/O banks displayed on the device in the Design Plan Editor. See Figure 9-3 on page 9-10. I/O banks group device pins together within a rectangular area. I/O Bank 0 starts at the top-left corner of the device with I/O banks incrementing in a clockwise direction around the device. You can move the mouse cursor over any of these I/O regions to identify which I/O bank it belongs. For more information about pin assignment support, refer to *Pin Assignments* on page 7-6.

Digital Clock Managers (DCMs)

Use the Design Planner to view DCMs on Virtex-II and Spartan-3 devices in the Design Plan Editor. See Figure 9-3 on page 9-10. The number of DCMs may vary depending upon the device selected. For example, DCMs are aligned with the block RAMs resources on the device and contain the following:

- For Virtex-II and Spartan-3 devices, each block RAM column contains two DCMs. One DCM is located above the top block RAM column and the other DCM is located below the bottom block RAM column.

- For Virtex-II Pro devices, a maximum of 4 or 8 DCMs can be located on the device depending upon the device part and package selected. When the device includes 4 DCMs, they are aligned above and below the right-most and left-most block RAM columns. When the device includes 8 DCMs, they are aligned above and below the two right-most and two left-most block RAM columns.

Note that DCM support provides the following:

- You can move the mouse cursor over any resource on the device to display a tool tip describing what it represents.
- You only can display resources.

Creating Regions for Xilinx Designs

Use the following recommendations to help you design plan regions in the Design Plan Editor for Xilinx devices:

- The Configurable Logic Block (CLB) coordinate system for the device depends on the following:
  - For Virtex and Virtex-E architectures, the CLB coordinate location row=1, col=1 starts at the top-left corner of the device.
  - For Virtex-4, Virtex-II Pro, Virtex-II, and Spartan-3 architectures, the CLB coordinate location row=1, col=1 starts at the bottom-left corner of the device.
- 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 9-17.
- It is *not* recommended that you overlap regions on Xilinx devices for some designs. The Xilinx place-and-route tool cannot always place these designs, and therefore, can potentially create an error. Keep in mind, however, that the Design Planner software can still support regions that overlap.
- For Spartan-3 architectures, it is recommended that you create regions that are at least 6 x 6 CLBs as a minimum size to obtain optimum region utilization.

Regions Driving Global Bus Signals

Regions that drive a global bus signal should include row or column overlap to all other regions that drive the same global bus signal.

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).

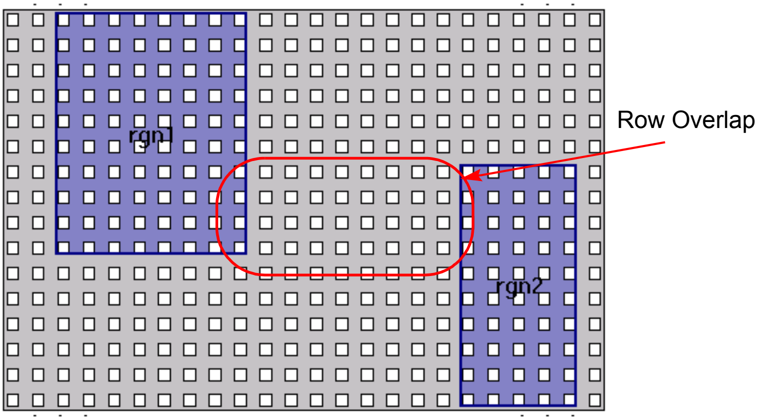


Figure 9-4: Overlapping Regions–1

Multiple Regions Driving a Global Bus

If multiple regions drive a global bus signal make sure that there is some overlap between all the regions that drive the signal.

Handling Xilinx Critical Paths (Design Planner)

These sections describe the following guidelines for critical paths:

- Splitting a Critical Path into Multiple Regions
- Creating Smaller Regions for Long Critical Paths
- Handling Critical Paths with High Fanout Nets
- Handling Critical Paths with Cascading Cells or Carry Chain Logic
- Handling Critical Paths with Bit Slicing
- Handling Critical Paths with Pipelining
- Handling Designs with Multiple Critical Paths
- Handling Critical Paths with Large Multiplexers

Splitting a Critical Path into Multiple Regions

If a critical path contains logic that should be placed very closely together, you can try 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.

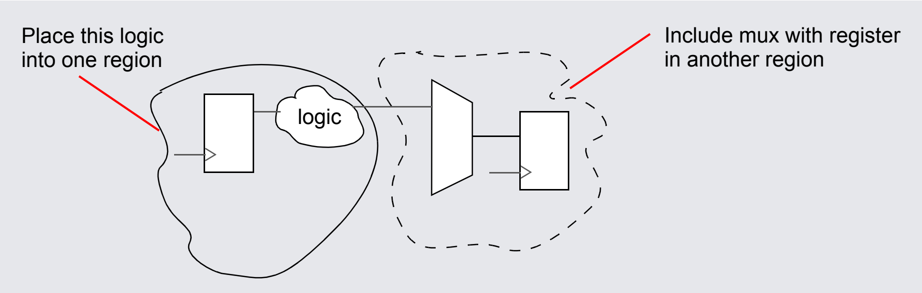


Figure 9-5: Splitting a Critical Path into Multiple Regions

Creating Smaller Regions for Long Critical Paths

For the most optimal timing results, you need to 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.

For example, say the critical path is contained in modules A, B, and C of the design. Determine the start point of the critical path in module A and where it ends in module A. Assign this portion of the critical path to one region. Determine the start point of the critical path in module B and where it ends in module B and assign this portion of the critical path to a second region. Then, determine the start point of the portion of the critical path that is contained in module C. Assign this portion of the critical path to a third region.

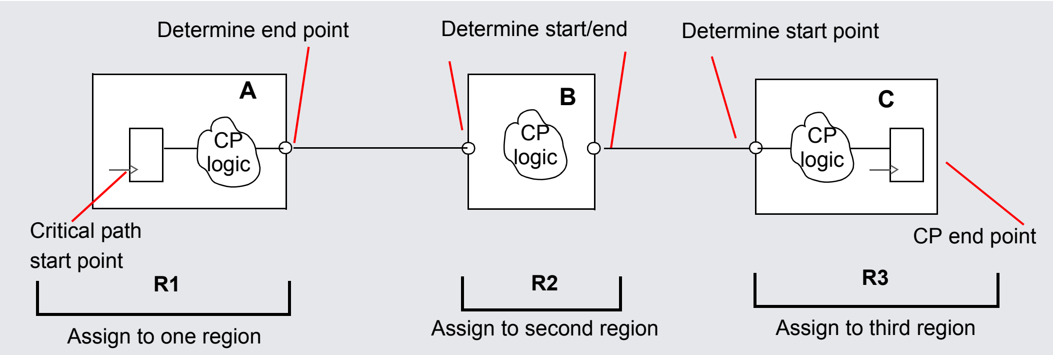


Figure 9-6: Place Critical Path into Smaller Regions

Handling 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. When a critical path contains a design with high fanout nets, it is advisable to place that critical path in a region that includes the first or last CLB row *near the middle column of the device*. (The secondary routing buffers can only be accessed from the first and last CLB row close to the vertical center of the device.)

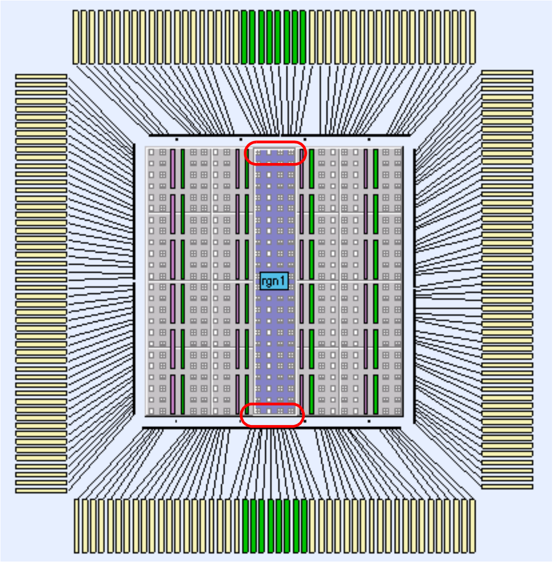
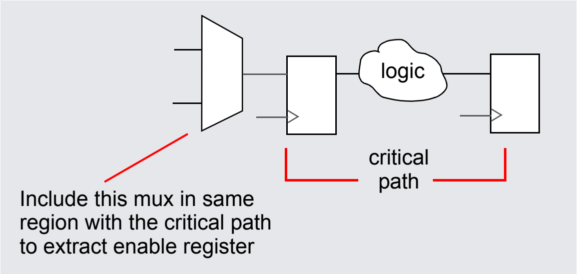


Figure 9-7: Critical Paths with High Fanout Nets

Extracting Enable Registers

For Design Planner software to effectively extract an enable register, both the mux and register must be kept together when applying design planning. If a register of a critical path is fed by a mux that is outside of the critical path, you must constrain the mux along with the register and the rest of the critical path logic into the same region.



Include this mux in same region with the critical path to extract enable register

Figure 9-8: Extracting Enable Register

Also, if the critical path is too large to fit into one block region so that you need to divide the path among two or more regions, place the mux along with the register into the same region to ensure that the enable register is extracted.

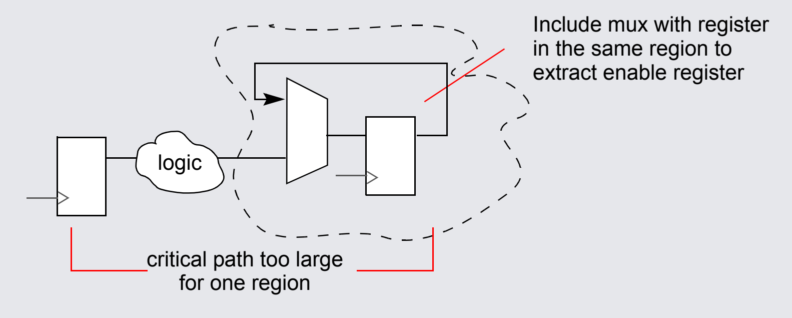


Figure 9-9: Extracting Enable Register with Large Critical Path

Handling Critical Paths with Cascading Cells or Carry Chain Logic

If a critical path contains either cascading cells or carry chains, the Carry Chain Design Rule Check (DRC) ensures that the region is large enough to support the cascade/carry chain assignment of:

- 2 bit slices/CLB for Virtex-E and Virtex
- 4 bit slices/CLB for Virtex-4, Virtex-II Pro, Virtex-II, and Spartan-3

For example, an 8-bit adder requires that you create a region with at least 4 CLBs in the vertical direction to accommodate the carry chain for Virtex and Virtex-E designs. If a region is not large enough, the Xilinx place-and-route tool fails.

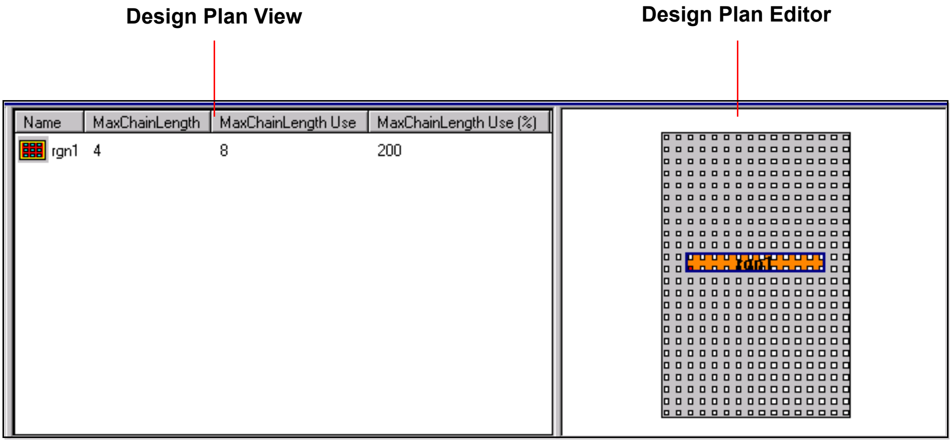
The Carry Chain DRC checks will be implemented after you have created a region and have assigned cascade/carry chain logic to that region:

- The Design Plan View displays:
  - MaxChainLength — The length of the longest cascade/carry chain that can fit into the specified region. This maximum length accommodates all cascade/carry chains less than or equal to this length.

- – MaxChainLength Use — The length of the longest cascade/carry chain assigned to the region, after you run Estimate Regions for the specified region from the Design Plan Editor.
- – MaxChainLength Use (%) — The 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 Estimate Regions for the specified region from the Design Plan Editor. This percentage can show that the cascade/carry chain exceeds its capacity to fit into the region.

- A Carry Chain DRC violation changes the color of the region area in the Design Plan Editor to orange. You must then resize the region to avoid a place and route failure.

  See *Design Plan Editor with Cascade/Carry Chain* on page 9-18.



Design Plan View

Design Plan Editor

Name	MaxChainLength	MaxChainLength Use	MaxChainLength Use (%)
rgn1	4	8	200

Figure 9-10: Design Plan Editor with Cascade/Carry Chain

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.

For example, if a 16-bit multiplexer is too large to fit in one region, then:

- Use bit-slicing to divide the datapath.
- Replicate the register and place with the common logic of the critical path.
- Place one half of the critical path in one region and the other half of the critical path in another region.

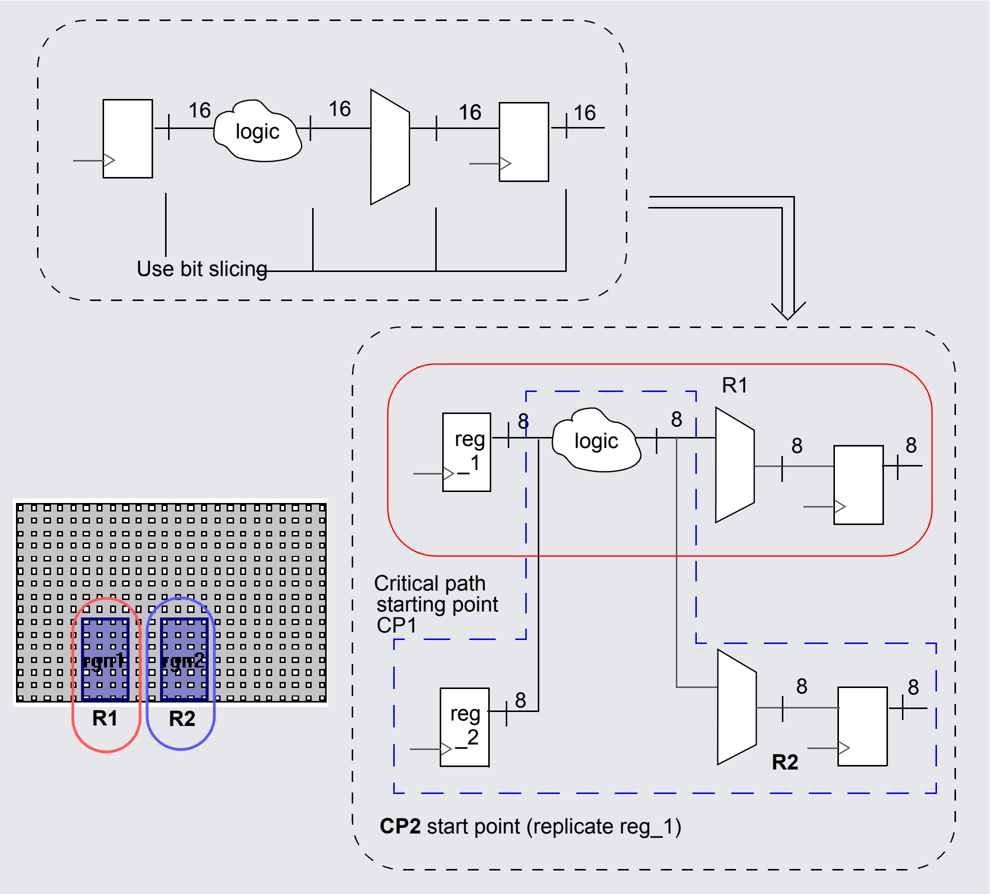


Figure 9-11: Critical Path with Bit Slicing

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:

- Place the critical path starting point and logic in one region.
- Place the multiplier or ROM with the pipeline register in another region.

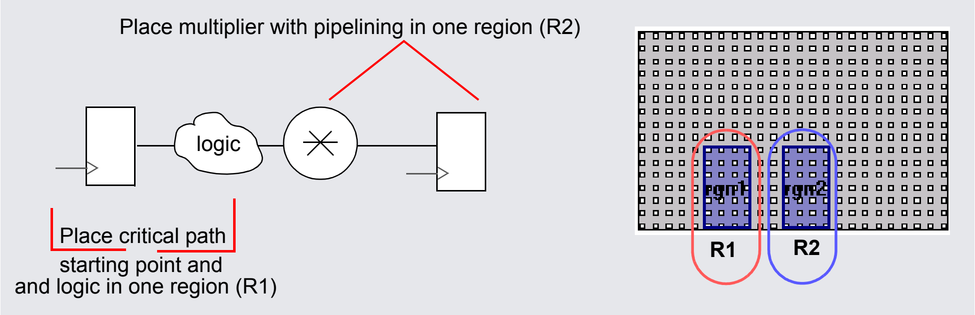

Place multiplier with pipelining in one region (R2)
logic
Place critical path
starting point and
and logic in one region (R1)
rgn1
rgn2
R1
R2

Figure 9-12: Critical Path with Pipelining

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 critical path common logic in a separate region and constrain any unrelated logic that is connected to each starting point registers in separate regions, respectively. Place the common logic region between these regions to minimize their distances.

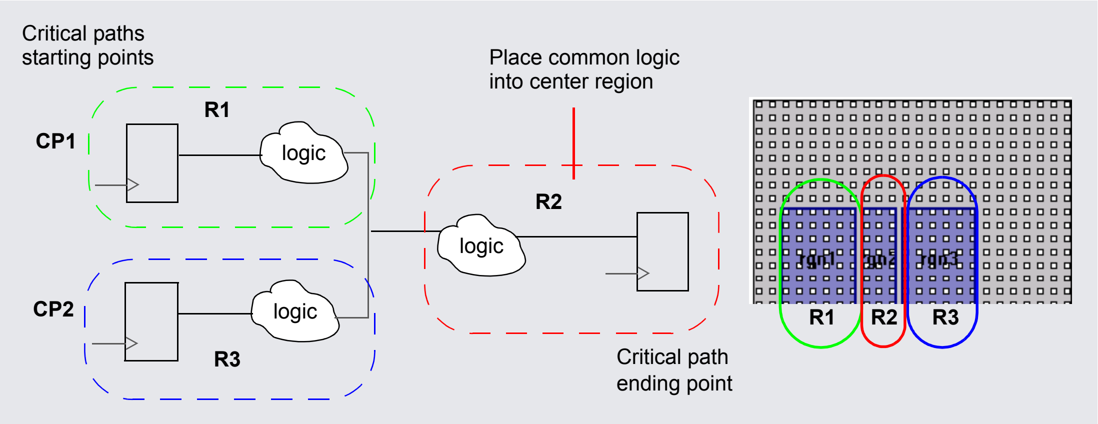


Figure 9-13: Multiple Critical Paths with Same End Point

Handling Critical Paths with Large Multiplexers

If the critical path contains a large multiplexer (MUX), make sure that the region containing the MUX also includes the control logic for that MUX.

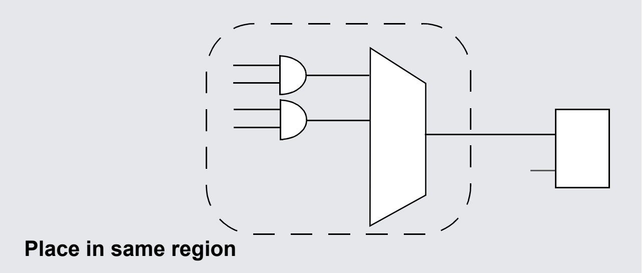


Figure 9-14: Critical Paths Containing Large Multiplexers

Handling Xilinx Black Boxes (Design Planner)

The following topics describe handling Xilinx black boxes in the Design Planner.

- Design Planning Xilinx Black Boxes on page 9-22
- Creating Block RAM Regions on page 9-25
- Assigning to Block RAM Regions on page 9-27

Design Planning Xilinx Black Boxes

You can design plan black boxes as follows:

- When you use the Design Plan Editor to assign a black box to a region, then placement constraints will be written to the `.ncf` file for that black box. Synplify Premier software will also search for the syn_resources attribute, to determine the type of black box (LUTs, registers, or block RAMs) and the size of the black box module (number of LUTs, registers, or block RAMs) to constrain.

  The Design Planner software supports the following types of black box:

  – Contains only LUTs.

  – Contains only block RAMs.

  – If you do not define the black box, by default, it is treated as a logic only black box.

  – Cannot support a mixed black box (logic and block RAMs in the same module). If this occurs, a warning message is displayed and constraints are not written to the`.ncf` file.

- Another way you can constrain a black box to a specific CLB location or region is by placing a constraint in the user constraint file (`.ucf`) for the Xilinx place-and-route tool.

  – Instantiate the black box.

  – If a critical path contains a black box, then place the black box in one region and the logic portion in another region as shown below.

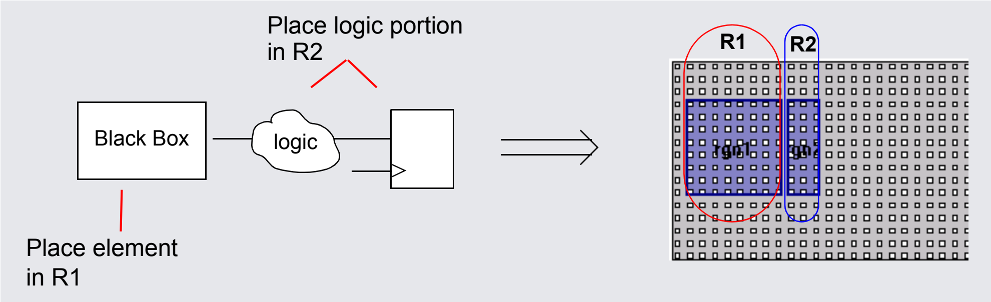


Figure 9-15: Critical Path Contains Black Box

Handling Xilinx Block RAMs (Design Planner)

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. The Design Planner tool allows you to create block RAM regions in the Design Plan Editor. This can help you visualize where block RAMs are placed on the device.

Block RAM support includes the following:

- The target FPGA device footprint displays the following resources: CLBs, Block RAMs (BRAMs), and I/O pins.
  - CLBs have their unique coordinate system starting at location (row=0, col=0). On Virtex-4, Virtex-II Pro, Virtex-II, and Spartan-3 devices, CLBs are subdivided into quadrants with row and column labels reflecting two units per CLB.
  - Block RAMs have their unique coordinate system starting at location (row=0, col=0).
  - A tool tip displays the memory size and coordinate location of block RAMs when you drag the cursor over these locations.
  - Refer to *Pin Assignments* on page 7-6 for information on I/O pins.
- The height and width of BRAMs resemble the images viewed from the Xilinx place-and-route tool. For example, the height of BRAMs:
  - Span 4 CLB rows (8 slices) on Virtex-4, Virtex-II Pro, Virtex-II, and Spartan-3 devices
  - Span 4 CLB rows on Virtex and Virtex-E devices

Creating Block RAM Regions

Use the Design Plan Editor to create block RAM regions. Block RAM regions can:

- Consist only of CLBs, only block RAMs, or a combination of CLBs and block RAMs.
- Overlap with each other.
- 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.

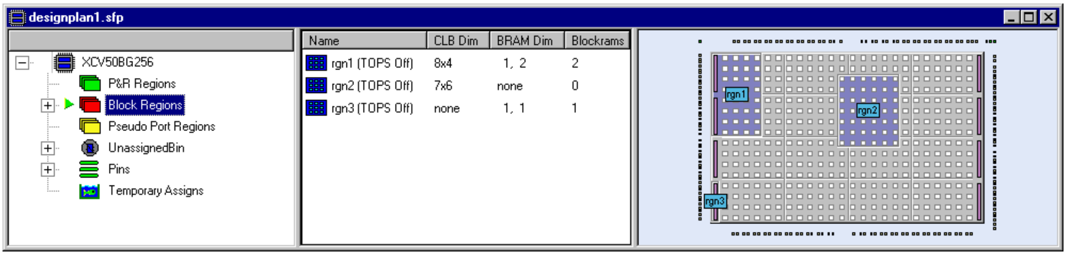
To create a block RAM region:

1. Place the cursor over the Design Plan Editor view and click the right mouse button to display a dialog box.
2. Select Add->Block Region from the dialog box.
3. Press the left mouse button while dragging the cursor across the desired rows and columns and then release the mouse button.

   A blue rectangle appears displaying the region you created.

Note: You can also move or resize block RAM regions. Any changes made to the region are reflected in the Design Planner view.

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. See the following example.



designplan1.sfp

- XCV50BG256
  - P&R Regions
  - Block Regions
  - Pseudo Port Regions
  - UnassignedBin
  - Pins
  - Temporary Assigns

Name	CLB Dim	BRAM Dim	Blockrams
rgn1 (TOPS Off)	8x4	1, 2	2
rgn2 (TOPS Off)	7x6	none	0
rgn3 (TOPS Off)	none	1, 1	1

Figure 9-16: Creating Block RAM Regions

The following tips are recommended when you create block RAM regions:

- 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. See *Critical Path Contains Block RAMs (Case 1)* on page 9-26.

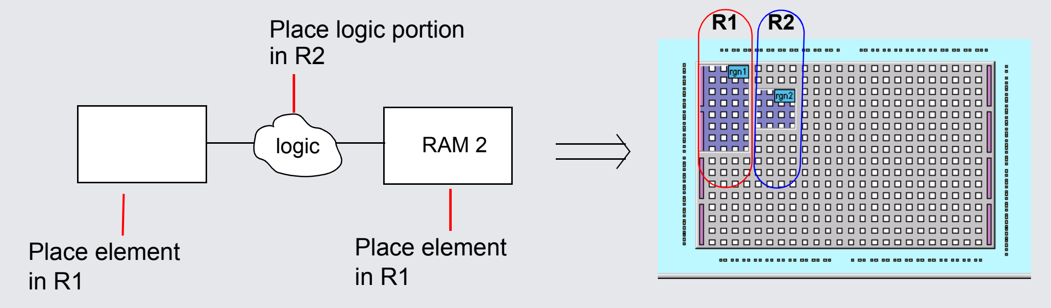


Figure 9-17: Critical Path Contains Block RAMs (Case 1)

- If the block RAMs span all CLB rows, select region R2 instead of R1. Region R2 allows more area for routing. See *Critical Path Contains Block RAMs (Case 2)* on page 9-26.

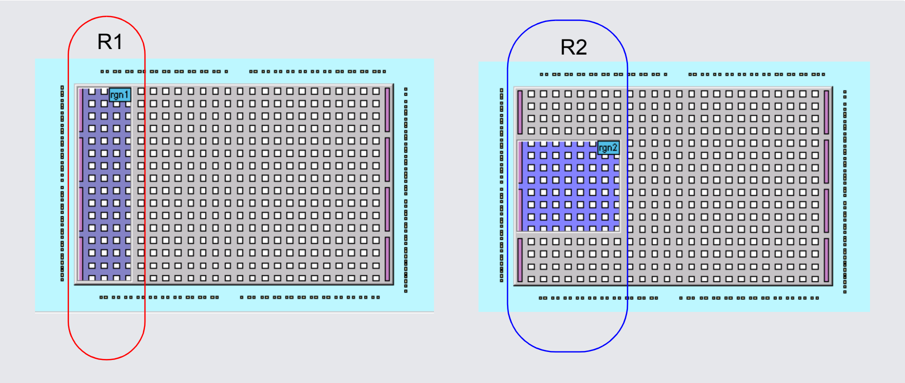


Figure 9-18: Critical Path Contains Block RAMs (Case 2)

Assigning to Block RAM Regions

When assigning RAM modules to block RAM regions:

- You might want to show rats nesting for block RAM regions. Refer to *Displaying Rats Nesting* on page 7-15.
- Drag and drop the RAM modules from the HDL Analyst RTL view to the Design Planner view.

Implementation of Block RAM Assignments

Block RAMs will be implemented as follows:

- Design Planner software automatically recognizes inferred block RAMs.
  - Make sure the register driving the RAM and the block RAM are assigned to the same region.
  - For Virtex-II and Spartan-3 designs, make sure the block RAM and its output register are assigned to the same region.
- For instantiated block RAMs with non-Xilinx primitive names, you must specify the `syn_resources "Blockrams=value"` attribute in the HDL source code.

Resolving syn_ramstyle Attribute Conflicts

Refer to the following table to resolve `syn_ramstyle` attribute conflicts when assigning RAM logic to block RAM regions. Conflicts that occur either from the HDL code or SCOPE editor are resolved similarly.

Block RAM Description	**HDL Code / SCOPE Editor**	**Design Plan Editor**
Inferred for block RAM	`block_ram`	Honors block_ram and constrains the logical RAM to the block RAM region.
Mapped to registers	`registers`	Honors registers, floats RAM logic, generates a warning.

Block RAM Description	**HDL Code / SCOPE Editor**	**Design Plan Editor**
Inferred for select_ram	select_ram	Honors select_ram, floats RAM logic, generates a warning.
Inferred for no_rw_check	no_rw_check	Honors no_rw_check and constrains logical RAM to the block RAM region.
Attribute conflicts between HDL code and SCOPE editor		Attributes specified in the SCOPE editor typically takes precedence over the HDL code when conflicts exist. It is highly recommended you avoid creating any attribute mismatches in the HDL code and SCOPE editor.

Conflicts that occur when assigning RAM logic to a particular type of block RAM region are resolved as follows:

- Block RAMs specified without an explicit type and assigned to a block RAM region will be implemented as block RAMs. You must estimate the block RAM region to ensure that block RAMs can fit into that region.
  - If block RAMs contain standard logic and are assigned to pure block RAM regions, then that 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.
  - If block RAMs do not contain standard logic and are assigned to a CLB only region, then the block RAM logic is not constrained and can float anywhere on the device.

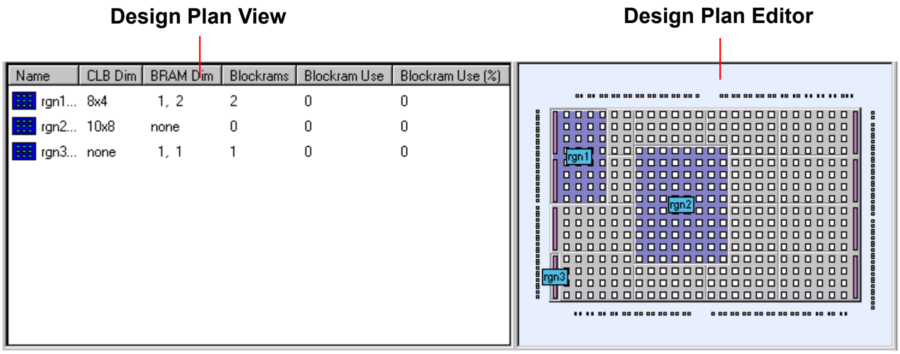
After Implementing Block RAM Regions

After you assign logical RAMs to block RAM regions, the following occur:

- The resulting implementation from the Design Plan Editor is saved to a Design Plan file (`.sfp`).
- Region estimations can be run. If a violation occurs the color of the block RAM region area in the Design Plan Editor changes to orange.
- After you run area estimations, block RAM utilization is displayed in the Design Plan view. The utilization report shows: dimensions of CLBs and

BRAMs, number of BRAMs, BRAM usage, and percentage of BRAM usage for each block RAM region.

- Right-click and select Show/Hide columns... from the Select Columns dialog box to enable these options in the Design Plan view.



Design Plan View

Design Plan Editor

Name	CLB Dim	BRAM Dim	Blockrams	Blockram Use	Blockram Use (%)
rgn1...	8x4	1, 2	2	0	0
rgn2...	10x8	none	0	0	0
rgn3...	none	1, 1	1	0	0

Figure 9-19: Block RAM Utilization from the Design Planner

- Location constraints for block RAM regions are written to a Xilinx netlist constraint file (`.ncf`). The Xilinx place-and-route tool should recognize and honor these constraints.

Handling Block Multipliers (Design Planner)

The following topics describe handling Xilinx block multipliers.

Block Multiplier Support

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. Block Mult support provides the following:

- Resources displayed on Xilinx Virtex-4, 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.
- Block Mult resources have their unique coordinate system starting at location (row=0, col=0).
- You can create a block Mult region and assign logic to the region. Thereafter, you can display block Mult capacity and utilization results for these resources.

Creating Block Mult Regions

Use the Design Plan Editor to create block Mult regions. Block Mult regions can:

- Consist only of CLBs, only block Mults, or a combination of CLBs and block Mults.

- Overlap with each other.
- Calculate region-to-region delay based on the CLB location. For regions consisting only of block Mults, region-to-region delay can be calculated from a representative CLB location.

To create a block Mult region:

1. Place the cursor in the Design Plan Editor, right-click and select Add->Block Region from the popup menu.
2. Press the left mouse button while dragging the cursor across the desired rows and columns and then release the mouse button.

   A blue rectangle appears displaying the region you created.

Note: You can also move or resize block Mult regions. Any changes made to the region are reflected in the Design Plan view.

Assigning to Block Mult Regions

When assigning multiplier instances to block Mult regions:

- You might want to show rats nesting for block Mult regions. Refer to *Displaying Rats Nesting* on page 7-15.
- To assign multipliers, drag and drop the multiplier instances from the HDL Analyst RTL view to the Design Plan Editor.

Region Utilization

The Design Planner software calculates resource capacity and usage by logic assigned to the region, and displays this information in the Design Plan view. To update and view the area of a region reflecting the actual utilization, perform the following:

1. Right-click on the device in the Design Plan Editor, then select Estimate Regions or Estimate All Regions from the popup menu. As the job runs, the region is greyed-out and you can:

- View a label placed in the upper-left corner of the region displaying the elapsed time of the job. This label is removed when the estimation job completes. Est Pending appears in the upper-left corner of all other regions waiting for region estimation.
- View the status in the Tcl Script window.
- Or, select Run->Job Status.

2. Click on Regions in the Design Plan Hierarchy Browser. This should update information for the rgn in the Design Plan view with statistics for the assigned logic.
3. To choose desired options to report, right-click in the Design Plan view and select Show/Hide Columns from the pull-down menu. You can display region usage for the following from the Select Columns dialog box: BlockMults, Block Mult Use, and Block Mult Use (%).

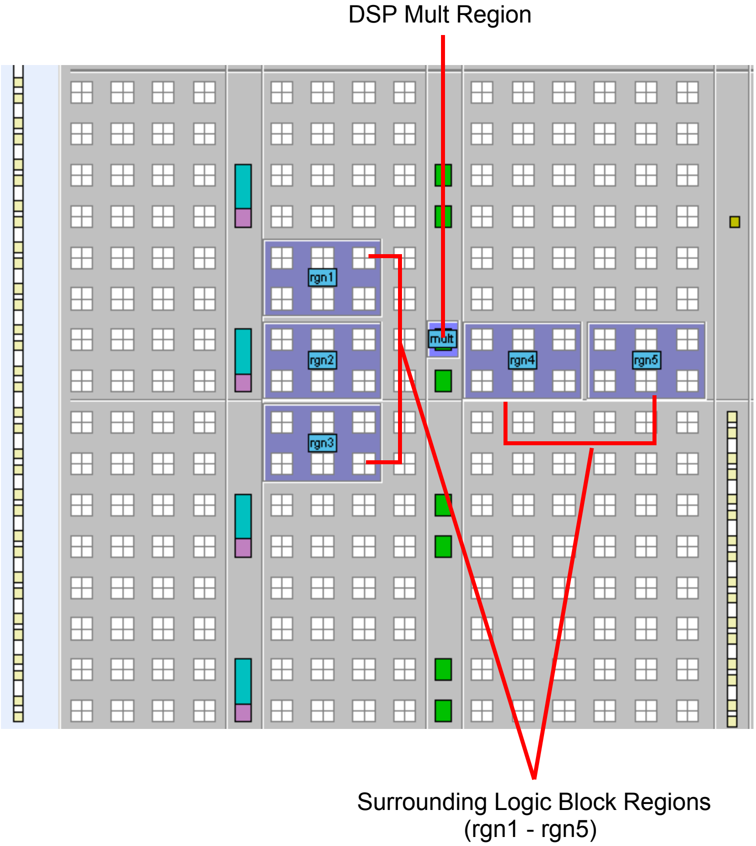
Handling DSP Blocks (Design Planner)

The DSP48 slices support many independent functions which include any of the following:

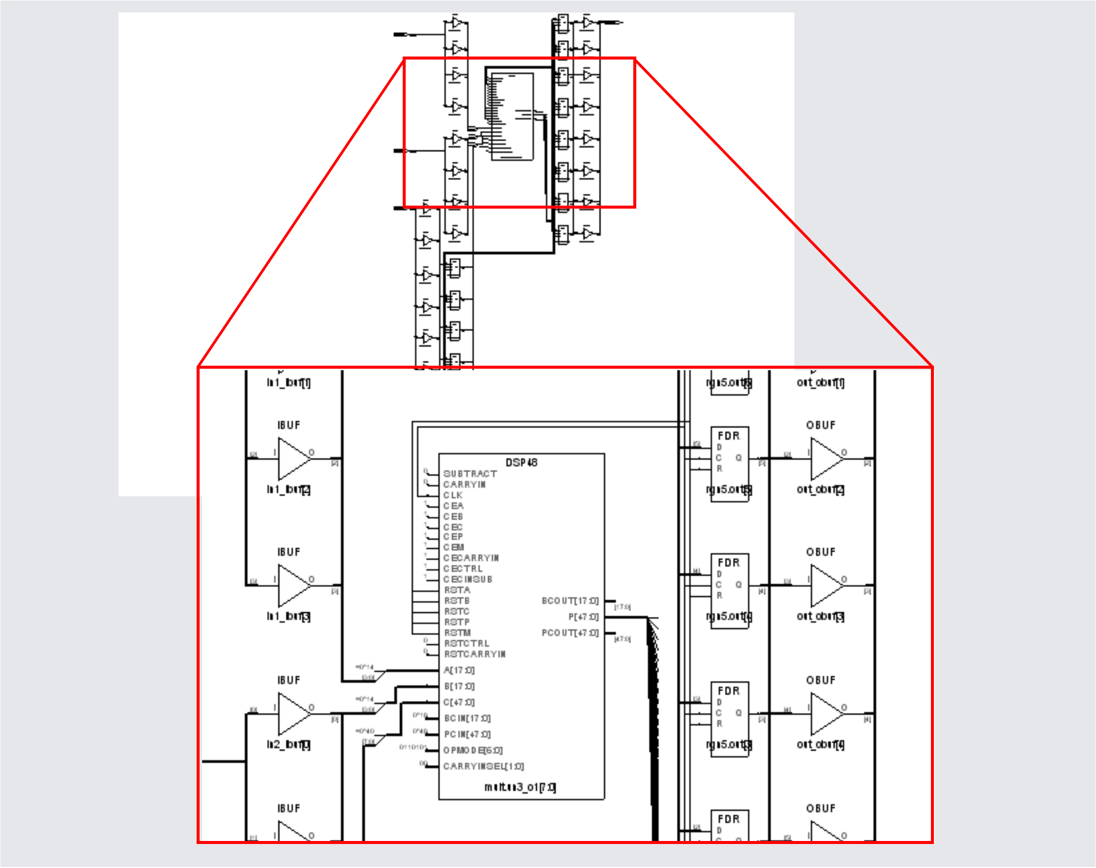
- Multipliers
- Multiplier accumulators (MACs)
- Multipliers followed by an adder
- Three-input adders
- Wide bus multiplexers
- Magnitude comparators
- Wide counters

The Design Planner tool allows you to create DSP block regions on the device. You can assign multipliers, for example, to this DSP region to help constrain the multiplier and its surrounding logic to this region for synthesis.

The following figure shows a section of the floorplan for a Virtex-4 device, where a multiplier is assigned to the mult region and its surrounding logic is assigned to regions rgn1 to rgn5.



After you run synthesis, the HDL Analyst Technology view shows that the multiplier and its surrounding logic are constrained to the DSP48 module.


in1_ibuf[1]
IBUF
in1_ibuf[2]
IBUF
in1_ibuf[3]
IBUF
in2_ibuf[0]
IBUF
DSP48
SUBTRACT
CARRYIN
CLK
CEA
CEB
CEC
CEP
CEM
CECARRYIN
CECTRL
CECINSUB
RSTA
RSTB
RSTC
RSTP
RSTM
RSTCTRL
RSTCARRYIN
A[17:0]
B[17:0]
C[47:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[6:0]
CARRYINSEL[1:0]
BCOUT[17:0]
P[47:0]
PCOUT[47:0]
mult_m3_o1[7:0]
FDR
D
C
Q
R
OBUF
I
O
out_obuf[1]
out_obuf[2]
out_obuf[3]
out_obuf[4]

Handling Xilinx IPs (Design Planner)

Synplify Premier offers a methodology to facilitate placement and timing controls for white boxes/IPs to improve quality of results (QoR) for the design. A white box IP is a RTL module contained in a HDL file and which is accompanied by a Xilinx netlist file that defines the contents of the IP. Synplify Premier physical synthesis supports the following Xilinx netlist file formats:

- `.edn` — Xilinx `.edif` netlist format used in the Intrusive IP flow. (See *Intrusive IP Flow* on page 9-35.)
- `.ngc` — encrypted Xilinx netlist format used in the Macro IP flow. (See *Macro IP Flow* on page 9-37.)
- `.ngo` — encrypted Xilinx netlist format used in the Macro IP flow. (See *Macro IP Flow* on page 9-37.)

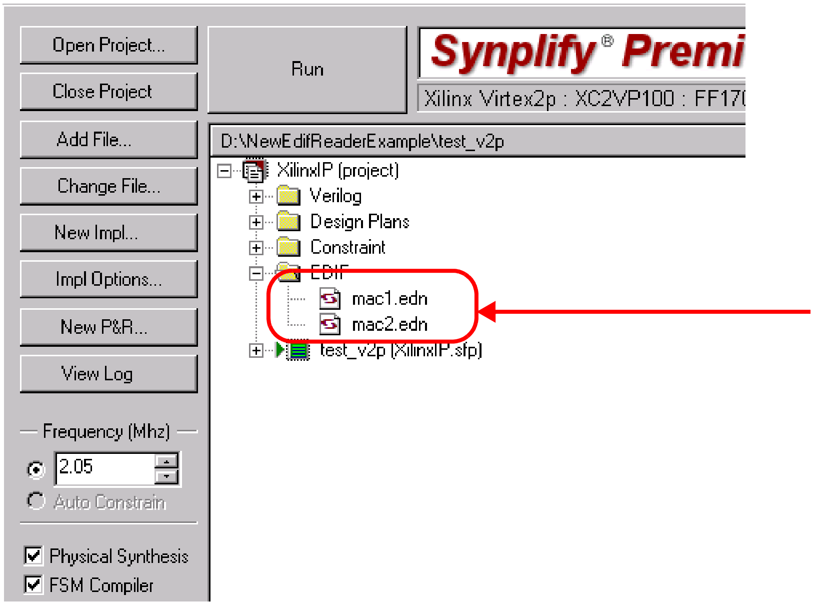
However, physical synthesis does not support black boxes and will generate an error message. A black box is a RTL module contained in a HDL file with only a wrapper specifying its input and output ports. The content of the black box is not defined.

Intrusive IP Flow

The intrusive IP flow allows the Synplify Premier software to perform logic and physical optimizations for the components contained within the IP. You must add the Xilinx netlist files (`.edn`) that include the contents of the IPs to your project. You can optionally include a design plan file (`.sfp`) which constrains the IP components to specified placement locations. The intrusive IP flow is the default mode to run physical synthesis providing the most optimal QoR improvements.

Note: This intrusive IP flow only supports Virtex-4, Virtex-II Pro, and Spartan-3 technologies when you run the Synplify Premier Graph-based Physical Synthesis feature.

The following figure shows the Xilinx netlist file added to the Project file.



Xilinx Virtex2p : XC2VP100 : FF17(

D:\NewEdifReaderExample\test_v2p

```
⊟ XilinxIP (project)
  ⊞ Verilog
  ⊞ Design Plans
  ⊞ Constraint
  ⊟ EDIF
      mac1.edn
      mac2.edn
  ⊞ test_v2p [XilinxIP.sfp]
```

Frequency (Mhz)

● 2.05

○ Auto Constrain

☑ Physical Synthesis

☑ FSM Compiler

Add Xilinx
Netlist File
(.edn)
to Project

You can run physical synthesis for the following intrusive IP flows with Xilinx .edn netlist files:

- Without a design plan file

  This intrusive IP flows lets the Synplify Premier software determine optimizations and placements for all the logic in the IP.

- With a soft region constraint for the IP

  This intrusive IP flow lets the Synplify Premier software move or tunnel logic across region boundaries to improve timing. You must create this IP region constraint before you run physical synthesis. See *Creating IP Region Constraints* on page 9-38.

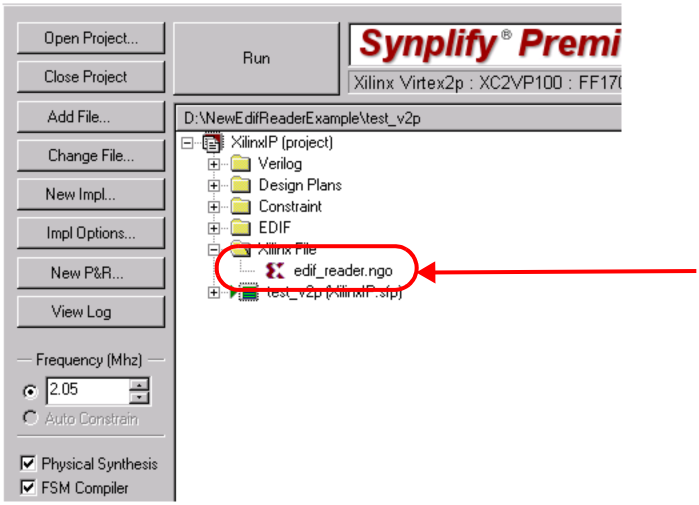
- With a hard region constraint for the IP

  This intrusive IP flow prevents the Synplify Premier software from moving or tunneling out logic inside the region boundaries to improve timing. However, logic not originally assigned to the region can be moved inside the region during optimizations. You must create this IP region constraint before you run physical synthesis. See *Creating IP Region Constraints* on page 9-38.

Macro IP Flow

The macro IP flow forces the Synplify Premier software to leave the contents of the IP untouched. Only a timing model is generated for the IP during synthesis. You must add the Xilinx netlist files (.ngc/.ngo) that include the contents of the IPs to your project. You must also include a design plan file (.sfp) which constrains the IP components to specified placement locations.

The following figure shows the Xilinx netlist file added to the Project file.



Add Xilinx Netlist File (.ngo/.ngc) to Project

When you run physical synthesis for the macro IP flow with a Xilinx `.ngc or` `.ngo` netlist file:

- Without a design plan file

  This macro IP flow is not supported for physical synthesis and will generate an error message.

- With an IP block region constraint for the IP

  This macro IP flow preserves the contents of the IP to the region boundary. This IP flow prevents the Synplify Premier software from moving or tunneling out logic outside the region boundary, as well as, placing logic outside the region into this region boundary. The Synplify Premier software preserves the hierarchy of the IP and prevents any pruning or optimizing of instances inside the IP. You must create this IP region constraint before you run physical synthesis. See *Creating IP Region Constraints* on page 9-38.

Creating IP Region Constraints

You can create region constraints for both the intrusive and macro IP flows, before you can run physical synthesis. It is recommended that you run through a first-pass logic synthesis flow to determine the size, shape, and resources required for the region constraint you need to create for physical synthesis.

Then, you can use design planning to create a region for the IP to improve QoR for the design. To do this, perform the following:

1. Open the Design Planner. To do this, you can either:
   - Click on the New Design Plan icon (▣).
   - Select File->New from the Project menu.
2. In the New dialog box, select the design plan file type and specify a Design Plan file name and file location directory to add to your project.

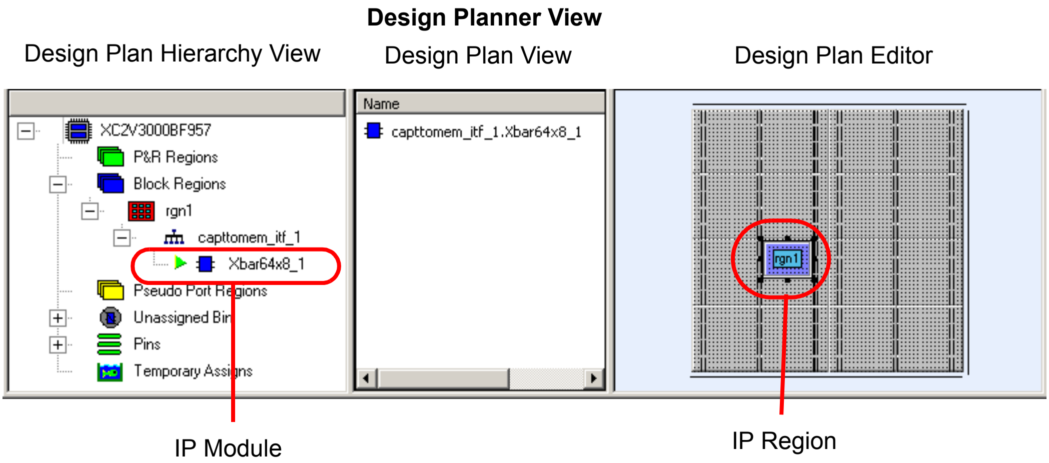
3. In the Design Plan Editor pane of the Design Planner view, create a region by doing either of the following:

   – Right-click and select Add->Block Region.

   – Use the left-mouse button to draw a region at the desired location.

   Good design planning is required to ensure better QoR improvements. Make sure to place regions so that they are near required resources, such as I/O ports or contain RAMs if necessary. Do not place regions where they might create an obstruction for the rest of the design. Design planning is an iterative process to ensure that IP regions are placed in an optimal location.

4. Assign IP logic to this region by selecting the IP module in the RTL Analyst view and dragging it to the desired location in the Design Plan Editor.



5. Specify the type of constraint to apply to this region. First select the IP region. Then, right-click and select Region Type and one of the following options:

   – Soft (Tunneling On)
   This is the default region type for the IP flows. The following Tcl command is saved to the design plan file:

   ```
   assign_property syn_rgn_tunnel_on {region_name} 1
   ```

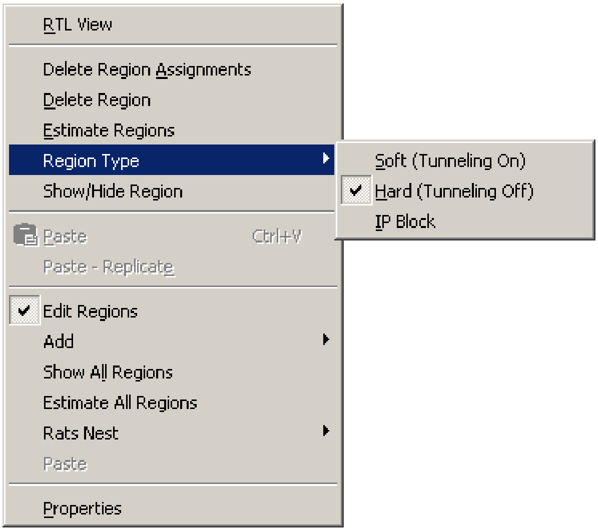
   – Hard (Tunneling Off)
   The following Tcl command is saved to the design plan file:

```
assign_property syn_rgn_tunnel_off (region_name} 1
```

– IP Block
  The following Tcl command is saved to the design plan file:

```
assign_property syn_rgn_ip_block {region_name} 1
```

See *Intrusive IP Flow* on page 9-35 and *Macro IP Flow* on page 9-37 for a description of these options.



6. Save the Design Plan file (`.sfp`).

7. Run physical synthesis.

CHAPTER 10

Design Flows and Process Optimization

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

Batch mode is a command-line mode where 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 10-4.

This section describes the following operations:

- Running Batch Mode on a Project File, next
- Running Batch Mode with a Tcl Script, on page 10-3

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 10-3.

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 10-5.

2. From a command prompt, go to the directory where the project files are located, and type one of the following, depending on which project you are using:

   ```
   synplify -batch project_file_name.prj
   synplify_pro -batch project_file_name.prj
   synplify_premier -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 UNIX systems, this is generally the monitor; on Windows systems, it is the `sdout.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 10-2.

1. Create a Tcl batch script. See *Creating a Tcl Synthesis Script,* on page 10-5 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
   ```

   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 UNIX 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:

- Crossprobing from the Tcl Script Window, next
- Using Tcl Commands and Scripts, on page 10-4
- Generating a Job Script, on page 10-5
- Creating a Tcl Synthesis Script, on page 10-5
- Using Tcl Variables to Try Different Clock Frequencies, on page 10-7
- Using Tcl Variables to Try Several Target Technologies
- Running Bottom-up Synthesis with a Script, on page 10-9

You can also use synhooks Tcl scripts, as described in *Automating Flows with synhooks.tcl*, on page 10-10.

Crossprobing from the Tcl Script Window

To crossprobe from the Tcl Script window (not a Synplify feature) to the source code, click on the Warnings tab and then double-click a line in the Tcl window. The software opens the relevant source code file and highlights the corresponding code. Crossprobing from the Tcl script window is useful for debugging error messages.

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` *<command_keyword>* in the Tcl window to see the syntax for the command.

2. To run a Tcl script, do the following:

   - Create a Tcl script. Refer to *Generating a Job Script,* on page 10-5 and *Creating a Tcl Synthesis Script,* on page 10-5.

   - Run the Tcl script by either typing `source` *Tcl_scriptfile* in the Tcl script window, or selecting File -> Run Tcl Script, selecting the Tcl file and clicking Open.

   The software runs the selected script and executes the commands in it. 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 this text in the Tcl window and paste it into a text file that you build to run as a Tcl script. For example:

```
add_file -verilog "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 from scratch.

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 `load` *`project`*`.prj`.

3. Add files. This may not be needed for an existing project.

   – Add source files with `add_file -vhdl` or `add_file -verilog`. Make sure the top-level file is last:

   ```
   add_file -vhdl "statemach.vhd"
   add_file -vhdl "rotate.vhd"
   add_file -vhdl "memory.vhd"
   add_file -vhdl "top_level.vhd"
   ```

   – Add constraint files with constraints and vendor-specific attributes. See *Using a Text Editor for Constraint Files*, on page 3-62 for details about this file.

   ```
   add_file -constraint "design.sdc"
   ```

4. Set the design synthesis controls and the output:

   – Set vendor-specific `set_option` controls as needed. See the appropriate vendor chapter in the *Reference Manual* for details.

   ```
   set_option -technology VIRTEX2
   set_option -part XC2V40
   set_option -package CS144
   set_option -speed_grade -6
   ```

   – Use the `set_option` command for implementation options.

   ```
   set_option -symbolic_fsm_compiler true
   set_option -frequency 30.0
   ```

   – Set the output file information with `project -result_file` and `project -log_file`.

5. Set the file and run options:

   – Save the project with `project -save`.

   – Run the project with `project -run`.

   – Open the RTL and Technology views:

```
open_file -rtl_view
open_file -technology_view
```

6. Check the syntax.

   – Check case, because Tcl is case-sensitive.

   – Start all comments with a hash mark (#).

   – Enclose all pathnames and filenames in double quotes.

   – Always use a forward slash (/) in directory and pathnames, even on the PC.

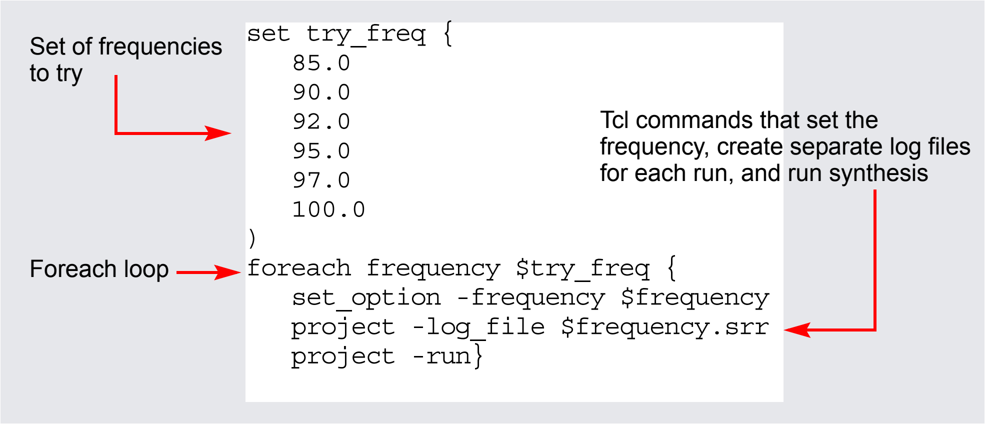
Using Tcl Variables to Try Different Clock Frequencies

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.



```
set try_freq {
   85.0
   90.0
   92.0
   95.0
   97.0
   100.0
)
foreach frequency $try_freq {
   set_option -frequency $frequency
   project -log_file $frequency.srr
   project -run}
```

Set of frequencies to try
Tcl commands that set the frequency, create separate log files for each run, and run synthesis
Foreach loop

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 10-7. 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 {

         ISPGDX APEX20K 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/simpletest/prep2_2.v"

# Create the Tcl variable to try different target technologies.
   set try_these
      ISPGDX APEX20K 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 10-5 for details.
2. Create a top-level script that reads the block scripts. Create the script with the with `project -new` command.
3. Add the top-level data:
   - Add source files with `add_file -vhdl` or `add_file -verilog`.
   - Add constraint files with `add_file -constraint`.
   - Set the top-level options with `set_option`.
   - Set the output file information with `project -result_file` and `project -log_file`.
   - Save the project with `project -save`.
   - Run the project with `project -run`.
4. Save the top-level script, and then run it using this syntax:

   source “*block_script*.tcl”

When you run this, 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 synhooks Script,* on page 4-14), 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 Synplicity <*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 5-58 in the *Reference Manual*.

Design Phase	Tcl Callback Function
Project Setup Callbacks	
Settings defaults for projects	proc syn_on_set_project_template
Creating projects	proc syn_on_new_project
Opening projects	proc syn_on_open_project
Closing projects	proc syn_on_close_project
Application Callbacks	
Starting the application after opening a project	proc syn_on_start_application
Exiting the application	proc syn_on_exit_application
Run Callbacks	
Starting a run. See *Example: proc syn_on_start_run*, on page 10-12.	proc syn_on_start_run
Ending a run	proc syn_on_end_run
Key Assignment Callbacks	
Setting an operation for Ctrl-F8. See *Example: proc syn_on_press_ctrl_f8*, on page 10-12.	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]
        }
}
```

The VIF Formal Verification Flow

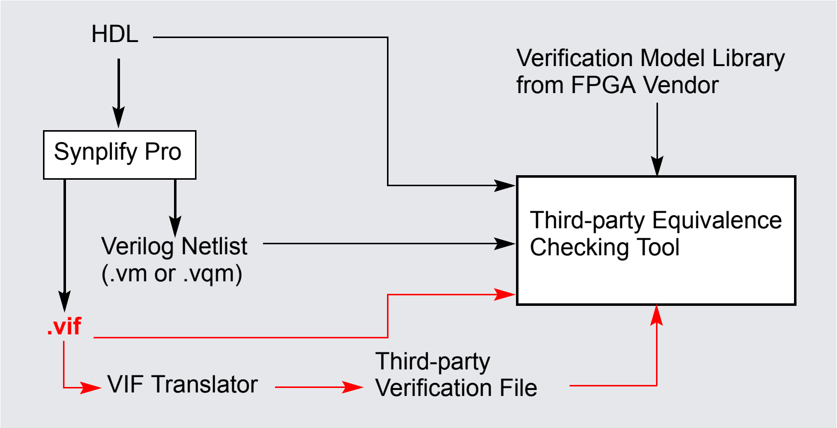
During synthesis, the Synplify Pro tool performs several sequential optimizations and design transformations to improve delay and area. These transformations make it difficult for a formal verification tool to match registers in the result netlist with the corresponding registers in the source HDL (a 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, next
- Generating a VIF File, on page 10-14
- Generating a VIF File, on page 10-14
- Using a Tcl Script for VIF Conversion, on page 10-16
- Handling Equivalency Check Failures, on page 10-18

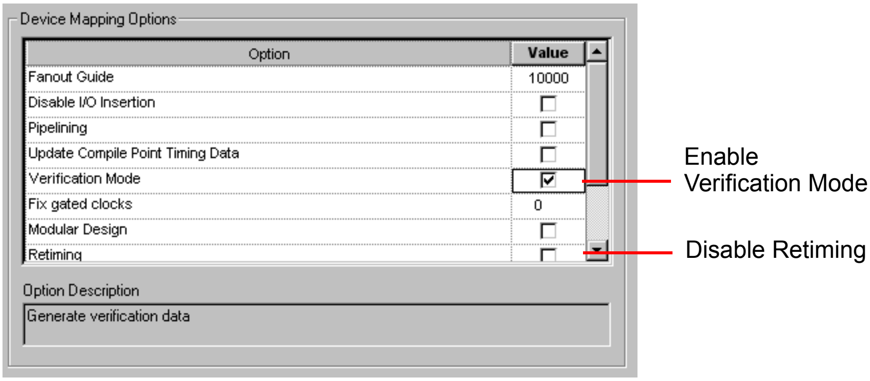
Overview of the VIF Flow

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:



Device Mapping Options

Option	Value
Fanout Guide	10000
Disable I/O Insertion	□
Pipelining	□
Update Compile Point Timing Data	□
Verification Mode	☑
Fix gated clocks	0
Modular Design	□
Retiming	□

Option Description

Generate verification data

Enable Verification Mode

Disable Retiming

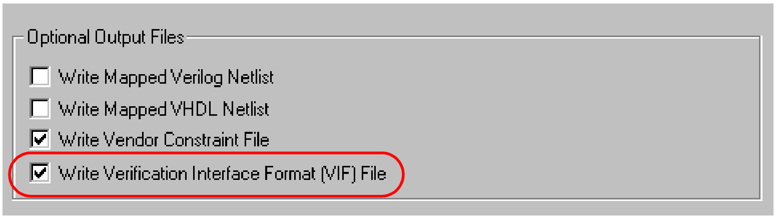
Figure 10-1: Device options for generating VIF output

- 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.



Optional Output Files

☐ Write Mapped Verilog Netlist
☐ Write Mapped VHDL Netlist
☑ Write Vendor Constraint File
☑ Write Verification Interface Format (VIF) File

Figure 10-2: Implementation Results Options for Generating VIF Output

Note: 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 5-62* 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

Optimization	VIF Command
Register replication	vif_set_equiv
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

5. 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 10-16 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.
6. 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.

Manual VIF Conversion

The following procedure describes how to convert `.vif` files manually.

1. Source the vif2conformal.tcl file by typing one of the following commands in the Synplify Pro Tcl window:

   ```
   source <synplify pro install>/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.

Automated VIF Conversion with Synhooks

You can create a script using the synhooks.tcl file (see *Automating Flows with synhooks.tcl*, on page 10-10) to automate the generation of verification files. A synhooks Tcl script example, synhooks_for_vif2conformal.tcl, has been provided and is located in the *install_dir*/examples directory.

The synhooks_for_vif2conformal.tcl Tcl script example 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:

1. 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**

2. Source the synhooks_for_vif2conformal.tcl file in the Synplify Pro Tcl window to setup automatic conversion.

   For example:

   % **source** *install_dir***/examples/synhooks_for_vif2conformal.tcl**

Note: For the second method, you will have to source the synhooks_for_vif2conformal.tcl file every time a new project is started, or 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.

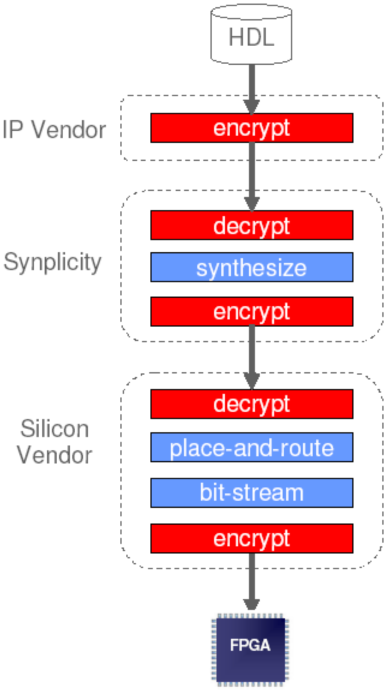
- Check the log file report and fix the errors reported.
- Check the optimization mapping in the vif file. See step 4 of *Overview of the VIF Flow,* on page 10-13 for a list of commands.

Protected Flow Support

Synplicity's protected flow specifies a secured flow for IP implementations. The IP protection is achieved through the use of encryption/decryption technology for the HDL (IP vendor), netlist, and bit-stream protection (Silicon vendor).

IP Flow Diagram

The following diagram illustrates the IP flow for FPGAs using Synplicity software.



Using IP in a Design

To use an IP in a design, do the following

1. Add the vendor-encryted IP with your other files in the project.

   The encrypted files must be encrypted by the vendor in the vendor format, not encrypted by the user.

2. Synthesize the design.

   The tool decrypts the IP and synthesizes it. The software then re-encrypts the IP file for the place-and-route tool. For Lattice technologies, the encryption can be decrypted by the place-and-route tool. For all other technologies, the place-and-route tool cannot decrypt the encrypted IP in the output netlist. The simulation netlists produced after synthesis do not contain the details of the IP. You need the netlist from the vendor to simulate the IP.

3. Analyze the design.

   The software treats the IP as a black box in HDL Analyst views. The RTL and Technology HDL Analyst tools do not allow you to view or push/pop into encrypted code.

Running Place-and-Route After Synthesis

For Altera and Xilinx technologies, you can automatically create a place-and-route implementation, and 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.

This section describes the following:

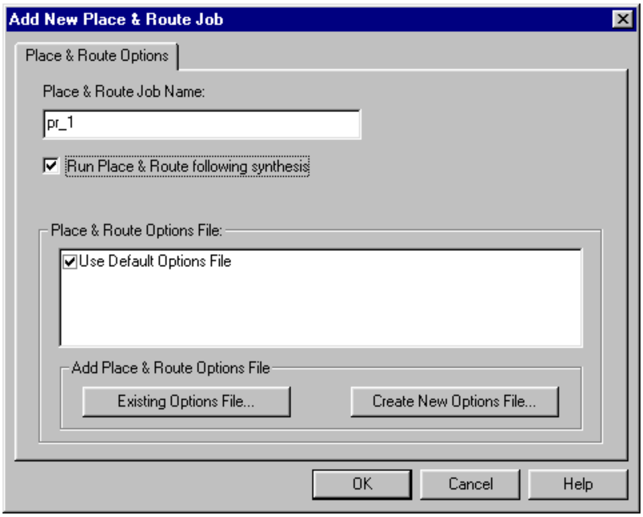
- Creating and Running P&R Projects, next
- Specifying Xilinx Place-and-Route Options, on page 10-23
- Backannotating Place-and-Route Data, on page 10-25
- Analyzing Physical Synthesis (Synplify Premier), on page 10-26

Creating and Running P&R Projects

For Altera and Xilinx technologies, the Synplify Pro and Synplify Premier tools automatically create a place-and-route implementation after the synthesis run is complete. The following steps show you how to create a new place-and-route implementation manually.

1. To create a new place-and-route implementation do one of the following:
   - Click on the New P& R... button from the Project view.
   - Select a synthesis implementation, then right-click and select Add New Place & Route Job from the popup menu.

   The Add New Place and Route Job dialog box opens. The available options vary slightly depending on the synthesis tool you are using and the chosen technology. See *Running Physical Synthesis, on page 11-9* for a description of the Synplify Premier place-and-route flow.



Add New Place & Route Job
Place & Route Options
Place & Route Options File:
Add Place & Route Options File
Existing Options File...
Create New Options File...
OK
Cancel
Help

Place & Route Job Name: pr_1

☑ Run Place & Route following synthesis

☑ Use Default Options File

2. Type a name for the place-and-route implementation in Place & Route Flow Name. A default place-and-route name appears in the display. Avoid using spaces in the implementation name.

3. For Xilinx users, select a place-and-route options file. See *Specifying Xilinx Place-and-Route Options*, on page 10-23 for details.

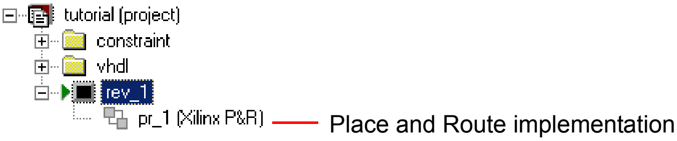
4. For Synplify Premier users, you can choose to backannotate data. See *Backannotating Place-and-Route Data*, on page 10-25 for details.

5. Enable the Run Place & Route following synthesis option and click OK.

   The application creates a place-and-route implementation under the current synthesis implementation in the Project view. 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.

6. Click the current implementation in the Project view to see the place-and-route implementation.



```
tutorial (project)
    constraint
    vhdl
    rev_1
        pr_1 (Xilinx P&R)
```

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 from the popup menu. You can repeat the preceding steps to add as many P&R implementations as you need.

7. Synthesize the design. You can either

   – Press the Run button.

   – Right-click 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, then running place-and-route invokes synthesis as well. After synthesis, the software automatically runs the place-and-route tool. If you have a Xilinx design and specified an options file, the software uses the options to run place-and-route.

8. To run in batch mode, do the following:

   – Create a place-and-route implementation, as described previously.

   – Use the `-run all` command to synthesize the design and then place and route. If the synthesis implementation is selected the software only runs synthesis; you must run place-and-route separately.

Specifying Xilinx Place-and-Route Options

This section shows you how to customize your Xilinx place-and-route run by specifying a place-and-route options file or xflow script. You can either use the default or specify a custom file.

1. To use the default place-and-route options, click the New P&R button in the Project view and select Use Default Options File in the dialog box. Click OK.

   The software uses the default Xilinx place-and-route options located in the xilinx_par.opt file when it runs place-and-route.

2. To use an existing options file (xflow script), do the following:

   – Click the New P&R 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.

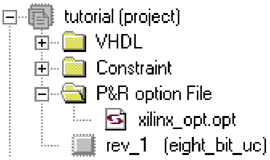
If you want to customize this file, edit the default file. The software uses the options in this file to place and route the design after synthesis. You can now view the results, as described in step 4.

3. To create a new Xilinx place-and-route options file, do either of the following:

   - Click the New P&R 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 Open.

     Alternatively, select File->New. Set the file type to Xilinx Option File. Type a file name; for example, design_par.opt. Enable the Add to Project option. Click OK.

     A text window opens with the file. The software creates an options file (.opt) with the default options and adds it to the project. The Project view displays this file. You can now edit this file to customize it.



   - Customize the options file by editing it. 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.

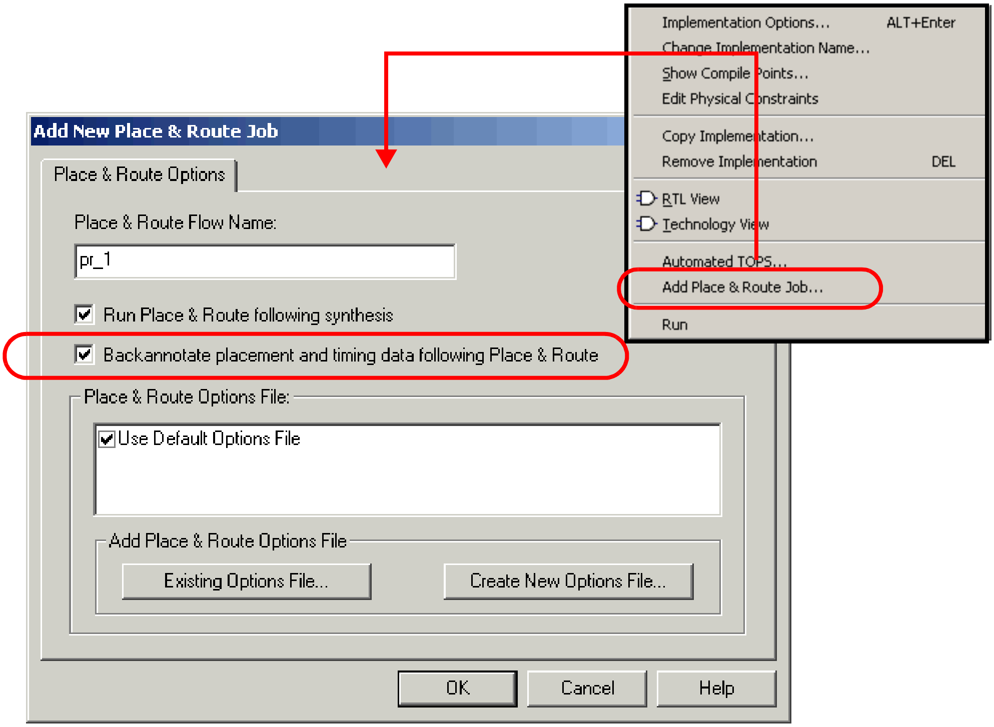
4. View the results.

   - Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
   - View the log file xflow.log for information about the run.

Backannotating Place-and-Route Data

In the Synplify Premier tool, you can also choose to back annotate place-and-route data which provides accurate timing and placement information during physical synthesis. To do this:

- Click on the New P&R... button from the Project view. On the Add New Place & Route Job dialog box, enable the Backannotate placement and timing data following Place & Route option.
- Select a synthesis implementation, then right-click and select Place & Route Options from the popup menu. Enable the Back Annotate option from the popup dialog box.


Implementation Options... ALT+Enter
Change Implementation Name...
Show Compile Points...
Edit Physical Constraints
Copy Implementation...
Remove Implementation DEL
RTL View
Technology View
Automated TOPS...
Add Place & Route Job...
Run
Add New Place & Route Job
Place & Route Options
Place & Route Options File:
Add Place & Route Options File
Existing Options File...
Create New Options File...
OK
Cancel
Help

Place & Route Flow Name: pr_1

☑ Run Place & Route following synthesis

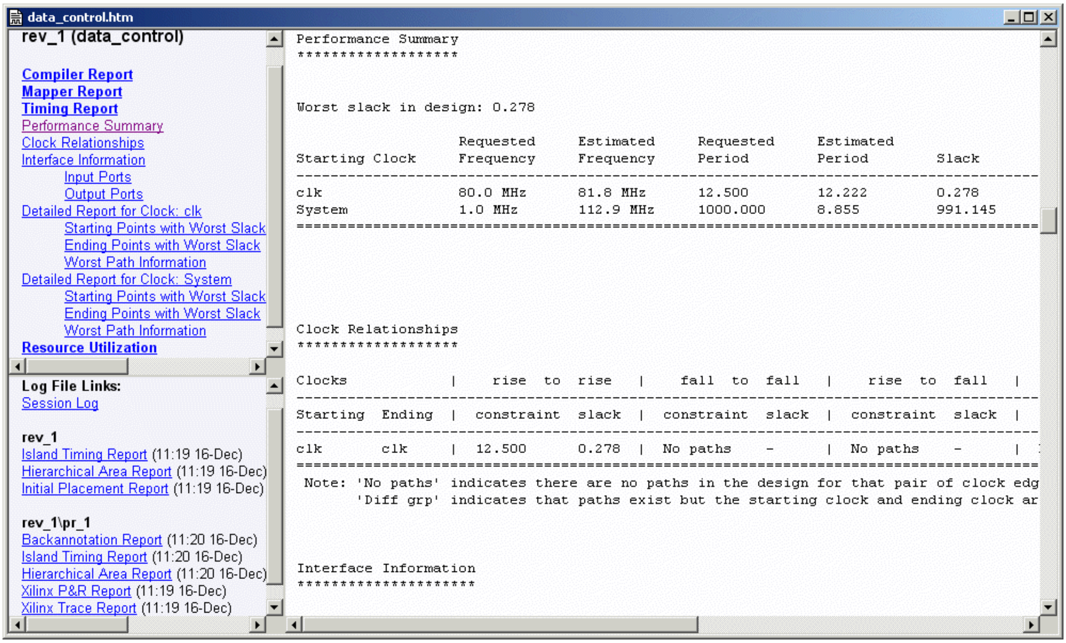
☑ Backannotate placement and timing data following Place & Route

☑ Use Default Options File

However, this option is only applicable for certain Altera and Xilinx technologies. See *Device Support for the Physical Synthesis Flows, on page 11-3*.

Analyzing Physical Synthesis (Synplify Premier)

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.


data_control.htm
rev_1 (data_control)
Compiler Report
Mapper Report
Timing Report
Performance Summary
Clock Relationships
Interface Information
Input Ports
Output Ports
Detailed Report for Clock: clk
Starting Points with Worst Slack
Ending Points with Worst Slack
Worst Path Information
Detailed Report for Clock: System
Starting Points with Worst Slack
Ending Points with Worst Slack
Worst Path Information
Resource Utilization
Log File Links:
Session Log
rev_1
Island Timing Report (11:19 16-Dec)
Hierarchical Area Report (11:19 16-Dec)
Initial Placement Report (11:19 16-Dec)
rev_1\pr_1
Backannotation Report (11:20 16-Dec)
Island Timing Report (11:20 16-Dec)
Hierarchical Area Report (11:20 16-Dec)
Xilinx P&R Report (11:19 16-Dec)
Xilinx Trace Report (11:19 16-Dec)

```
Performance Summary
*******************


Worst slack in design: 0.278

                   Requested     Estimated     Requested     Estimated
Starting Clock     Frequency     Frequency     Period        Period        Slack
------------------------------------------------------------------------------------
clk                80.0 MHz      81.8 MHz      12.500        12.222        0.278
System             1.0 MHz       112.9 MHz     1000.000      8.855         991.145
====================================================================================





Clock Relationships
*******************

Clocks            |    rise  to  rise   |    fall  to  fall   |    rise  to  fall   |
------------------------------------------------------------------------------------
Starting  Ending  |  constraint  slack  |  constraint  slack  |  constraint  slack  |
------------------------------------------------------------------------------------
clk       clk     |  12.500      0.278  |  No paths    -      |  No paths    -      |
====================================================================================
 Note: 'No paths' indicates there are no paths in the design for that pair of clock edg
       'Diff grp' indicates that paths exist but the starting clock and ending clock ar



Interface Information
*********************
```

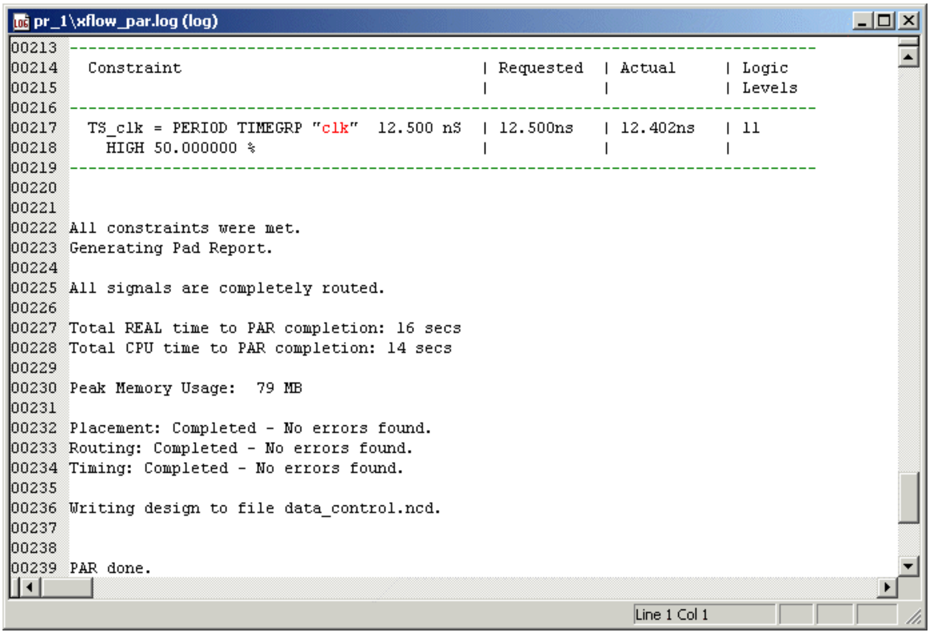
See *Viewing the Log File*, on page 4-2 for complete information on how to interpret the log file results.

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:

- Analyzing Timing, on page 4-73
- The Island Timing Report, on page 4-83

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 Full Chip

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.



```
--------------------------------------------------------------------------------
  Constraint                                | Requested  | Actual     | Logic
                                            |            |            | Levels
--------------------------------------------------------------------------------
  TS_clk = PERIOD TIMEGRP "clk"  12.500 nS  | 12.500ns   | 12.402ns   | 11
    HIGH 50.000000 %                        |            |            |
--------------------------------------------------------------------------------


All constraints were met.
Generating Pad Report.

All signals are completely routed.

Total REAL time to PAR completion: 16 secs
Total CPU time to PAR completion: 14 secs

Peak Memory Usage:  79 MB

Placement: Completed - No errors found.
Routing: Completed - No errors found.
Timing: Completed - No errors found.

Writing design to file data_control.ncd.


PAR done.
```

For more information on analyzing synthesis results graphically, see the following topics:

- Synplify Premier Physical Analyst Tool, on page 5-2
- Analyzing With the HDL Analyst Tool, on page 4-56

MultiPoint Synthesis

This document describes the MultiPoint™ synthesis flow, which automates the traditional bottom-up flow for large designs. This feature is available with the Synplify Pro product, for use with certain technology families.

- Traditional Bottom-up Design and MultiPoint Synthesis, on page 10-28
- The Synplify Pro MultiPoint Synthesis Flow, on page 10-29

For information about technology-specific flows, see *The Altera LogicLock Flow*, on page 10-39 and *Using the Xilinx Modular Flow*, on page 10-54

Traditional Bottom-up Design and MultiPoint Synthesis

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 run time limits. The bottom-up flow permits partial recompiles and multiprocessing to speed up design compilation.

For certain device technologies, the Synplicity MultiPoint™ synthesis flow lets you design incrementally and synthesize designs that exceed runtime limits for top-down synthesis. For details of these flows, see *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29 for a generic flow diagram. *The Altera LogicLock Flow*, on page 10-39, and *Using the Xilinx Modular Flow*, on page 10-54 for technology-specific flows.

Traditionally, bottom-up design requires designers to write and maintain time-consuming and error-prone scripts that direct synthesis and keep track of design dependencies. The MultiPoint flow automates difference-based incremental synthesis, and eliminates the need for these scripts.

The MultiPoint flow lets you break down a design into smaller synthesis units, called 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 may have any number of compile points, and compile points may be nested. See the *Reference Manual* for details about compile points.

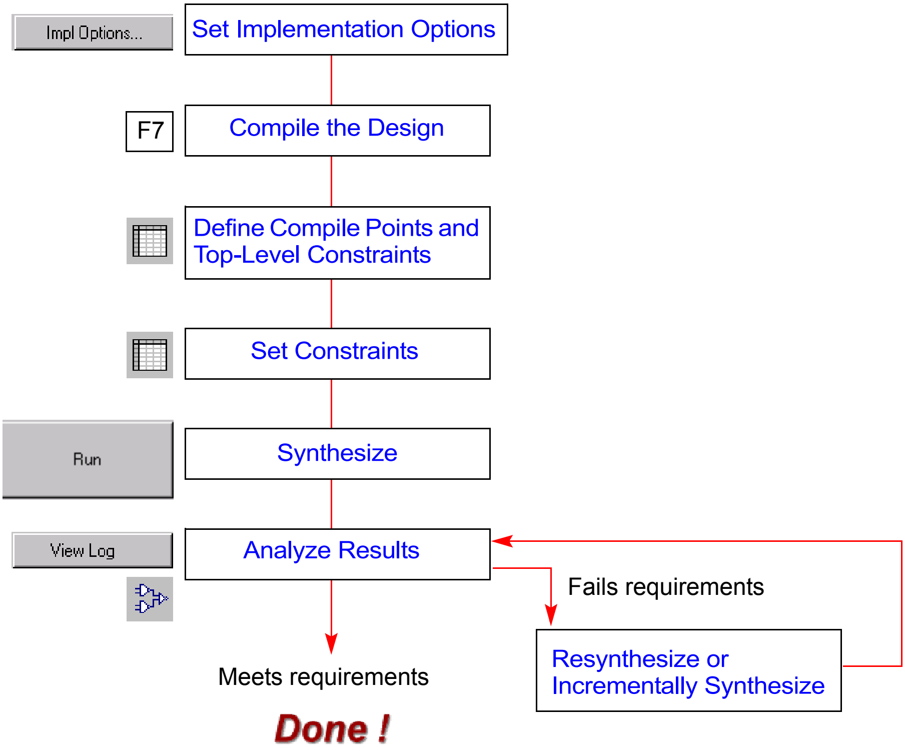
You must provide timing constraints (timing budgeting) for each compile point; the more accurate the constraints, the better your results. Constraints are not automatically budgeted, so manual time budgeting is important.

The Synplify Pro MultiPoint Synthesis Flow

This section describes the Synplify Pro MultiPoint™ Synthesis flow, which contains the following steps to automate the traditional bottom-up flow for large designs.

- Set Implementation Options, on page 10-30
- Compile the Design, on page 10-31
- Set Implementation Options, on page 10-30
- Define Compile Points and Top-Level Constraints, on page 10-31
- Set Constraints, on page 10-33
- Synthesize, on page 10-36
- Analyze Results, on page 10-37
- Resynthesize or Incrementally Synthesize, on page 10-38

The following figure shows the general procedure for using the Synplify Pro MultiPoint flow. For flows with vendor-specific details, see *The Altera LogicLock Flow*, on page 10-39 and *The Xilinx MultiPoint Synthesis Flow*, on page 10-48. For Actel designs, follow the generic flow.



Set Implementation Options

The first step in MultiPoint synthesis is to set the implementation options, just as with the regular design flow. Multipoint synthesis is not available with the Synplify tool.

1. Start Synplify Premier or Synplify Pro, set up a design project for the MultiPoint flow, and open the project for the top-level design.
2. Press the Impl Options button in the Project view to open the Implementation Options dialog box.
3. Set the following:
   - Select a technology that supports the MultiPoint flow and set the device, part and speed grade options.
   - Set the global frequency, and any other optimization options.

4. For Synplify Premier MultiPoint synthesis, also do the following:
   - In the Implementation Options dialog box, go to the tab and disable Netlist Optimization Options.
   - For Altera Stratix devices, also disable Create MAC Hierarchy.
   - Remember that Synplify Premier Multipoint synthesis ignores the following optimizations for compile points: Feedthrough Optimization, Constant Propagation, and Create Always/Process Level Hierarchy.

   You are now ready to compile the design (*Compile the Design*, on page 10-31).

Compile the Design

After setting the implementation options, you must compile the design. This is the second step in the *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29. The Synplify tool does not support multipoint synthesis.

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.

   If you are a Synplify Pro user, the next step is to define compile points (*Define Compile Points and Top-Level Constraints*, on page 10-31).

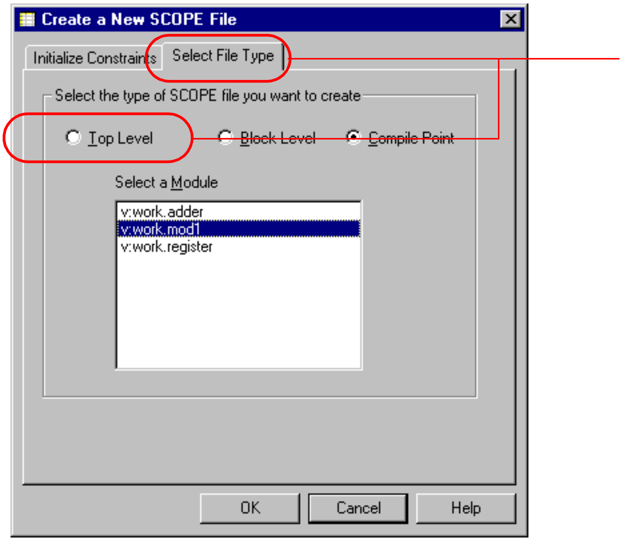
Define Compile Points and Top-Level Constraints

This is a step in *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29. The Synplify tool does not support multipoint synthesis. 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 the *Reference Manual* 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.



Select the type of SCOPE file you want to create

○ Top Level ○ Block Level ● Compile Point

Select a Module

v:work.adder
v:work.mod1
v:work.register

Click and select.

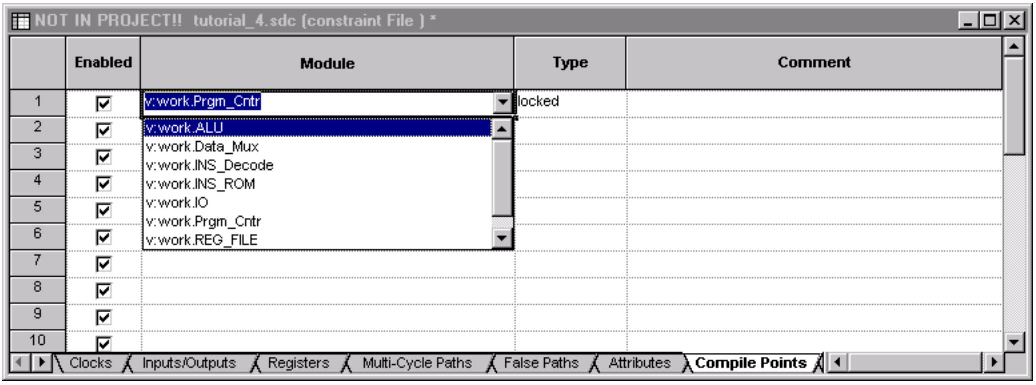
Alternatively, you can create a new top-level constraint file when you create the module-level constraint files, as described in *Create Compile Point and Top-Level Constraint Files,* on page 10-34.

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 is locked.

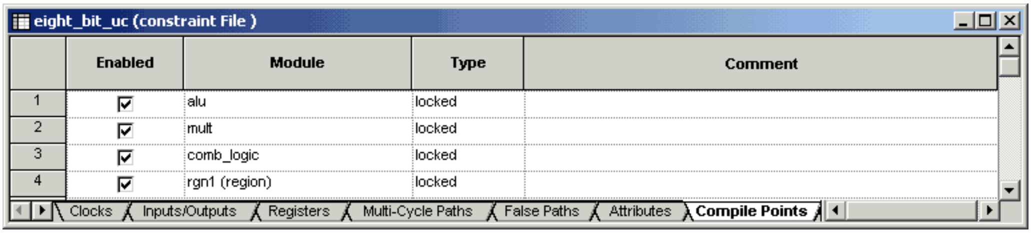
   This tags the module as a compile point. The following figure shows the the prgm_cntr module set as a compile point in the Synplify Pro flow.


NOT IN PROJECT!! tutorial_4.sdc (constraint File) *

	Enabled	Module	Type	Comment
1	☑	v:work.Prgm_Cntr	locked	
2	☑			
3	☑			
4	☑			
5	☑			
6	☑			
7	☑			
8	☑			
9	☑			
10	☑			

v:work.ALU
v:work.Data_Mux
v:work.INS_Decode
v:work.INS_ROM
v:work.IO
v:work.Prgm_Cntr
v:work.REG_FILE

The next figure shows the rgn1 (region) module set as a compile point in the Synplify Premier Multipoint flow.


eight_bit_uc (constraint File)

	Enabled	Module	Type	Comment
1	☑	alu	locked	
2	☑	mult	locked	
3	☑	comb_logic	locked	
4	☑	rgn1 (region)	locked	

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 10-33.

Set Constraints

This is a step in the *The Synplify Pro MultiPoint Synthesis Flow, on* page 10-29. You must 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 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.

See the following sections for details about compile point constraints:

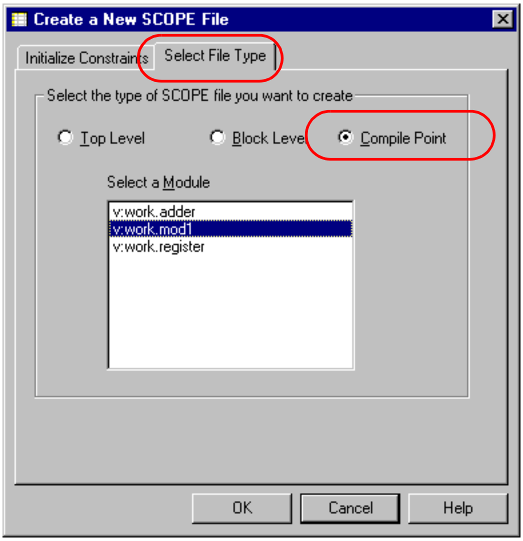
- Create Compile Point and Top-Level Constraint Files, next
- Set Compile Point Constraints, on page 10-35

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.



Select the type of SCOPE file you want to create

○ Top Level ○ Block Level ● Compile Point

Select a Module

v:work.adder
v:work.mod1
v:work.register

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 10-31.

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.

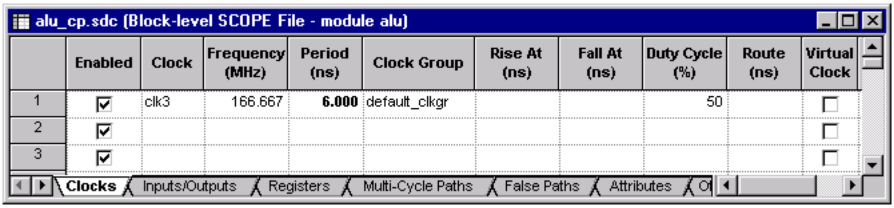
   Two SCOPE windows open, one for the top-level and one for the compile point constraint file. You must define constraints for both the top-level and the compile point. See *Set Compile Point Constraints*, on page 10-35 for details about setting compile point constraints. Set top-level constraints as in a normal design flow.

Set Compile Point Constraints

To create or modify the Synplify Premier or Synplify Pro 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.



alu_cp.sdc (Block-level SCOPE File - module alu)

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock
1	☑	clk3	166.667	**6.000**	default_clkgr			50		□
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 `cp_name_number.sdc` to the current directory.

Synthesize

This is a step in *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29 . The Synplify tool does not support MultiPoint synthesis. 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

This is a step in *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29. Multipoint synthesis is not supported in the Synplify tool. 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 in *The Synplify Pro MultiPoint Synthesis Flow, on* page 10-29. Multipoint synthesis is not supported in the Synplify tool. 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 Impl 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, select click Impl Options and enable Update Compile Point Timing Data. Click Run.

   The software regenerates the model file for each compile point when it synthesizes the compile points. The new model file is used to synthesize the parent. The option remains in effect until you disable it.
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.

The Altera LogicLock Flow

You can use Synplify Pro and Synplify Premier MultiPoint synthesis in conjunction with Altera's LogicLock methodology to design and lock down a design, one section at a time. The design team can work on individual modules separately and concurrently, and then integrate them into the top-level design. The following Synplify Pro and Synplify Premier procedures are specific to Altera; for a general description of how to use the MultiPoint flow, see *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29.

- Using Synplify Pro With the Altera LogicLock Flow
- Using Synplify Premier With the Altera LogicLock Flow

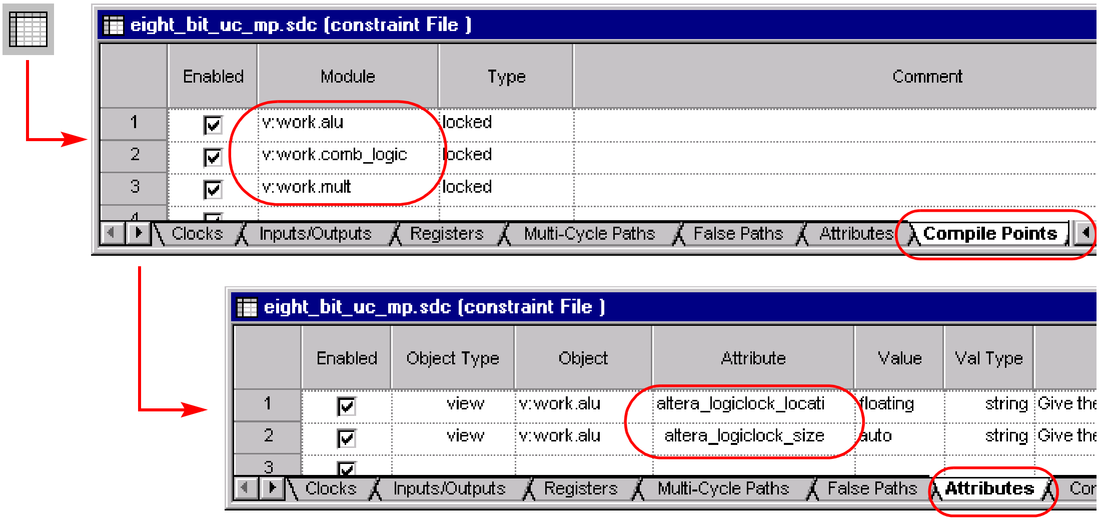
Using Synplify Pro With the Altera LogicLock Flow

To implement Synplify Pro MultiPoint synthesis with the Altera LogicLock flow, perform the steps in the following procedure:

1. Set up a project file as usual. In particular, do the following:
   - Set the target device to one of the Altera families that use LogicLock: Apex, ApexII, Mercury, Excalibur, or Stratix.
   - Set the implementation options. On the Implementation Results tab, make sure to enable the Write Mapped Verilog/VHDL Netlist option.
   - Compile the design.
2. Define compile points in the top-level `.sdc` file.
   - Click the Compile Points tab, and set compile points. A compile point is a module that is treated as a block for incremental synthesis In subsequent synthesis iterations, the software does not resynthesize the compile point unless the hierarchical interface changes. See the

Reference Manual for details about compile points. The following example shows three compile points set: ALU, comb_logic, and mult.

– Click the Attributes tab and set the altera_logiclock_location and altera_logiclock_size attributes. The following figure shows these attributes set for the ALU compile point. Save the file.



eight_bit_uc_mp.sdc (constraint File)

	Enabled	Module	Type	Comment
1	☑	v:work.alu	locked	
2	☑	v:work.comb_logic	locked	
3	☑	v:work.mult	locked	
4	☐			

eight_bit_uc_mp.sdc (constraint File)

	Enabled	Object Type	Object	Attribute	Value	Val Type	
1	☑	view	v:work.alu	altera_logiclock_locati	floating	string	Give the
2	☑	view	v:work.alu	altera_logiclock_size	auto	string	Give the
3	☑						

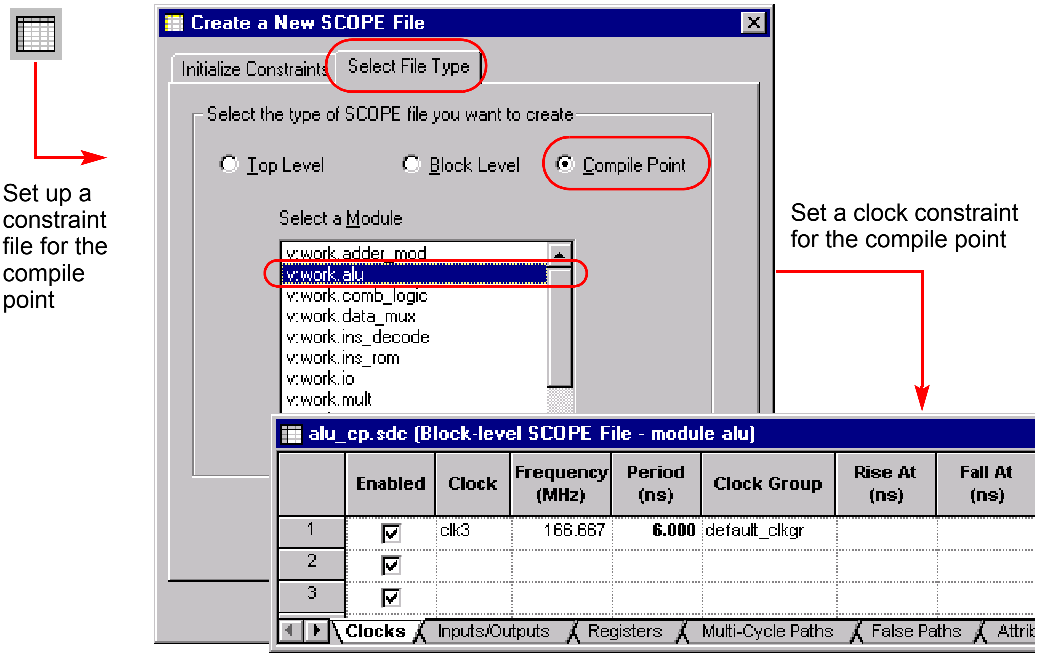
3. Create a compile point constraint file for each compile point.

   – 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 example 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.



Set up a constraint file for the compile point

Set a clock constraint for the compile point

Create a New SCOPE File

Initialize Constraints | Select File Type

Select the type of SCOPE file you want to create

○ Top Level ○ Block Level ● Compile Point

Select a Module

v:work.adder_mod
v:work.alu
v:work.comb_logic
v:work.data_mux
v:work.ins_decode
v:work.ins_rom
v:work.io
v:work.mult

alu_cp.sdc (Block-level SCOPE File - module alu)

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)
1	☑	clk3	166.667	**6.000**	default_clkgr		
2	☑						
3	☑						

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. For each compile point, the software generates a separate subdirectory with a complete set of output files. It also generates a model file that contains timing information and which is used to synthesize the next hierarchical level.

5. Place and route the design.

   You can hierarchically place and route the design, because each compile point has a separate set of output files. The Quartus software places the compile point modules you created in the LogicLock regions.

6. 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.

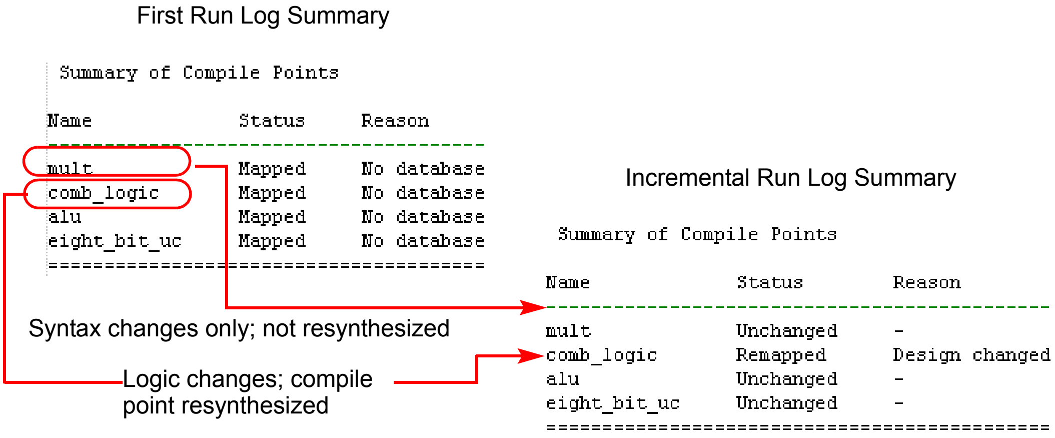
- Check that the design meets the target frequency for the design. Use the Log Watch window or check the log file.
- Open the log file and check the following:
  - Check top-level and compile point boundary timing. You can also check this visually using the RTL and Technology view schematics. If you 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.
- 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.

7. To synthesize the design incrementally, do the following:
   - Make the design changes needed in the compile points.
   - Click Run to resynthesize your design incrementally.

   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 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.


First Run Log Summary
Incremental Run Log Summary
Syntax changes only; not resynthesized
Logic changes; compile point resynthesized

```
 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
=======================================
```

```
 Summary of Compile Points

Name              Status        Reason
---------------------------------------------
mult              Unchanged     -
comb_logic        Remapped      Design changed
alu               Unchanged     -
eight_bit_uc      Unchanged     -
=============================================
```

Using Synplify Premier With the Altera LogicLock Flow

To implement Synplify Premier MultiPoint synthesis with the Altera LogicLock flow, perform the steps in the following procedure:

1. Set up a project file as usual. In particular, do the following:
   - Set the target device to one of the Altera families that use LogicLock: Apex, ApexII, Excalibur, Mercury, Cyclone, or Stratix.
   - Set the implementation options. On the Netlist Restructure tab, make sure to disable all the Netlist Optimization Options. For Altera Stratix devices, disable the Create MAC Hierarchy option also.
   - Compile the design.
2. Create a design plan to constrain regions on the device. To do this:
   - Create a new design plan.
   - Create constraint regions on the device. Then, assign necessary logic to these regions. Run region estimation for the regions and check all DRC information.

- For regions to be defined as a compile point, enable the LogicLock option for the region.
- Save the design plan file (.sfp).

3. Run Compile Physical Hierarchy to compile the design to include regions as modules in the netlist that is created.

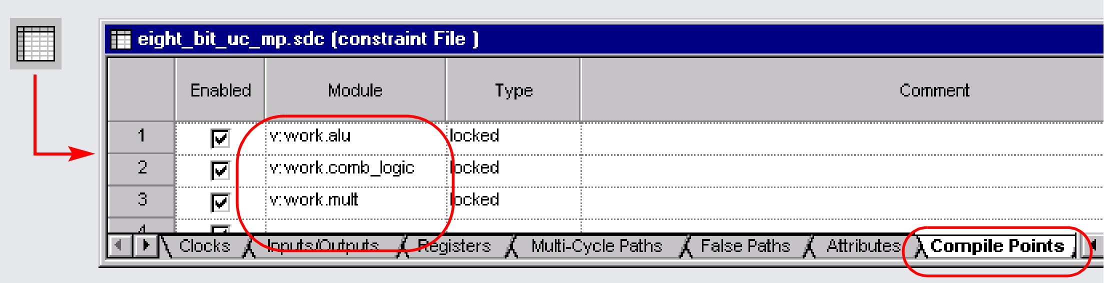
4. Define compile points and top-level constraints in the top-level .sdc file.

   - A compile point is a module or a Synplify Premier design plan region that is treated as a block for incremental synthesis. In subsequent synthesis iterations, the software does not resynthesize the compile point unless the module or region changes. Click the Compile Points tab, select the modules you want as compile points, and set Type to locked. Currently, the locked compile point model is the only one that is supported.
   - Click the Attributes tab and set the altera_logiclock_location and altera_logiclock_size attributes.

Note: Region compile points use the constraints from the deisgn plan (.sfp) file. Module compile attributes are ignored from the .sdc file.

- Define the constraints for the design, as usual.
- Save the file.

The following example shows three compile points set: ALU, comb_logic, and mult.

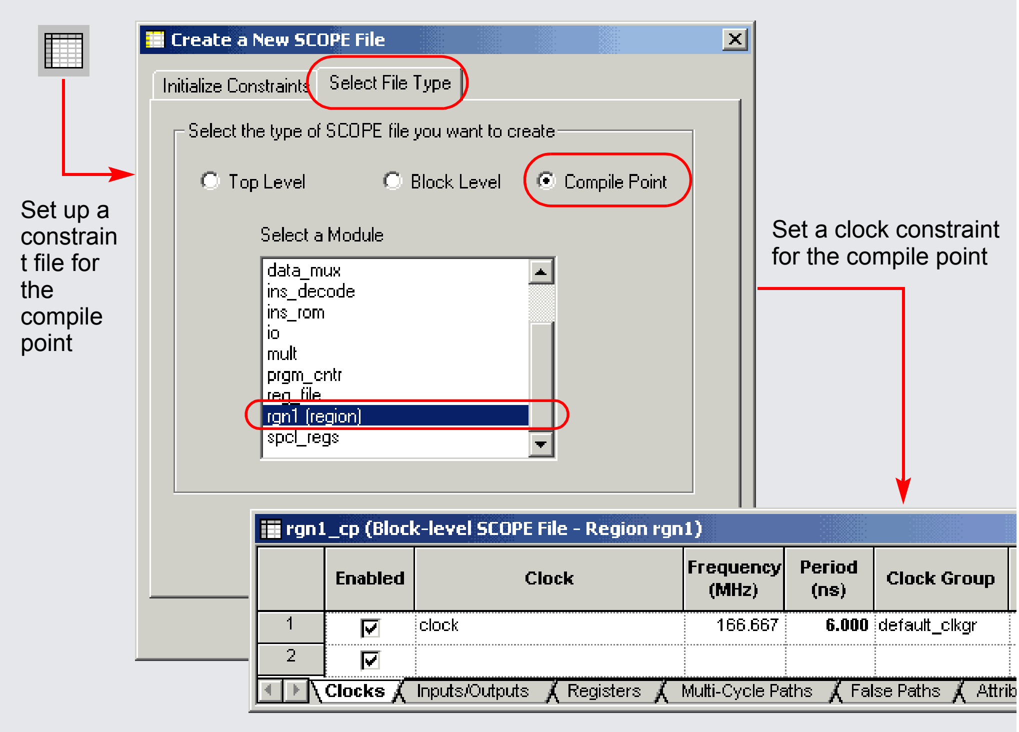

eight_bit_uc_mp.sdc (constraint File)

	Enabled	Module	Type	Comment
1	☑	v:work.alu	locked	
2	☑	v:work.comb_logic	locked	
3	☑	v:work.mult	locked	

In subsequent iterations, the software does not resynthesize the compile point unless the compile point changes.

5. Create a separate compile point constraint file for each compile point you defined.

   – Click the SCOPE icon.



Set up a constraint file for the compile point

Set a clock constraint for the compile point

Create a New SCOPE File

Select the type of SCOPE file you want to create

○ Top Level ○ Block Level ● Compile Point

Select a Module

- data_mux
- ins_decode
- ins_rom
- io
- mult
- prgm_cntr
- reg_file
- rgn1 (region)
- spcl_regs

rgn1_cp (Block-level SCOPE File - Region rgn1)

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group
1	☑	clock	166.667	**6.000**	default_clkgr
2	☑				

   – In the Create a New SCOPE File dialog box, click the Select File Type tab, then click Compile Point, and select the compile point.

   – In the next dialog box, select the top-level `.sdc` file that defines the compile points.

   – Set the clock constraint and I/O timing for the compile point. This can be the same as the top level.

   – Click the Attributes tab and type in the syn_allowed_resource attribute for the compile point (the object type is view). Remember that allowed resources set for a lower level count as part of the resources for the higher level. Save the file.

Note: Set the syn_allowed_resource attribute for module compile points only. Region compile points use the constraints from the design plan (`.sfp`) file.

6. 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. It generates netlists and a model file for each compile point, and stores these files in subdirectories named after the compile point. It also generates a model file that contains timing information and which is used to synthesize the next hierarchical level.

7. Place and route the design.

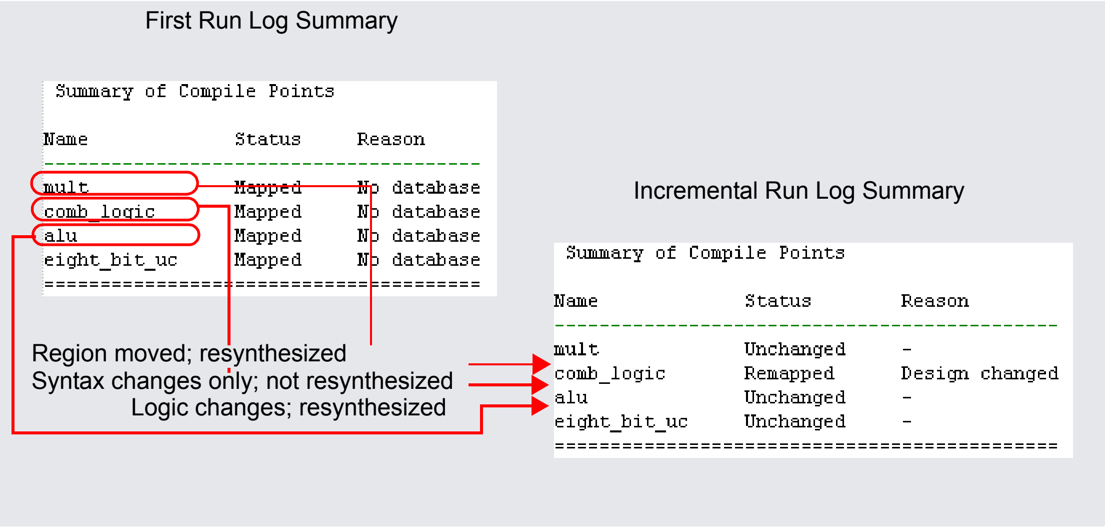
   You can hierarchically place and route the design, because each designated LogicLock point has a separate`.vqm` file. The Quartus software places the compile point modules you created in the LogicLock regions.

8. To synthesize the design incrementally, do the following:

   – Make the design changes needed.

   – Click Run to resynthesize your design incrementally.

     For an incremental run, the software only resynthesizes compile points whose logic, implementation options, or timing constraints have changed. The following figure compares 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. 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.


First Run Log Summary
Incremental Run Log Summary
Region moved; resynthesized
Syntax changes only; not resynthesized
Logic changes; resynthesized

```
 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
==============================================
```

```
 Summary of Compile Points

Name                 Status           Reason
-----------------------------------------------------
mult                 Unchanged        -
comb_logic           Remapped         Design changed
alu                  Unchanged        -
eight_bit_uc         Unchanged        -
=====================================================
```

– To resynthesize all compile points, whether or not they have changed, select Run-> Resynthesize All. If a compile point interface has not changed, the model file is not regenerated even though the compile point is resynthesized. The information in the old model file is used to synthesize the next level.

– To force a new model file to be generated, click Impl Options and enable Update Compile Point Timing Data.

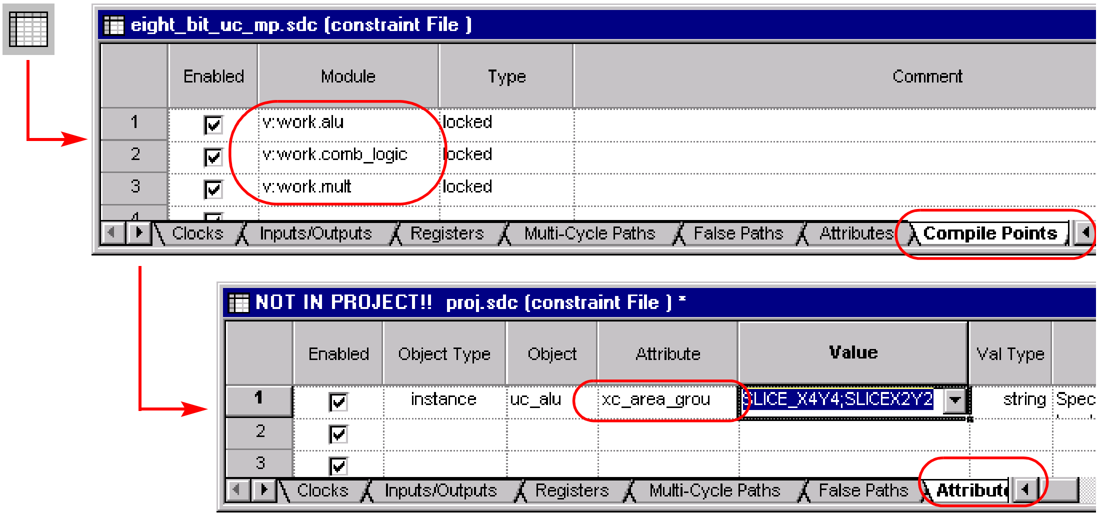
The Xilinx MultiPoint Synthesis Flow

You can use the Synplify Pro MultiPoint synthesis in conjunction with the Xilinx place-and-route tool to design and lock down a design, one section at a time. The design team can work on individual modules separately and concurrently, and then integrate them into the top-level design. The following Synplify Pro procedures are specific to Xilinx; for a general procedure on how to use the MultiPoint flow, see *The Synplify Pro MultiPoint Synthesis Flow*, on page 10-29 .

Using Synplify Pro With Xilinx MultiPoint Synthesis

To implement Synplify Pro MultiPoint synthesis with the Xilinx MultiPoint Synthesis flow, perform the steps in the following procedure:

1. Set up a project, set implementation options, and compile the project
   - Set up a project as usual, select the Xilinx target device, and set the implementation 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.
   - Click the Attributes tab. Set Object Type to instance, set Object to the compile point, and set the xc_area_group attribute to define the region. The following figure shows the attribute set for the ALU compile point. Save the file.



eight_bit_uc_mp.sdc (constraint File)

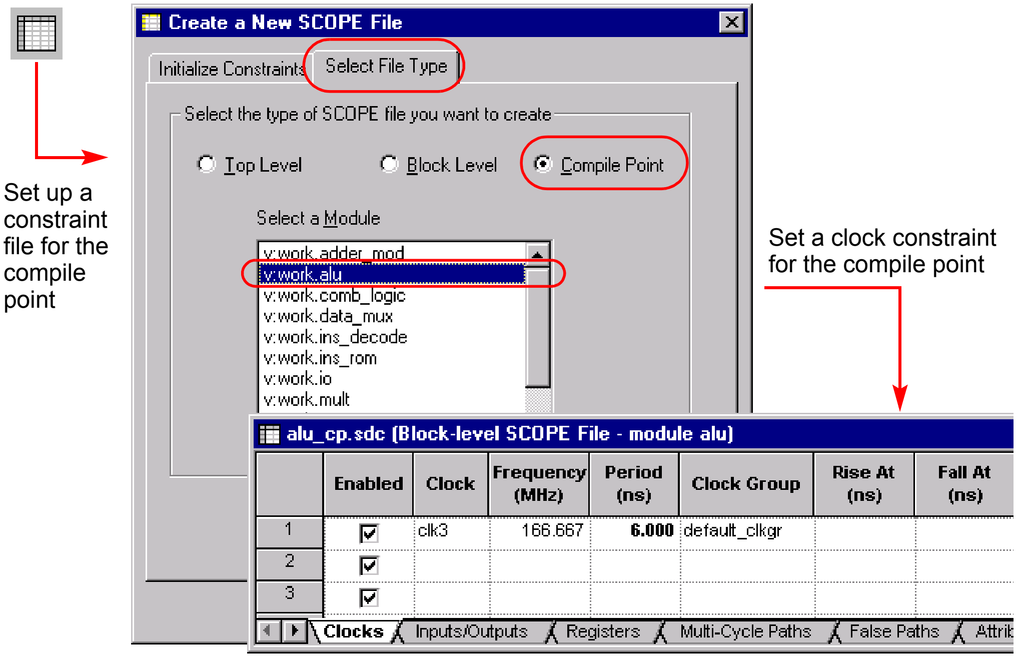
	Enabled	Module	Type	Comment
1	☑	v:work.alu	locked	
2	☑	v:work.comb_logic	locked	
3	☑	v:work.mult	locked	

NOT IN PROJECT!! proj.sdc (constraint File) *

	Enabled	Object Type	Object	Attribute	Value	Val Type	
1	☑	instance	uc_alu	xc_area_grou	SLICE_X4Y4;SLICEX2Y2	string	Spec
2	☑						
3	☑						

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.



Set up a constraint file for the compile point

Set a clock constraint for the compile point

Create a New SCOPE File

Initialize Constraints | Select File Type

Select the type of SCOPE file you want to create

○ Top Level ○ Block Level ● Compile Point

Select a Module

v:work.adder_mod
v:work.alu
v:work.comb_logic
v:work.data_mux
v:work.ins_decode
v:work.ins_rom
v:work.io
v:work.mult

alu_cp.sdc (Block-level SCOPE File - module alu)

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)
1	☑	clk3	166.667	**6.000**	default_clkgr		
2	☑						
3	☑						

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.

   The place-and-route software places the compile point modules in the regions you defined with the xc_area_group attribute.

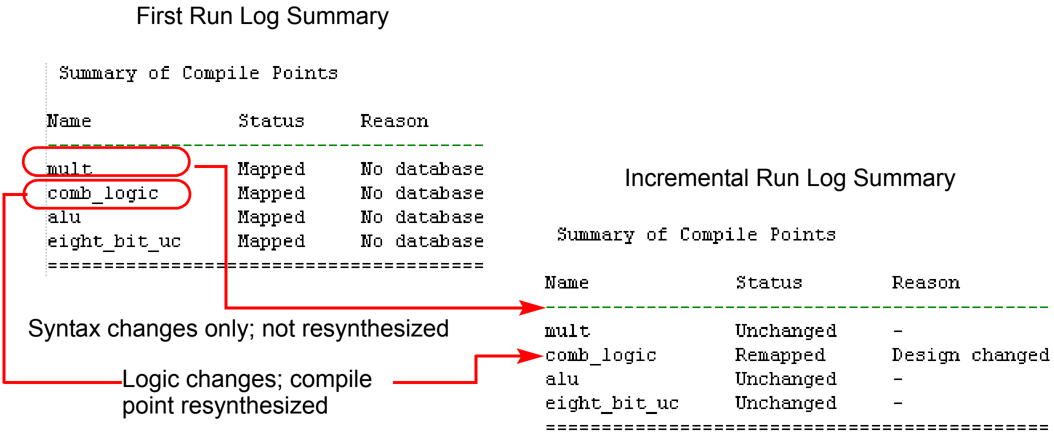
6. To synthesize the design incrementally, do the following:

   – 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.



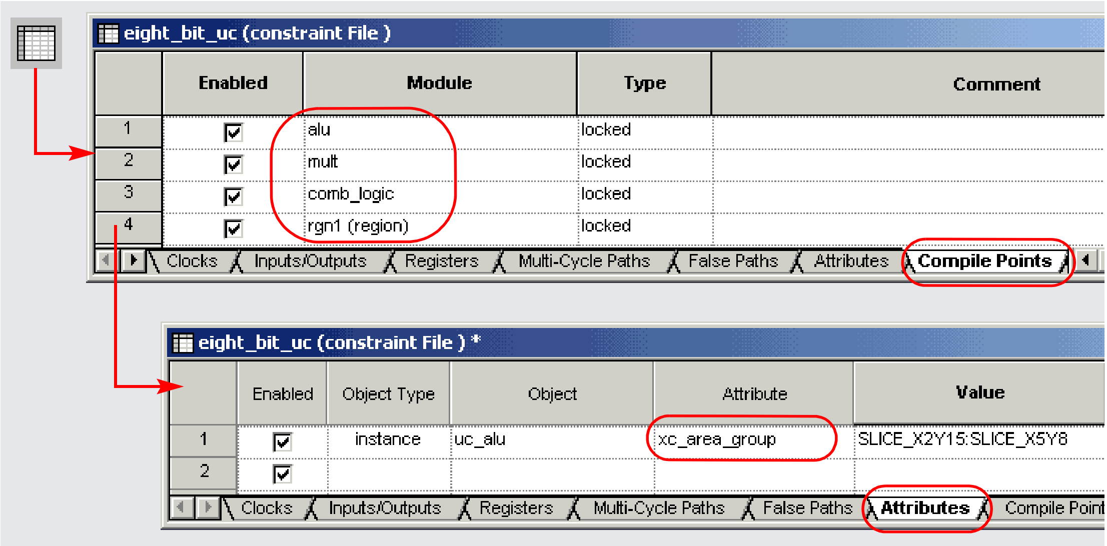
First Run Log Summary

```
 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
========================================
```

Incremental Run Log Summary

```
 Summary of Compile Points

Name                Status        Reason
---------------------------------------------
mult                Unchanged     -
comb_logic          Remapped      Design changed
alu                 Unchanged     -
eight_bit_uc        Unchanged     -
=============================================
```

Syntax changes only; not resynthesized
Logic changes; compile point resynthesized

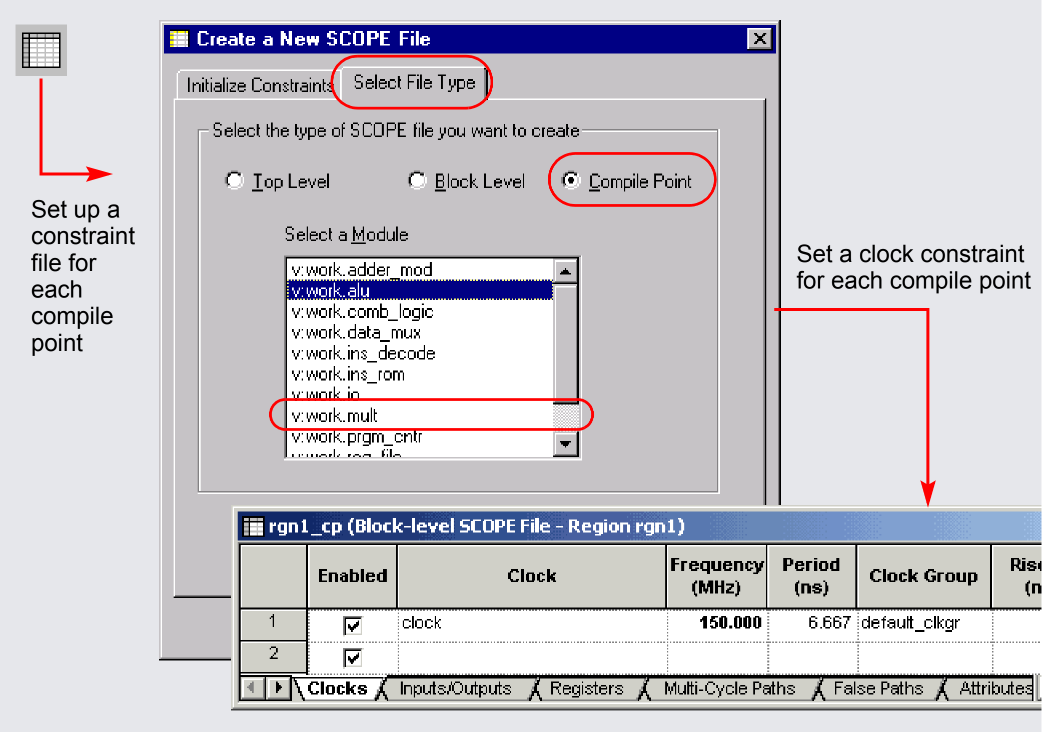


eight_bit_uc (constraint File)

	Enabled	Module	Type	Comment
1	☑	alu	locked	
2	☑	mult	locked	
3	☑	comb_logic	locked	
4	☑	rgn1 (region)	locked	

eight_bit_uc (constraint File) *

	Enabled	Object Type	Object	Attribute	Value
1	☑	instance	uc_alu	xc_area_group	SLICE_X2Y15:SLICE_X5Y8
2	☑				



Set up a constraint file for each compile point

Create a New SCOPE File

Select the type of SCOPE file you want to create

○ Top Level ○ Block Level ● Compile Point

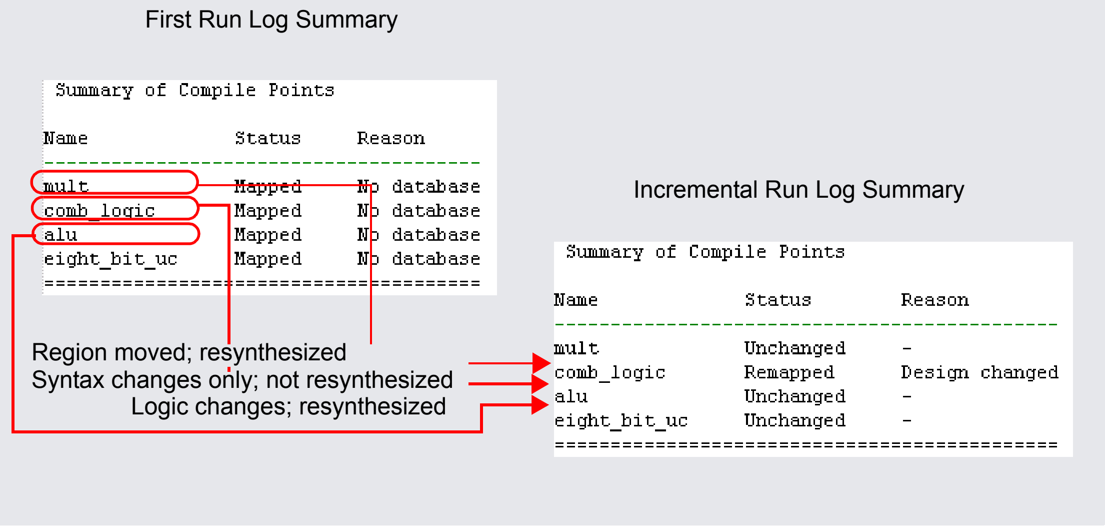
Select a Module

- v:work.adder_mod
- v:work.alu
- v:work.comb_logic
- v:work.data_mux
- v:work.ins_decode
- v:work.ins_rom
- v:work.io
- v:work.mult
- v:work.prgm_cntr
- v:work.reg_file

Set a clock constraint for each compile point

rgn1_cp (Block-level SCOPE File - Region rgn1)

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise (n
1	☑	clock	150.000	6.667	default_clkgr	
2	☑					



First Run Log Summary

```
 Summary of Compile Points

Name                Status      Reason
-----------------------------------------
mult                Mapped      No database
comb_logic          Mapped      No database
alu                 Mapped      No database
eight_bit_uc        Mapped      No database
=========================================
```

Incremental Run Log Summary

```
 Summary of Compile Points

Name                Status          Reason
---------------------------------------------
mult                Unchanged       -
comb_logic          Remapped        Design changed
alu                 Unchanged       -
eight_bit_uc        Unchanged       -
=============================================
```

Region moved; resynthesized

Syntax changes only; not resynthesized

Logic changes; resynthesized

Using the Xilinx Modular Flow

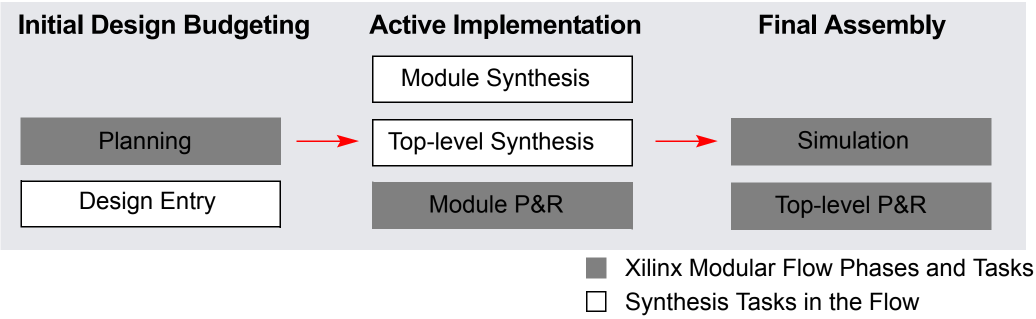
The modular flow was introduced in the Synplify Pro products to handle the needs of complex designs. This section contains an introduction and descriptions of the three phases of the modular design flow:

- Overview of Modular Flow Design Stages, next
- Initial Design Budgeting
- Active Implementation, on page 10-58
- Final Assembly, on page 10-63
- Design Files and Area Design Planning, on page 10-65

Overview of Modular Flow Design Stages

As designs become more complex, designers need to partition large designs into smaller modules to manage the complexity, leverage team design resources, and handle design changes more efficiently. The Synplify Pro tool supports a modular flow for Xilinx Virtex devices that allows designers to address these issues.

With the Xilinx Virtex modular flow, a team leader partitions the design and workload to work in parallel on modules that are subordinate to the top-level design. The modular flow consists of three design phases: Initial Design Budgeting, Active Implementation, and Final Assembly. The following figure illustrates how to use the Synplicity synthesis tools within this flow to accomplish tasks in the first two phases.



The steps in each phase are briefly described in the following sections, with emphasis on the tasks done with the synthesis tool.

Initial Design Budgeting

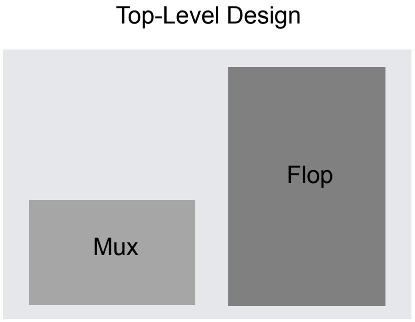
In this phase, the team leader uses the team design model to partition the design into modules, define the physical partition, and define the top-level design and global design constraints. Based on initial floorplanning, the lead team designer assigns physical locations to the modules.

Planning

Using the team design model, the team leader defines the HDL top-level design, the global design constraints, and determines the positions of each of the modules by doing some preliminary floorplanning. The team leader partitions the design into smaller self-contained modules or structures, and assigns the modules to different teams or designers. This planning stage can be merged with the next stage of Design Entry.

1. Partition the design into smaller self-contained modules or structures.
2. Use initial floorplanning to assign the modules to specific physical locations on the target device.
3. Allocate global resources like clock buffers.
4. Assign each module to a designer or a design team.

This modular flow uses a simple example to illustrate the flow. The design consists of a top-level design with two lower-level modules, mux and flop. See *Design Files and Area Design Planning*, on page 10-65 for the files.

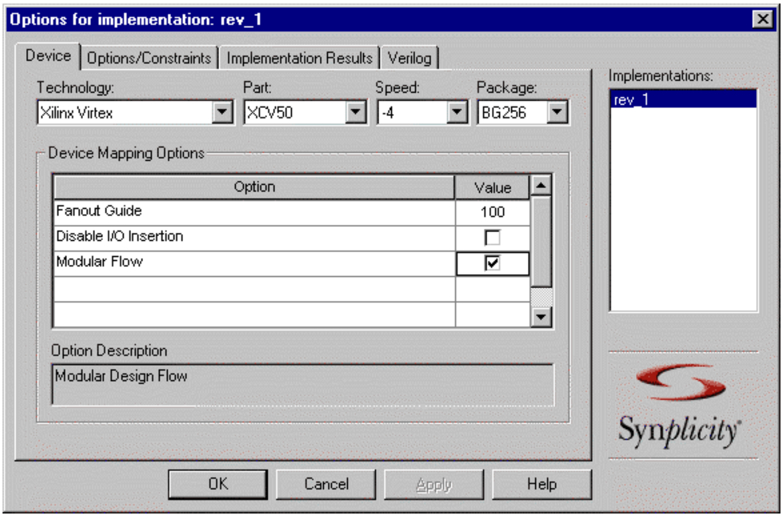


Design Entry (Team Leader)

The team leader must

1. Create the following files and add them to a project:
   - A top-level design file. The HDL source code for the top-level design contains the top-level module with all the global logic such as clock resources, inter-module logic connections, and connections between I/O ports and modules. For the top-level file used as an example here, see *Top-Level Design File*, on page 10-65.
   - A file for each module. This file is a black box wrapper for the module and lists the inputs and outputs. It also contains attributes to declare the module a black box (syn_black_box) and set the physical location of the module, based on initial floorplanning (xc_modular_region). For information about generating the area numbers for xc_modular_region, see *Determining the Area Range for xc_modular_region*, on page 10-69. For the module files used in this example, see *Module Mux File*, on page 10-67 and *Module Flop File*, on page 10-68.
   - A top-level constraint file. You can use the SCOPE interface to set global clock constraints. If there is a conflict between the global clock constraint and a clock constraint set on a module, the global clock constraint is used.

– Synthesize the design. Make sure you set Technology to Xilinx Virtex , Virtex-II, Virtex-II Pro, or Virtex-4 and check the Modular Flow checkbox on the Device tab of the Implementation Options form.



Options for implementation: rev_1

Technology: Xilinx Virtex
Part: XCV50
Speed: -4
Package: BG256

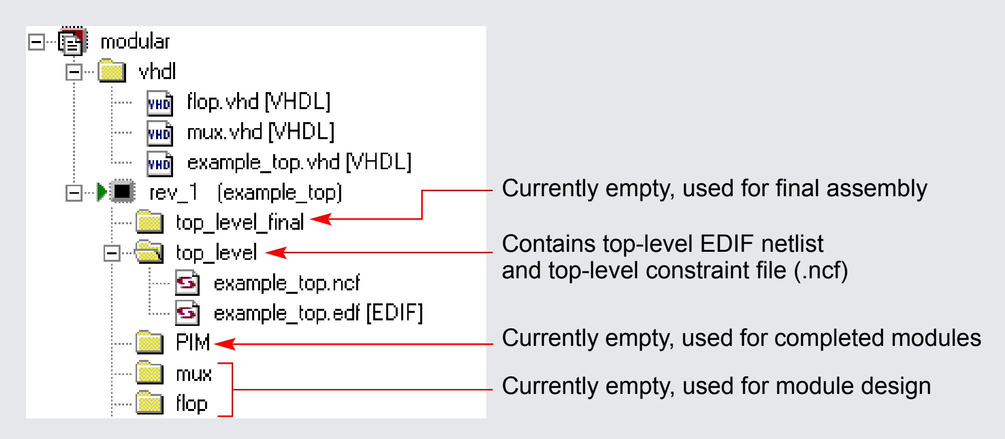
Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	□
Modular Flow	☑

Option Description
Modular Design Flow

Implementations: rev_1

When you check Modular Flow, the software generates the directory structure needed for the flow in the Xilinx place-and-route tool.



modular
vhdl
flop.vhd [VHDL]
mux.vhd [VHDL]
example_top.vhd [VHDL]
rev_1 (example_top)
top_level_final
top_level
example_top.ncf
example_top.edf [EDIF]
PIM
mux
flop
Currently empty, used for final assembly
Contains top-level EDIF netlist and top-level constraint file (.ncf)
Currently empty, used for completed modules
Currently empty, used for module design

The software creates top-level .edif and .ncf files, which it places in the top-level directory. It issues warnings if it finds a module that is instantiated more than once at the top level.

2. Generate a top-level .ngo file that contains the top-level area constraints.
   - Start the Xilinx place-and-route tool.
   - Go to the top_level directory and run **ngdbuild** in **initial** mode. Use this syntax and type the command at the command line:

     ngdbuild -modular initial <*toplevel*>.edf

   To run our example, the command is **ngdbuild -modular initial example_top.edf.** This command generates a .ngd file and a .ngo file that contains the top-level area constraints for place and route.

3. Archive the directories, and give each designer or design team working on a module a copy of the .prj file, the top-level HDL file, the sub-module HDL files (black box wrappers), and the top-level .sdc file with region constraints.

Design Entry (Module Level)

At this stage, the module designer must do the following:

1. Open the source code file(s) for the module you are responsible for, and delete the syn_black_box attribute. Do not remove the xc_modular_region attribute.

   For example, if you are the designer responsible for the mux design, you open the source code file for this module and delete the syn_black_box attribute. From your perspective, it is no longer a black box, because you are going to create the internal logic for it.

2. Complete the internal logic design for the module.

Active Implementation

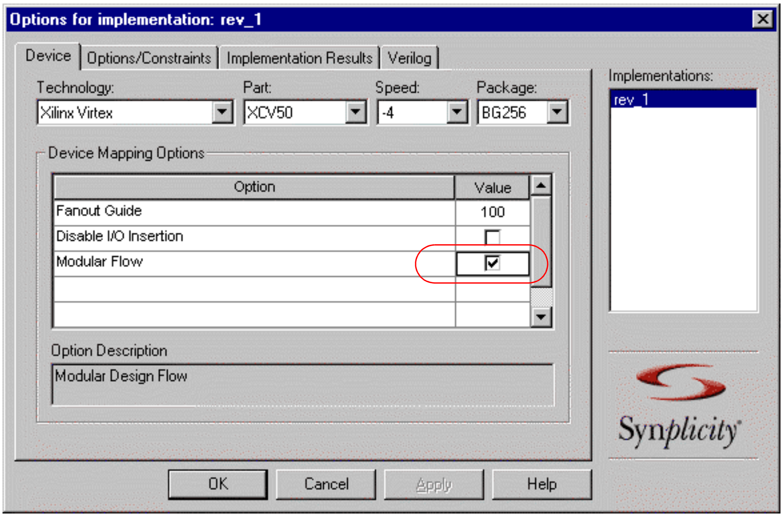
The active implementation phase starts after the designer finishes the module-level design and passes on the project to each development team. This phase consists of module synthesis, top-level synthesis, development of the physical partitions, and module-level placement and routing.

Module Synthesis

The advantage to using the modular design flow is the elimination of dependencies and the need for all teams to be done before an individual team can synthesize a module design with the overall design. Individual teams can synthesize their assigned modules with the top-level design without depending on the progress of other design teams. It allows the module designer to iterate the module design with the top level more frequently, and evolve the overall design and separate modules more efficiently.

Although this is module-level synthesis, you actually synthesize your module design with the top-level design, the top-level constraint file, wrapper files for other modules, and optional module-level constraint files. To use our example, if you are assigned to mux, you create HDL source code for it. You synthesize mux using the project created by the team leader, replacing the original mux wrapper file with the design you created. The project file includes the top-level design source code and constraint file, as well as the wrapper file for the flop module, which remains a black box from your perspective.

1. Start with the project set up by the team leader and set device options in the synthesis tool with Project->Implementation Options. The module file is now updated to contain the design.
   - On the Device tab, set the device to Xilinx Virtex or Xilinx Virtex-II.
   - Check the Modular Flow checkbox.



Options for implementation: rev_1

Technology: Xilinx Virtex

Part: XCV50

Speed: -4

Package: BG256

Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	☐
Modular Flow	☑

Option Description

Modular Design Flow

Implementations:

rev_1

- On the Implementation Results tab, check that the name of the output netlist matches the name of the module you are actively synthesizing as it occurs in the top-level EDIF file, so that the name of the netlist for this module and the name in the top-level EDIF file are the same. For example, the name for the output file for mux must be `mux.edf` to match the name of the component in the top-level EDIF file.
- Set any other device options you want, and click OK.

The software issues warnings if it finds either of the following:

- A module that is instantiated more than once at the top level.
- Internal tristates. Unlike the regular design flow, in the modular flow the software cannot move internal tristates up to the top level because of the strict hierarchy limits required by this flow.

2. Compile the design with Run->Compile Only. You need to do this to initialize constraints for the SCOPE environment.
3. Set module-level constraints if needed. Use the following procedure:
   - Select the hierarchical object (module) in the RTL view. Right-click and select SCOPE->Edit Module Constraints to open a SCOPE window and enter module constraints. Do not use the SCOPE icon, because that opens the top-level constraints file.
   - Set the module-level constraints and save the file. The file name is prefixed by module_. For example, the constraint file generated for the mux module is called `module_mux.sdc`.
   - Add it to the project. If there is a conflict between the global clock constraint (set at the top level) and a clock constraint set on a module, the global clock constraint is used.
4. Click the Run button to synthesize.

   As the design continues to evolve, the top-level source files sometimes change to accommodate overall design objectives. Make sure to use the most current top-level files. If there are design changes at the top level, all design teams must be updated with the changes by the team leader.

   The software generates the directory structure needed to continue the modular flow in the Xilinx place-and-route tool. For our example, you see the following directories after synthesizing mux with a black box for flop:

   - `constraint`, which contains the constraint files for the design

- `top_level_final`, which is still empty, but will be used for final assembly
- `top_Level`, which contains `example_top.edf`, the EDIF netlist for the top level and `example_top.ncf`, the top-level place-and-route constraint file
- `PIM` (Physically Implemented Module), which is currently empty, but will be used to hold modules as they are completed
- `mux`, which contains `mux.edf`, the EDIF netlist for the top level, and `mux.ncf`, the place-and-route constraint file for the module
- `flop`, which is still empty, because flop is still a black box

Unless you synthesize the top level, you have now finished synthesis. The rest of the flow uses the Xilinx place-and-route tool.

Top-level Synthesis

Optionally, you can synthesize the top-level design as an intermediate check, although it is not necessary with the modular flow. This is because the top-level module contains the physical locations of the devices and all modules use the same top-level area constraint file. At this point, you have finished the synthesis phase. The rest of the flow uses the Xilinx place-and-route tool.

Module Placement and Routing

After synthesis, you use the Xilinx placement and routing tool (Xilinx Software Solutions Version 3.1I) to place and route the module. The completed module can be then be instantiated in the top-level design. Each module is placed and routed independently with the Xilinx P&R tool. The modules are developed in parallel, so the post-routing timing results of one module can be used with placement and routing constraints when you synthesize another module with the Synplify Pro software.

The place-and-route commands are included here for completeness. If you need more information about the commands, see the Xilinx software documentation.

You must run placement and routing from the command line, but you could create a script to automate this process.

1. Open the Xilinx place-and-route software.

2. Go to the module directory (in this case, the `mux` directory) and type the following at the command line to run the **ngdbuild** command in module mode:

   ```
   ngdbuild -p <part_num> -modular module -active <module>.edf
   <path_to_top_level_ngo_file>
   ```

   For our example, the command is

   ```
   ngdbuild -p xcv50-6bg256 -modular module -active mux
   ..\top_level\example_top.ngo
   ```

   This command runs the **`ngdbuild`** command in module mode. It uses the area constraints from the top-level `.ngo` file to build an `.ngo` file in the module directory.

3. Map the module with this command:

   ```
   map <top_level_file>.ngd
   ```

   For our example:

   ```
   map example_top.ngd
   ```

   This command uses the `.ngo` file in the module directory to generate `.ngd` and `.ncd` files in the module directory.

4. Place and route the module with the following command:

   ```
   par -w <top_level_file>.ncd <top_level_file_par>.ncd
   ```

   The second `.ncd` file is the placed and routed file that is generated by this command, so name it something distinct and meaningful to you. By not overwriting the original `.ncd` file, you can try different design options. The `_par` suffix is a convention that lets you keep track of when you generated the `.ncd` file. For our example, use this command:

   ```
   par -w example_top.ncd example_par.ncd
   ```

5. Open the place-and-route tool and check your placement results with this command:

   ```
   fpga_editor <top_level_file_par>.ncd
   ```

   For our example, the name of the `.ncd` file is `example_par`.

6. Copy the lower-level module files into the PIM directory. From the module directory, type the following

   ```
   pimcreate -ncd -w <top_level_file_par>.ncd
   <path_to_pim_directory> <module_file>
   ```

   For the module file, just specify the name of the module file without any extensions. For example:

   ```
   pimcreate -ncd -w example_par.ncd ..\PIM mux
   ```

   This command creates a directory under PIM called `mux`, into which it copies the `mux.ncd` and `mux.ngo` files. The PIM directory is an intermediate holding area where module designers deposit module files as they are completed. The PIM files are used by the team leader for final assembly of the design.

Final Assembly

After all subordinate modules have been synthesized separately, use their combined netlists to place and route the entire design.The project leader merges the physical partition files from each development team and then does the final placement and routing to create a final netlist.

1. Complete all the modules, and copy them to the PIM directory. See *Module Synthesis*, on page 10-59 for details. At this point, you can do functional and timing simulation before placing and routing the top level with the Xilinx P&R tool.

2. Open a command prompt, and go to the `top_level_final` directory, and type the following:

   ```
   ngdbuild -p <part_num> -modular assemble -pimpath
   <path_to_pim_dir> -use_pim <module_name>
   <path_to_top_level_ngo_file>
   ```

   This command runs **`ngdbuild`** in assembly mode, specifying each of the modules in the pim directory. You must repeat **`-use_pim`** *`<module_name>`* as many times as needed to specify all the modules in the design. For example:

   ```
   ngdbuild -p xcv50-6bg256 -modular assemble -pimpath ..\PIM -use_pim
   mux -use_pim flop example_top.ngo
   ```

   The command generates a top-level `.ngd` file that is a fully expanded design file.

3. Map the design with the following command. The command maps each module using the files in the pim directory.

   ```
   map <top_level_file>.ngd
   ```

   ```
   For our example:
   ```

   ```
   map top.ngd
   ```

4. Place and route the design with the following command. The command uses the files in the PIM directory.

   ```
   par -w <top_level_file>.ncd <top_level_file_par>.ncd
   ```

   For example:

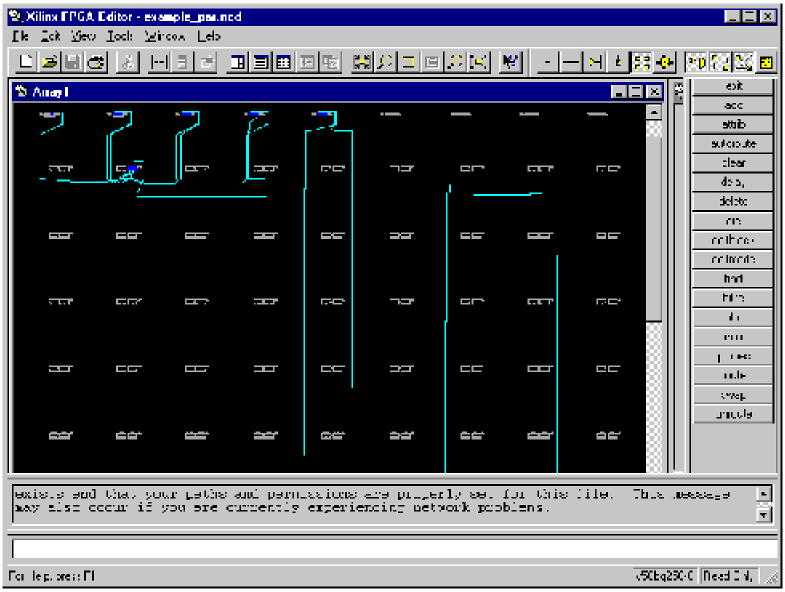
   ```
   par -w example_top>.ncd example_par.ncd
   ```

5. Open the place-and-route tool and verify the locations of the modules with this command:

   ```
   fpga_editor <top_level_file_par>.ncd
   ```

   To use our example, you type the following:

   ```
   fpga_editor <top_level_file_par>.ncd
   ```


Xilinx FPGA Editor - example_par.ncd
File Edit View Tools Window Help
Array1
exit
attrib
clear
delete
find
For Help, press F1

exists and that your paths and permissions are properly set for this file. This message may also occur if you are currently experiencing network problems.

Design Files and Area Design Planning

This section contains the top-level design files for the example used in the modular flow and discusses how to estimate module area.

Top-Level Design File

This is the top-level file (`example_top.vhd` or `example_top.v`) used as an example in the modular design flow. It contains the ports for the whole design and instantiations of the two lower-level modules.

VHDL

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

entity example_top is port (
   inmux : in std_logic_vector(3 downto 0);
   sel : in std_logic_vector(1 downto 0);
   reset : in std_logic;
   clk : in std_logic;
   output : out std_logic);
end example_top;

architecture beh of example_top is

component mux is port (
   inmux : in std_logic_vector(3 downto 0);
   sel : in std_logic_vector(1 downto 0);
   outmux : out std_logic);
end component;

component flop is port (
   inflop : in std_logic;
   clk : in std_logic;
   reset : in std_logic;
   outflop : out std_logic);
end component;

signal bridge : std_logic;

begin
U0 : mux port map (inmux,sel,bridge);
U1 : flop port map (bridge,clk, reset, output);
end beh;
```

Verilog

```
module modular_design_top (inmux, sel, reset, clk, out);

   input [3:0] inmux;
   input [1:0] sel;
   input        reset;
   input        clk;
   output       out;

wire bridge;

mux   U0    (.inmux(inmux), .sel(sel), .outmux(bridge));
flop U1 (.inflop(bridge), .outflop(out), .reset(reset),
.clk(clk));

endmodule
```

Module Mux File

For the module, the team leader creates a port map, declares the module as a black box, and specifies the physical location of the module on the FPGA (`mux.vhd` or `mux.v`). The syn_black_box attribute declares this module to be a black box, until the module designer deletes the attribute and creates the internals for the module. The xc_modular_region attribute specifies the range of CLBs (or slices in Virtex-II designs) on the FPGA into which this module can be placed. In this case, mux goes in the region between R1C1 and R5C5.

VHDL

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

entity mux is port (
   inmux : in std_logic_vector(3 downto 0);
   sel : in std_logic_vector(1 downto 0);
   outmux : out std_logic);
end mux;

architecture beh of mux is

   attribute syn_black_box : boolean;
   attribute syn_black_box of beh : architecture is true;
   attribute xc_modular_region : string;
   attribute xc_modular_region of beh : architecture is
"CLB_R1C1:CLB_R5C5";

begin
end beh;
```

Verilog

```
module mux (inmux, sel, outmux) /* synthesis syn_black_box
xc_modular_region="CLB_R1C1:CLB_R5C5" */;

   input   [3:0] inmux;
   input   [1:0] sel;
   output        outmux;

endmodule
```

Module Flop File

For the module, the team leader creates a port map, declares the module as a black box, and specifies the physical location of the module on the FPGA (`flop.vhd` or `flop.v`). The syn_black_box attribute declares this module to be a black box, until the module designer deletes the attribute and creates the internals for the module. The xc_modular_region attribute specifies the range of CLBs (or slices in Virtex-II designs) on the FPGA into which this module can be placed. In this case, flop goes in the region between R7C7 and R8C8.

VHDL

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

entity flop is port (
   inflop : in std_logic;
   clk : in std_logic;
   reset : in std_logic;
   outflop : out std_logic);
end flop;

architecture beh of flop is
   attribute syn_black_box : boolean;
   attribute syn_black_box of beh : architecture is true;
   attribute xc_modular_region : string;
   attribute xc_modular_region of beh : architecture is
"CLB_R7C7:CLB_R8C8";

begin
end beh;
```

Verilog

```
module flop (inflop, outflop, reset, clk) /* synthesis
syn_black_box xc_modular_region="CLB_R7C7:CLB_R8C8" */;

   input   inflop;
   input   reset;
   input   clk;
   output outflop;

endmodule
```

Determining the Area Range for xc_modular_region

You determine the area range by first estimating the area of the module and then fitting it on to the FPGA along with the other modules.

1. Estimate the area of the module. You can use one of these methods:
   - Often, the team leader has a rough idea of the area of the module. Use this rough area estimate to determine the area range on the FPGA.
   - Synthesize the lower-level modules with the modular design option turned off. The log file gives you the resource usage, which you can use to calculate the area of the module.
2. Determine where it should be placed on the FPGA, based on its size.
3. Indicate placement area range with row and column numbers, and use these numbers as the value of the xc_modular_region attribute.
   - For Virtex designs, use this format: `CLB_R<`*`n`*`>C<`*`n`*`>:CLB_R<`*`n`*`>C<`*`n`*`>`. For example, `CLB_R3C3:CLB_R8C8`.
   - For Virtex-II designs, use this format: `SLICE_R<`*`n`*`>C<`*`n`*`>:SLICE_R<`*`n`*`>C<`*`n`*`>`. For example, `SLICE_R3C3:SLICE_R8C8`.

Integrating with Third-Party Software

This section discusses how to use synthesis results with software from other vendors to accomplish your design needs. For information about working with Altera and Xilinx software, refer to *Working with Altera Designs*, on page 8-11, *Working with Xilinx Designs*, on page 8-28, and *The Xilinx Multi-Point Synthesis Flow*, on page 10-48. You can also use a synhooks Tcl script (see *Automating Flows with synhooks.tcl*, on page 10-10) to integrate third-party software.

Resynthesizing with QuickLogic SpDE 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.

Synopsys DesignWare Component Support

In the Synplify Premier software, Synplicity DW-compatible models have been created for a number of the Synopsys® DesignWare® components. These models extract the functionality of the component, but not its implementation. The mappers synthesize the model to the most appropriate implementation.

Synopsys Designware Components are supported in certain Actel, Altera, and Xilinx technologies.

For a complete list of the supported components, see *Translating Third-party Libraries, on page H-1* in the *Reference Manual*.

Working with the Identify RTL Debugger

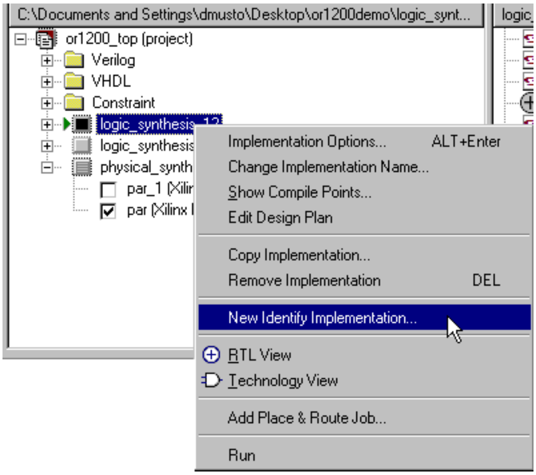
The Identify RTL Debugger is a dual-component system that is part of an HDL design flow process. The system consists of the Identify Instrumentor and Identify Debugger software tools. The combination of these tools allows you to probe your HDL design in the target environment. This system fits easily into your existing design flow, with only a few modifications.

The Identify Instrumentor tool allows you to select your design instrumentation at the HDL level and then create an on-chip hardware probe. The Identify Debugger tool interacts with the on-chip hardware probe and allows you to perform live debugging of the design. The combined system allows you to debug your design faster, easier, and more efficiently.

The Synplify, Synplify Pro, and Synplify Premier synthesis tools have integrated the Identify Instrumentor into the synthesis user interface. To use the Identify Instrumentor, create an Identify implementation and then launch the Identify Instrumentor from within the synthesis tool.

To add an Identify implementation:

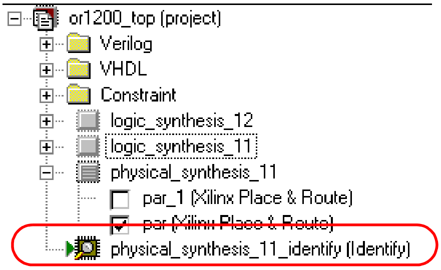
1. In the synthesis interface, open the design you want to instrument.
2. Do one of the following tasks to add an Identify implementation:
   – With the project implementation selected, right-click and select New Identify Implementation from the pop-up menu.



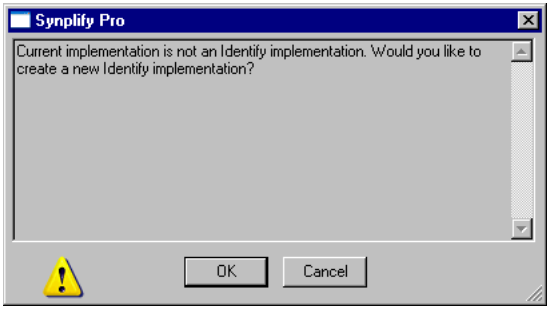
– Select Project->New Identify Implementation.

An Options for Implementations 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.

3. Close the Options for Implementations dialog box. An Identify implementation is created.

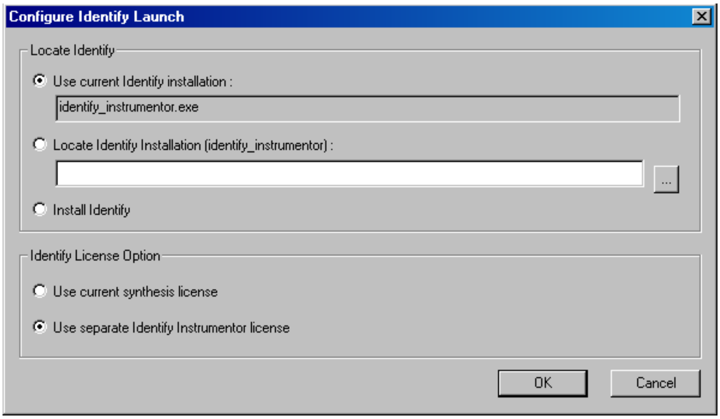


4. Open the Identify Instrumentor either by selecting the Launch Identify icon ( ) in the toolbar or by selecting Run->Identify Instrumentor. If you do not have an implementation created when you select Run->Launch Identify Instrumentor or select the Launch Identify icon ( ), the following message dialog box appears. Select OK.



Current implementation is not an Identify implementation. Would you like to create a new Identify implementation?

- If the location of the Identify Instrumentor executable is unknown, a Launch Identify dialog box appears.
- If the synthesis application locates the Identify software, it opens with the path to the Identify instrumentor executable.



Configure Identify Launch

Locate Identify

● Use current Identify installation :

identify_instrumentor.exe

○ Locate Identify Installation (identify_instrumentor) :

○ Install Identify

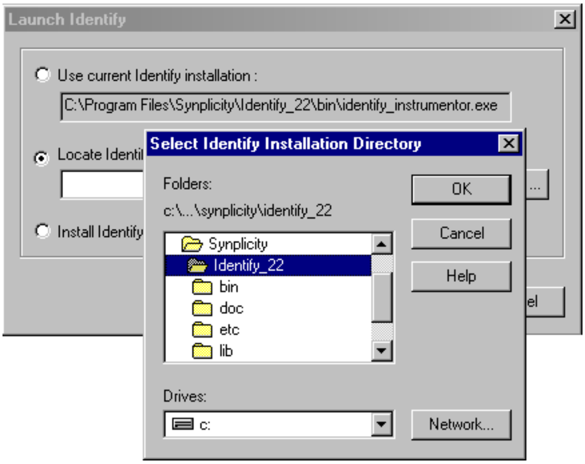
Identify License Option

○ Use current synthesis license

● Use separate Identify Instrumentor license

Note: If the icon and 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 Locate Identify Installation (identify_instrumentor): and the ... button. This opens the Select Identify Installation Directory dialog box. Locate and select your current Identify installation directory.



Launch Identify

○ Use current Identify installation :

C:\Program Files\Synplicity\Identify_22\bin\identify_instrumentor.exe

● Locate Identi

○ Install Identify

Select Identify Installation Directory

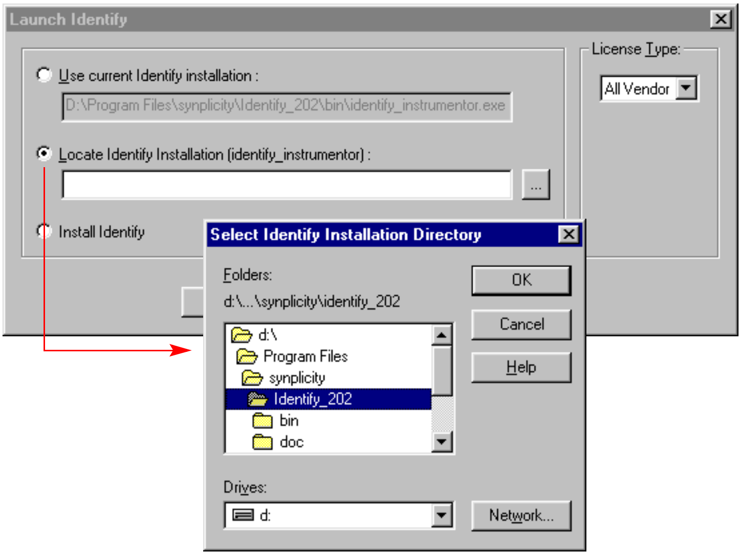
Folders:

c:\...\synplicity\identify_22

Synplicity
Identify_22
bin
doc
etc
lib

Drives:

c:



Launch Identify

○ Use current Identify installation : D:\Program Files\synplicity\Identify_202\bin\identify_instrumentor.exe

● Locate Identify Installation (identify_instrumentor) :

○ Install Identify

License Type: All Vendor

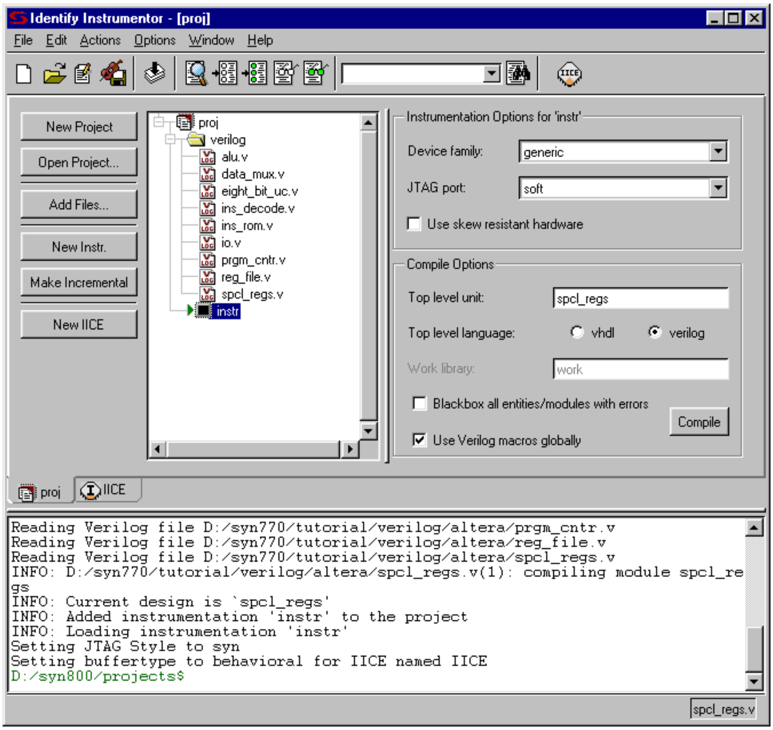
Select Identify Installation Directory

Folders: d:\...\synplicity\identify_202

Drives: d:

- If you do not have the Identify software installed, select Install Identify and click OK. Install the Identify software before proceeding as described above.

5. The Identify Instrumentor software interface opens, with an Identify project automatically set up for the design to be debugged.



Instrumentation Options for 'instr'

Device family: generic

JTAG port: soft

□ Use skew resistant hardware

Compile Options

Top level unit: spcl_regs

Top level language: ○ vhdl ● verilog

Work library: work

□ Blackbox all entities/modules with errors

☑ Use Verilog macros globally

```
Reading Verilog file D:/syn770/tutorial/verilog/altera/prgm_cntr.v
Reading Verilog file D:/syn770/tutorial/verilog/altera/reg_file.v
Reading Verilog file D:/syn770/tutorial/verilog/altera/spcl_regs.v
INFO: D:/syn770/tutorial/verilog/altera/spcl_regs.v(1): compiling module spcl_re
gs
INFO: Current design is `spcl_regs'
INFO: Added instrumentation 'instr' to the project
INFO: Loading instrumentation 'instr'
Setting JTAG Style to syn
Setting buffertype to behavioral for IICE named IICE
D:/syn800/projects$
```

6. 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

7. 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.
8. Place and route the instrumented design after synthesis.
9. Use the Identify Debugger tool to debug the instrumented design. You do not have access to the Identify Debugger with the evaluation copy. To use the Identify Debugger, you must have a full-up version of Identify.

CHAPTER 11

Synplify Premier Design Flow

This document describes how to use the Synplify Premier tool, with and without a design plan, in a physical synthesis design flow.

Topics include:

Synplify Premier Physical Synthesis Flows

Synplify Premier supports the following physical synthesis flows:

- Graph-based physical synthesis—a fully automated flow for incremental performance improvement producing a design with detailed placement. Graph-based physical synthesis is currently available for Virtex-II Pro, Virtex-4, and Spartan-3.
- Graph-based physical synthesis with a design plan—a graph-based physical synthesis flow that is guided by a design plan. It also produces a design with detailed placement. Graph-based physical synthesis with a design plan is currently available for Virtex-II Pro, Virtex-4, and Spartan-3.
- Design-plan based physical synthesis—an interactive flow that lets you perform physical synthesis optimization using the Design Planner view. The Design-plan based physical synthesis produces a design with coarse placement. The Design-plan based physical synthesis is available for Altera Cyclone, Cyclone-II, Stratix, Stratix-GX, and Stratix-II and Xilinx Virtex, Virtex-II, and Virtex-E.

Device Support for the Physical Synthesis Flows

The following table describes the devices that are supported for the following Synplify Premier Physical Synthesis flows:

Physical Optimization Flows	Altera Devices	Xilinx Devices
Graph-based Physical Synthesis		Spartan-3 Virtex-II Pro Virtex-4
Graph-based Physical Synthesis with a Design Plan (Design Planner option of the Synplify Premier tool only)		Spartan-3 Virtex-II Pro Virtex-4
Design Plan-based Physical Synthesis (Design Planner option of the Synplify Premier tool only)	Cyclone Cyclone II Stratix Stratix-II Stratix-II GX	Virtex Virtex-II Virtex-E

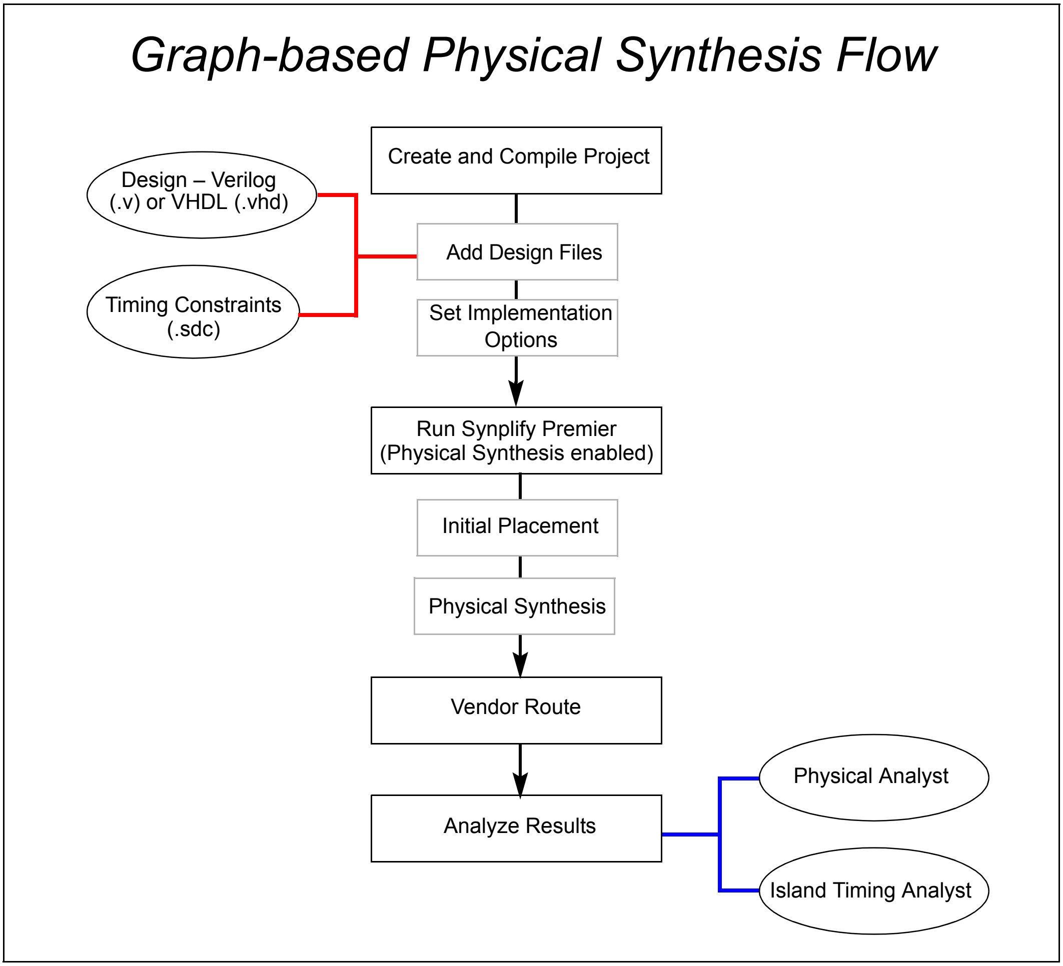
Graph-based Physical Synthesis

The Synplify Premier tool includes a Graph-based Physical Synthesis feature. This graph-based physical synthesis approach provides a single-pass flow for 90 nm FPGAs. Pre-existing wires, switches, and placement sites used for routing represent a detailed routing resource graph of the FPGA, which measures delay and availability of wires. Physical synthesis performs concurrent placement and synthesis optimizations to ensure fast routes for critical paths and generates a fully-placed and physically-optimized netlist ready for the vendor place-and-route tool. This push-button flow easily provides from 5% to 20% timing improvements.

Graph-based Physical Synthesis methodology allows the Synplify Premier software to constrain assigned logic to specific CLB locations for an entire device, while optimizing the design based on this placement information. The Synplify Premier tool integrates synthesis and placement by performing concurrent placement and synthesis optimizations for the design based on timing, constraints, and the device technology.

Graph-based Physical Synthesis Design Flow

The following figure shows the Graph-based Physical Synthesis design flow, which includes the Synplify Premier features and tools to run physical synthesis. Note that this feature is currently applicable for certain device technologies only.


Graph-based Physical Synthesis Flow
Create and Compile Project
Design – Verilog (.v) or VHDL (.vhd)
Add Design Files
Timing Constraints (.sdc)
Set Implementation Options
Run Synplify Premier (Physical Synthesis enabled)
Initial Placement
Physical Synthesis
Vendor Route
Analyze Results
Physical Analyst
Island Timing Analyst

Graph-based Physical Synthesis Tasks

To run Graph-based Physical Synthesis successfully with Synplify Premier, be aware of the following conditions:

- You must select a target technology that supports Graph-based Physical Synthesis.
- Xilinx place-and-route software is available for initial placement of the design.

  - You do not need to add a place-and-route job to your project.
  - Check the xflow_gp.log file for place-and-route results.
- Make sure the design is complete (including all IPs with no black boxes) and properly constrained.
- Simply click on the Run button to start this push-button physical synthesis flow, which also includes running placement and routing.

  The Synplify Premier software automatically controls the settings for these optimizations.
- You do not have to specify a design plan file (`.sfp`) for your project

  Creating a `.sfp` file requires using the separately-licensed Design Planner option of Synplify Premier.

For more information, see the following topics:

- For applicable Graph-based Physical Synthesis technologies, see *Device Support for the Physical Synthesis Flows* on page 11-3.
- For the Synplify Premier with Design Planning, see *Design Plan-based Physical Synthesis Flow* on page 11-8.

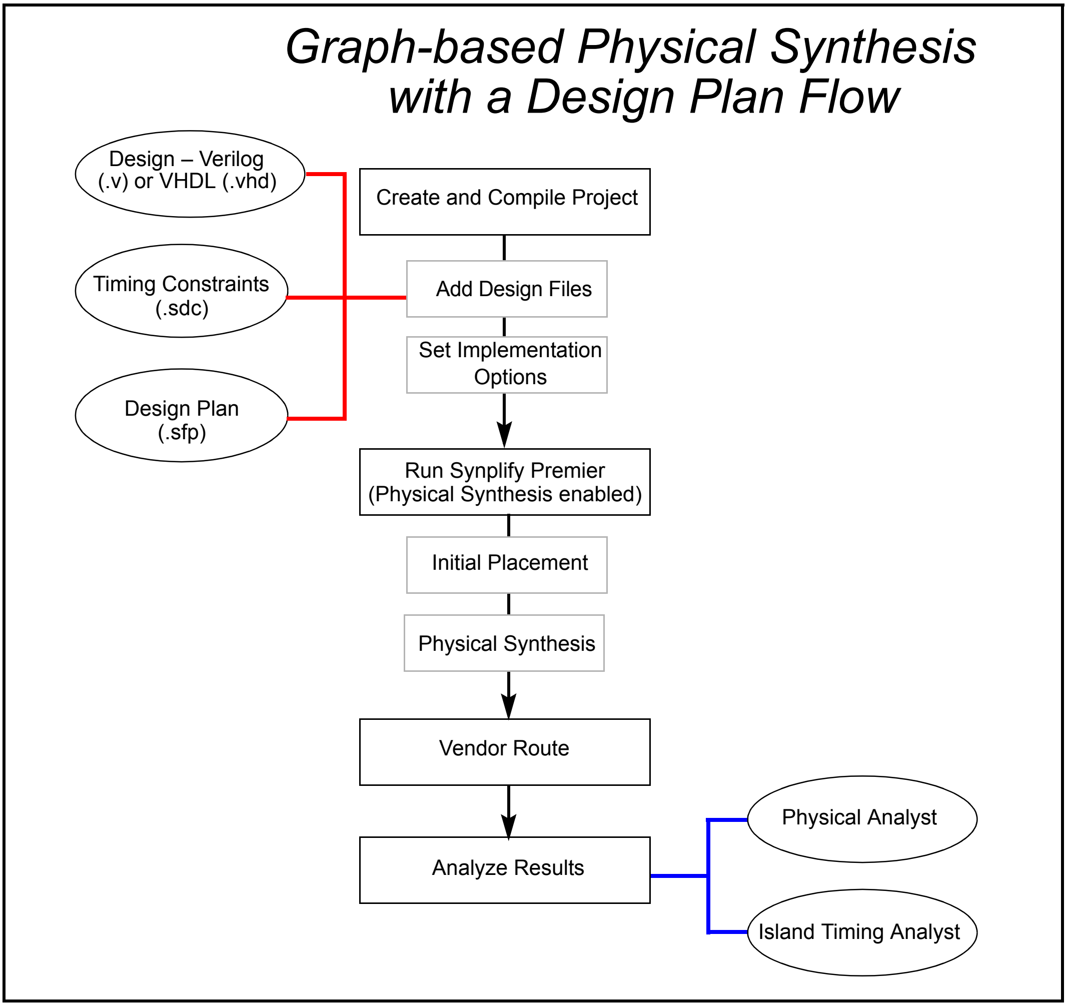
Graph-based Synthesis Flow Specifications

This feature helps simplify the process that provides critical path timing improvements for the design. The Synplify Premier tool can implement Graph-based Physical Synthesis for the following conditions:

- Device is Xilinx Virtex-4, Virtex-II Pro, or Spartan-3 architecture.
- Designs can contain block RAMs, block multipliers, or external user-defined modules provided in an EDIF netlist.
- Placement is defined for LOC and RLOC constraints.
- Xilinx place-and-route software is available for initial placement of the design.
- Black boxes, compile points, and MultiPoint synthesis area groups are not present in the design.
- Regions can be created to constrain logic within a particular area of the device (Design Planner option of Synplify Premier only).

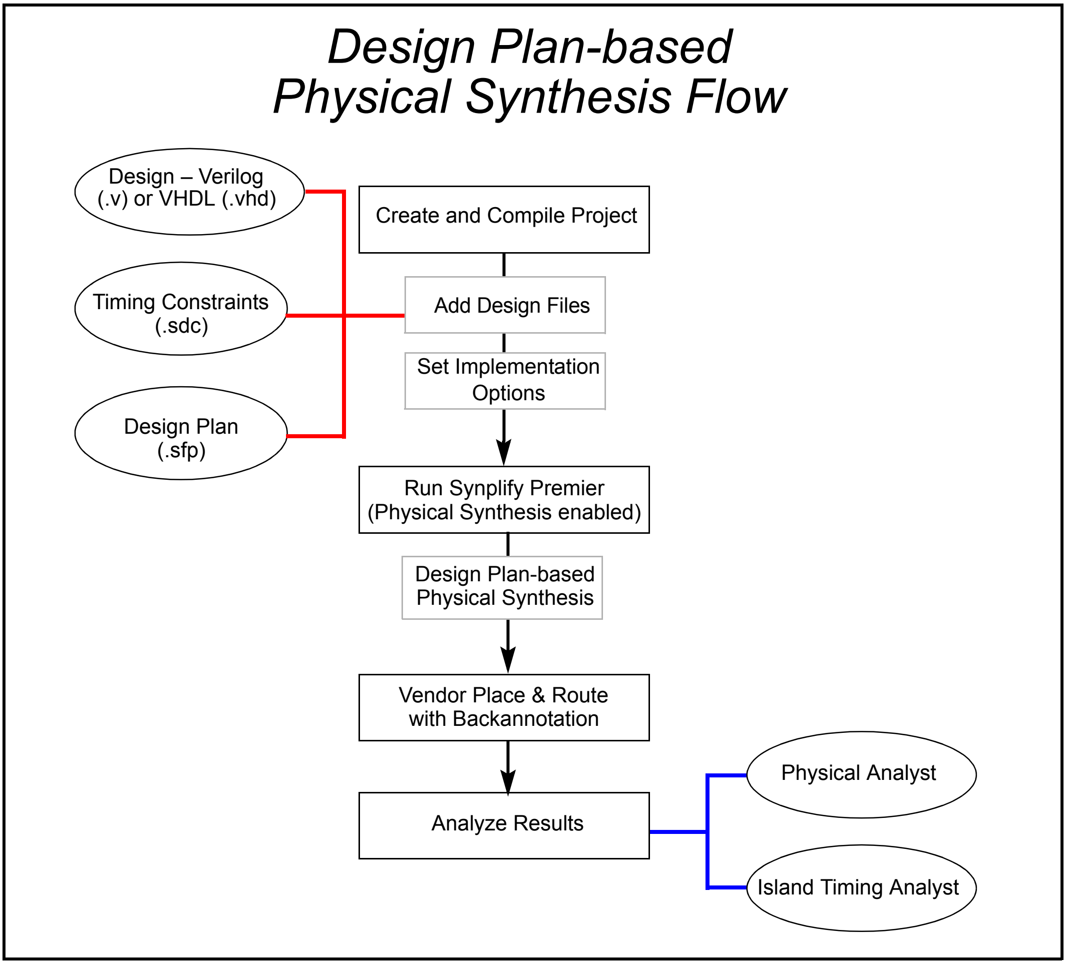
Graph-based Physical Synthesis with a Design Plan Flow

The Graph-based Physical Synthesis with a Design Plan flow for Virtex-II Pro, Virtex-4, and Spartan-3 combines the graph-based flow with detailed placement with physical optimization using the Design Planner.


Graph-based Physical Synthesis with a Design Plan Flow
Design – Verilog (.v) or VHDL (.vhd)
Timing Constraints (.sdc)
Design Plan (.sfp)
Create and Compile Project
Add Design Files
Set Implementation Options
Run Synplify Premier (Physical Synthesis enabled)
Initial Placement
Physical Synthesis
Vendor Route
Analyze Results
Physical Analyst
Island Timing Analyst

Design Plan-based Physical Synthesis Flow

The Design Planner is a separate licensed option to the Synplify Premier software. You use the Design Planner to create a design plan to create and edit regions in your design. The following figure shows the Design Planner design flow, which includes the features and tools to run physical synthesis. Note that some of these features are only applicable for certain device technologies.


Design Plan-based
Physical Synthesis Flow
Design – Verilog (.v) or VHDL (.vhd)
Timing Constraints (.sdc)
Design Plan (.sfp)
Create and Compile Project
Add Design Files
Set Implementation Options
Run Synplify Premier (Physical Synthesis enabled)
Design Plan-based Physical Synthesis
Vendor Place & Route with Backannotation
Analyze Results
Physical Analyst
Island Timing Analyst

Design Planner Tasks

1. First run the Synplify Premier software with the Design Planner option using standard timing constraints in a normal logic synthesis flow to determine if timing performance enhancements are needed.

2. Run the place-and-route tool.

3. Analyze the results using the following tools:

   – Synthesis log file and the place-and-route results file.

   – Physical Analyst to view and analyze placement information.

   – Island Timing Analyst to view and analyze timing information.

   Use these tools to help facilitate creating these physical constraints. Currently, these tools are only available for Altera and Xilinx devices that support place-and-route with backannotation.

4. Create a design plan with the Design Planner, to which you can interactively assign the critical paths to rows or regions on the device.

5. Run Synplify Premier with the design plan to optimize the design.

6. Rerun the place-and-route tool and analyze results.

Running Physical Synthesis

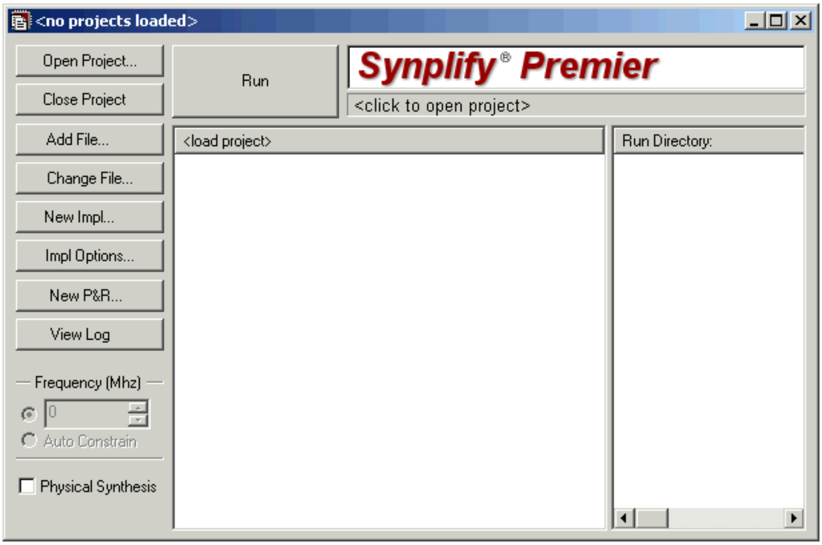
The following describes the tasks to run Graph-based, Graph-based with a design plan, or Design plan-based physical synthesis.

- Create the Project File on page 11-10
- Set Implementation Options on page 11-11
- Run Place-and-Route on page 11-18
- Synthesize the Design on page 11-21
- Analyze Results on page 11-22
- Running Multiple Implementations on page 11-24

Create the Project File

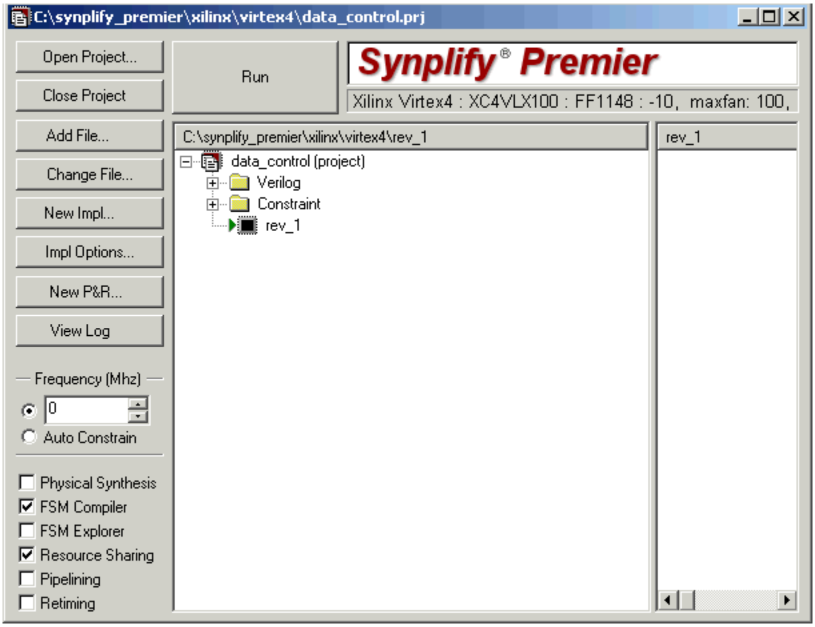
The Synplify Premier physical synthesis flow requires a design project file (`.prj`). To do this:

1. Open the Synplify Premier tool.



2. Create a project.
   - File->New
   - Click on the Open Project button, then New Project.
3. Add source files:
   - HDL source files (`.v`/`.vhd`)
   - Constraint files (`.sdc`)
   - For the Design Planner option, add design plan files (`.sfp`). (See Chapter 7, *Design Planning and Optimizations* if you do not have a design plan file.)

   See Chapter 2, *Project Setup* for information on how to add source files.



Xilinx Virtex4 : XC4VLX100 : FF1148 : -10, maxfan: 100,

Frequency (Mhz)

● 0
○ Auto Constrain

□ Physical Synthesis
☑ FSM Compiler
□ FSM Explorer
☑ Resource Sharing
□ Pipelining
□ Retiming

4. Save the project file.

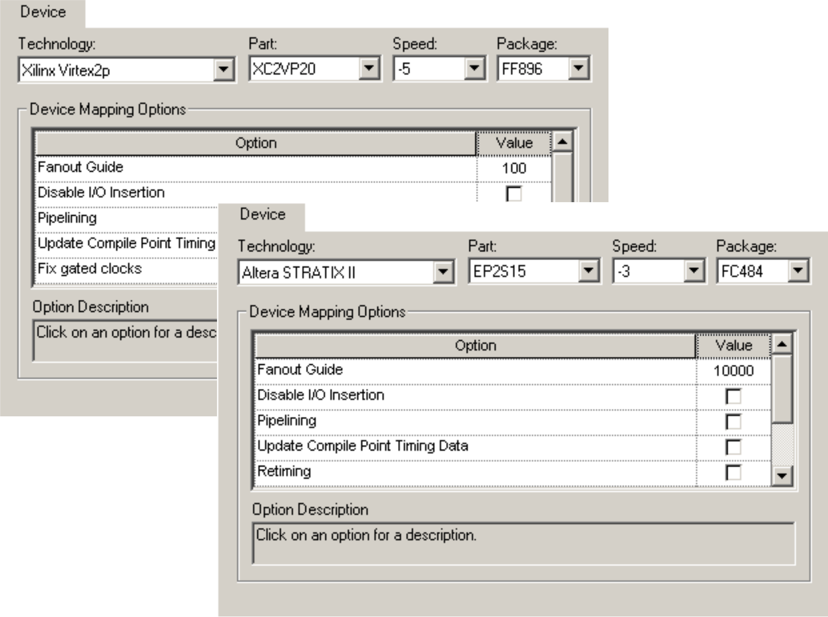
Set Implementation Options

After creating your design project, you can specify the options for the physical synthesis design run.

Bring up the Implementations Options dialog box (Impl Options button).

1. In the Device panel, set Device technology and options for:
   - Technology, part, speed, and package
   - Device mapping options

Note: Currently, Graph-based Physical Synthesis only supports Xilinx Virtex-4, Virtex-II Pro, and Spartan-3 technologies.



Device

Technology: Xilinx Virtex2p
Part: XC2VP20
Speed: -5
Package: FF896

Device Mapping Options

Option	Value
Fanout Guide	100
Disable I/O Insertion	□
Pipelining	
Update Compile Point Timing	
Fix gated clocks	

Option Description

Click on an option for a desc

Device

Technology: Altera STRATIX II
Part: EP2S15
Speed: -3
Package: FC484

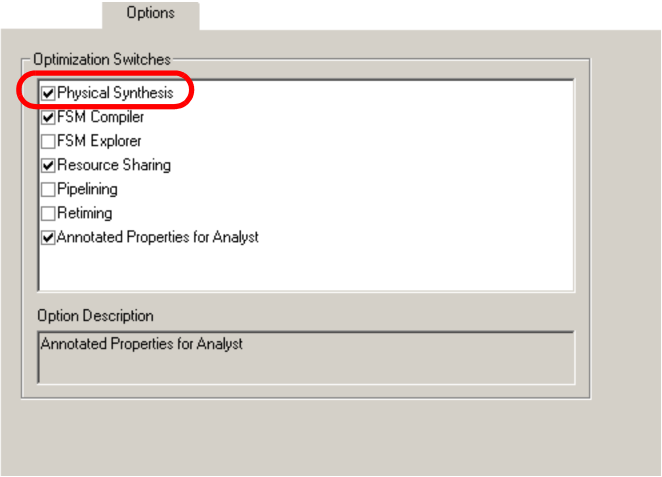
Device Mapping Options

Option	Value
Fanout Guide	10000
Disable I/O Insertion	□
Pipelining	□
Update Compile Point Timing Data	□
Retiming	□

Option Description

Click on an option for a description.

2. Click on the Options tab and set optimization switches for physical synthesis. Make sure to enable the Physical Synthesis option.



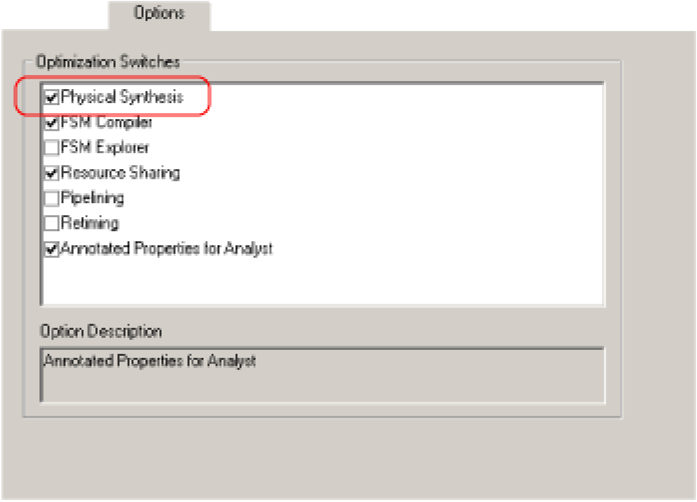
Optimization Switches

☑ Physical Synthesis
☑ FSM Compiler
☐ FSM Explorer
☑ Resource Sharing
☐ Pipelining
☐ Retiming
☑ Annotated Properties for Analyst

Option Description

Annotated Properties for Analyst

You can also enable the Physical Synthesis switch from the Project view.



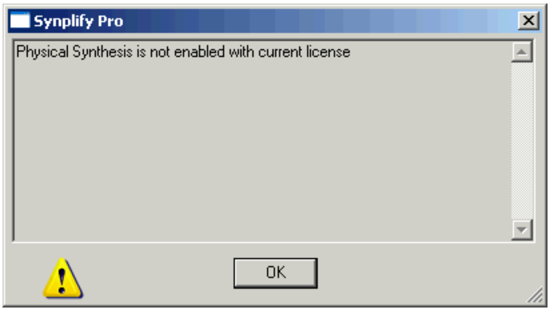
Optimization Switches

☑ Physical Synthesis
☑ FSM Compiler
☐ FSM Explorer
☑ Resource Sharing
☐ Pipelining
☐ Retiming
☑ Annotated Properties for Analyst

Option Description

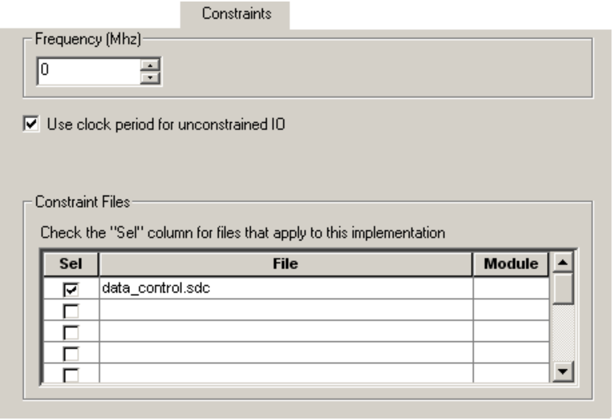
Annotated Properties for Analyst

However, if you try to enable this option without a Synplify Premier license, the following message appears:



Physical Synthesis is not enabled with current license

3. Click on the Constraints tab:
   - Set an overall target frequency for the design. See *Specifying Global Frequency and Constraint Files* on page 3-6 for information.
   - Make sure the constraint file that you want to use is selected. If you do not see the desired constraint files in the pane, you need to either create one, or add an existing `.sdc` file to your project.



Frequency (Mhz): 0

☑ Use clock period for unconstrained IO

Constraint Files

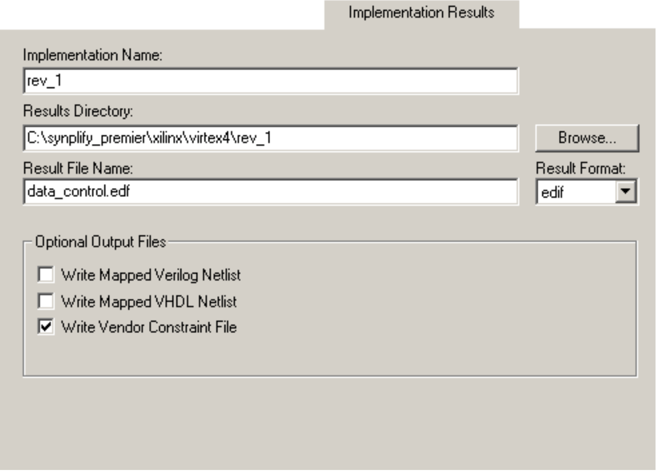
Check the "Sel" column for files that apply to this implementation

Sel	File	Module
☑	data_control.sdc	
□		
□		
□		
□		

See:

- Setting Constraints in the SCOPE Window on page 3-18 for information on creating constraint files.

4. Click on the Implementation Results tab and specify your output options. See *Specifying Result Options* on page 3-9 for details.



Implementation Name: rev_1

Results Directory: C:\synplify_premier\xilinx\virtex4\rev_1

Result File Name: data_control.edf

Result Format: edif

Optional Output Files

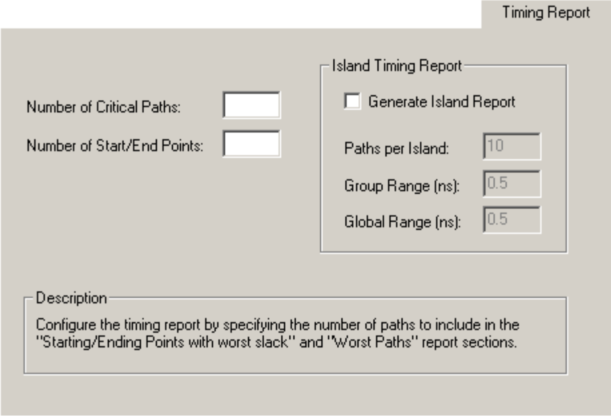
- □ Write Mapped Verilog Netlist
- □ Write Mapped VHDL Netlist
- ☑ Write Vendor Constraint File

5. Click on the Timing Report tab and specify the following:

   – Number of critical paths and start/end points to display in the timing report.

   – Enable the option to generate an island timing report. Then specify the parameters used to generate the timing report.

   If you do not wish to generate an island timing report at this time, you can choose to use the interactive Island Timing Analyst tool after you run synthesis instead. See *The Island Timing Analyst* on page 4-88.



Number of Critical Paths:

Number of Start/End Points:

Island Timing Report

- □ Generate Island Report

Paths per Island: 10

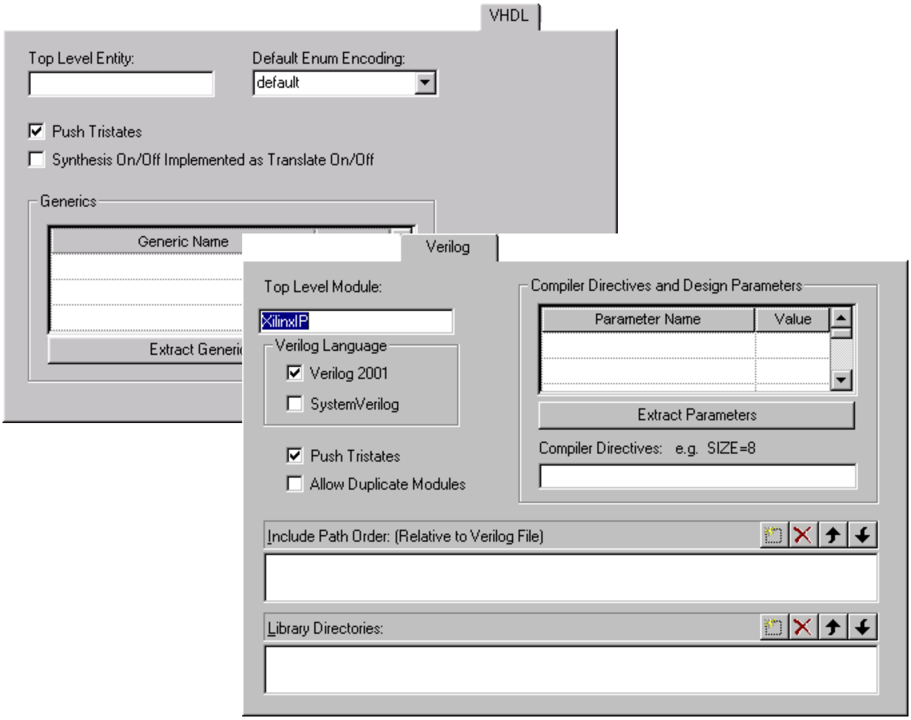
Group Range (ns): 0.5

Global Range (ns): 0.5

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.

6. From the Verilog/VHDL tab, specify the desired HDL options. See *Setting Verilog and VHDL Options* on page 3-11.



Top Level Entity:

Default Enum Encoding: default

☑ Push Tristates

☐ Synthesis On/Off Implemented as Translate On/Off

Top Level Module: XilinxIP

☑ Verilog 2001

☐ SystemVerilog

☑ Push Tristates

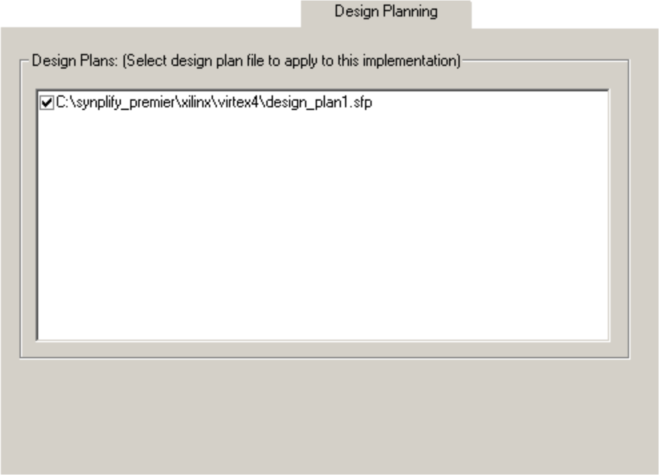
☐ Allow Duplicate Modules

Compiler Directives: e.g. SIZE=8

Include Path Order: (Relative to Verilog File)

Library Directories:

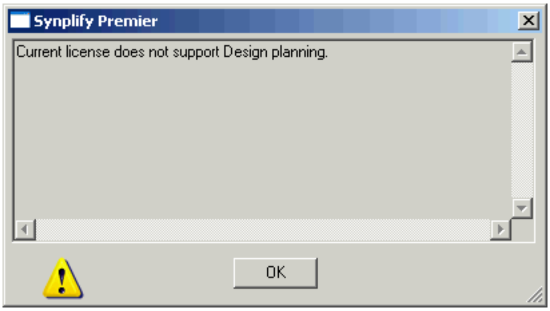
7. If you are running the Design-based or Graph-based with a design plan physical synthesis flow, click on the Design Planning tab and make sure the design plan file (`.sfp`) is selected. For Altera and certain Xilinx technologies, you must create a design plan (`.sfp`) to run physical synthesis. However, you do not need to select a design plan file to run Graph-based physical synthesis.



Design Plans: (Select design plan file to apply to this implementation)

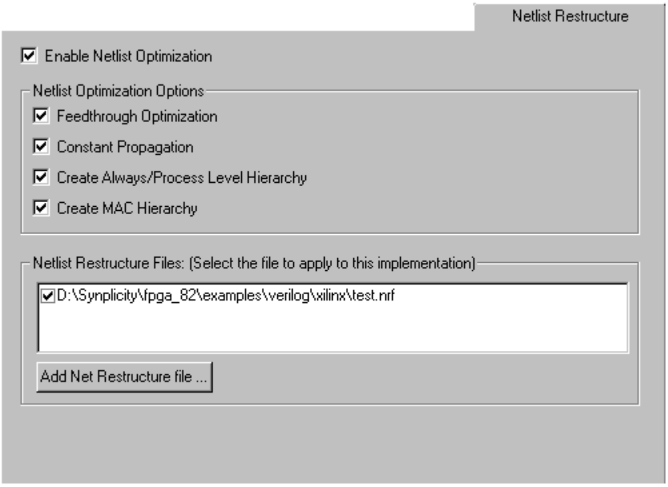
☑ C:\synplify_premier\xilinx\virtex4\design_plan1.sfp

If you try to enable a design plan file (.sfp) without a Design Planner license, a popup message appears stating that the current license does not support design planning.



Current license does not support Design planning.

8. From the Netlist Restructure tab, specify the following options:
   - Enable any necessary netlist optimizations. The Create MAC Hierarchy option is available only for certain Altera technologies.
   - Include any necessary netlist restructure file (.nrf) for which bit slicing or zippering might have been performed.



☑ Enable Netlist Optimization

Netlist Optimization Options

☑ Feedthrough Optimization

☑ Constant Propagation

☑ Create Always/Process Level Hierarchy

☑ Create MAC Hierarchy

Netlist Restructure Files: (Select the file to apply to this implementation)

☑ D:\Synplicity\fpga_82\examples\verilog\xilinx\test.nrf

9. Click OK to apply the implementation options.

Run Place-and-Route

The following instructions describe how to set-up the place-and-route option to run after physical synthesis has completed.

Note that the Graph-based Physical Synthesis and the Graph-based with a design plan flows automatically runs the Xilinx place-and-route tool for initial placement during synthesis and requires no setup. By default the software uses the place-and-route xilinx_gp.opt options file.

To create a final place-and-route job to run after synthesis, do the following:

1. From the Project view window, press the New P&R... button and specify the following:
   - Place-and-route implementation name. A default place-and-route name (for example, pr_1) appears in the display. Avoid using spaces in this implementation name.
   - Whether to automatically run the place-and-route implementation after the main synthesis flow.
   - Whether or not to include back annotation data from place-and-route during physical synthesis. This option is only available for Altera

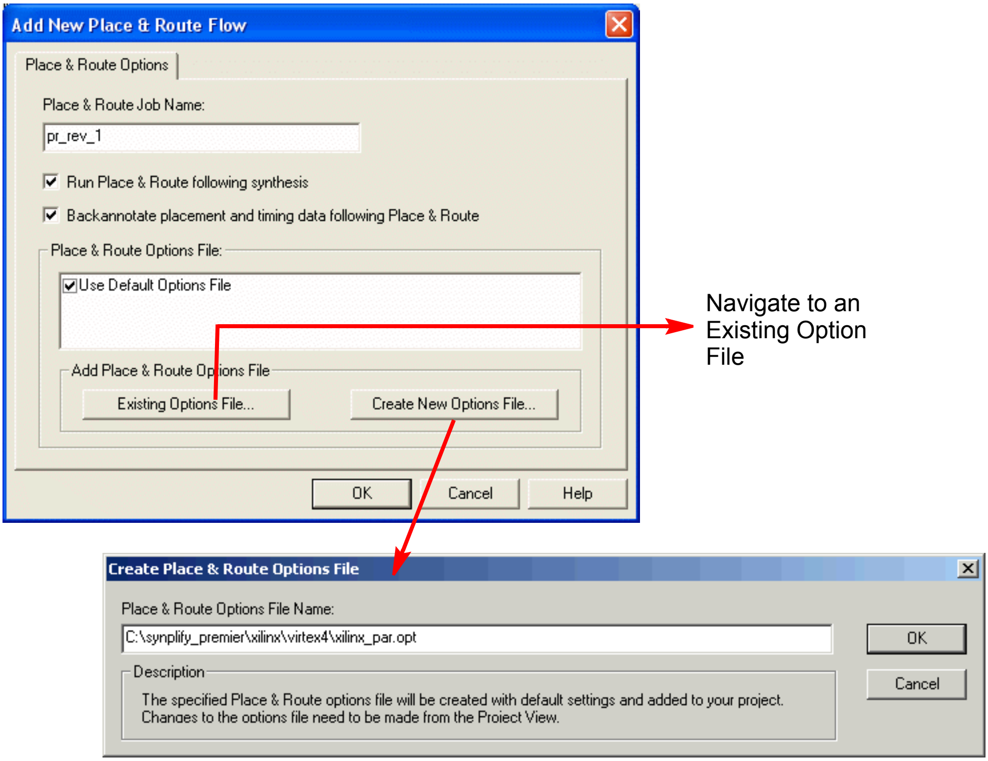
Cyclone, Cyclone-II, Stratix, Stratix-II, and Stratix-GX and Xilinx Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-4 technologies.

- Include the default place-and-route options file (`xilinx_par.opt`) or a customized options file (`.opt`) if desired. If you do not specify a user-generated file with custom place-and-route options, then a default place-and-route options file is used during physical synthesis. The customized options file is only available for certain Xilinx technologies.

You can either:

- Click on the Existing Options File button and navigate to the location of the options file.
- Otherwise, click on the Create New Options File button to create an options file.

The specified file name will be created with default settings and added to your project so that you can edit this file in the Project view.



Add New Place & Route Flow

Place & Route Job Name: pr_rev_1

☑ Run Place & Route following synthesis

☑ Backannotate placement and timing data following Place & Route

Place & Route Options File:

☑ Use Default Options File

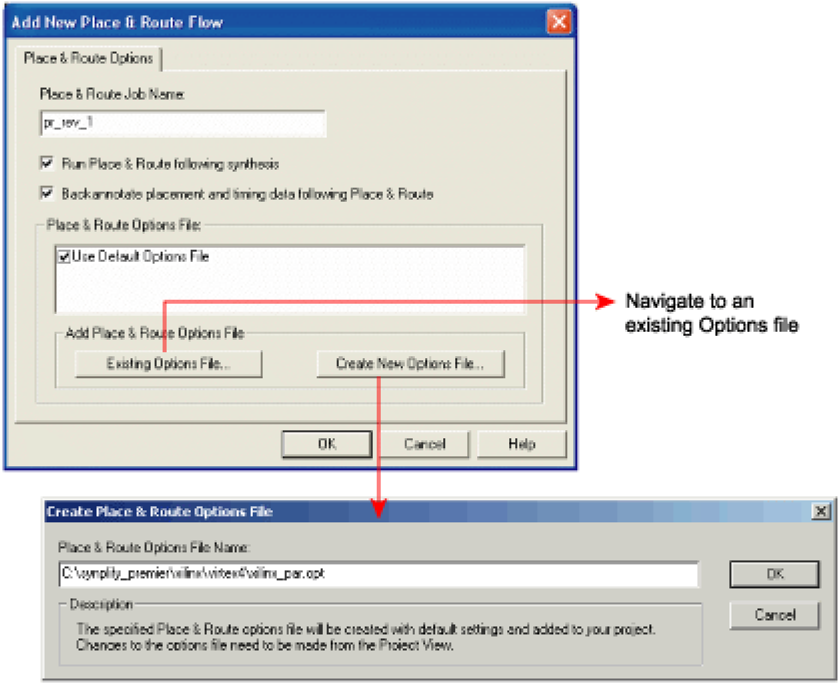
Navigate to an Existing Option File

Create Place & Route Options File

Place & Route Options File Name: C:\synplify_premier\xilinx\virtex4\xilinx_par.opt

Description

The specified Place & Route options file will be created with default settings and added to your project. Changes to the options file need to be made from the Project View.



Add New Place & Route Flow

Place & Route Options

Place & Route Job Name: pr_rev_1

☑ Run Place & Route following synthesis

☑ Backannotate placement and timing data following Place & Route

Place & Route Options File:

☑ Use Default Options File

Add Place & Route Options File

Navigate to an existing Options file

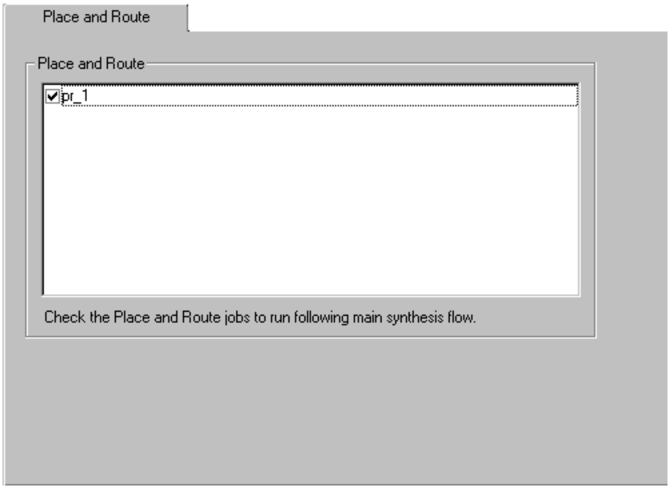
Create Place & Route Options File

Place & Route Options File Name: C:\synplify_premier\xilinx\virtex4\xilinx_par.opt

Description

The specified Place & Route options file will be created with default settings and added to your project. Changes to the options file need to be made from the Project View.

Once you create and add the place-and-route job for the implementation, this job should be enabled on the Place and Route tab of the Options for Implementation dialog box.



Place and Route

Place and Route

☑ pr_1

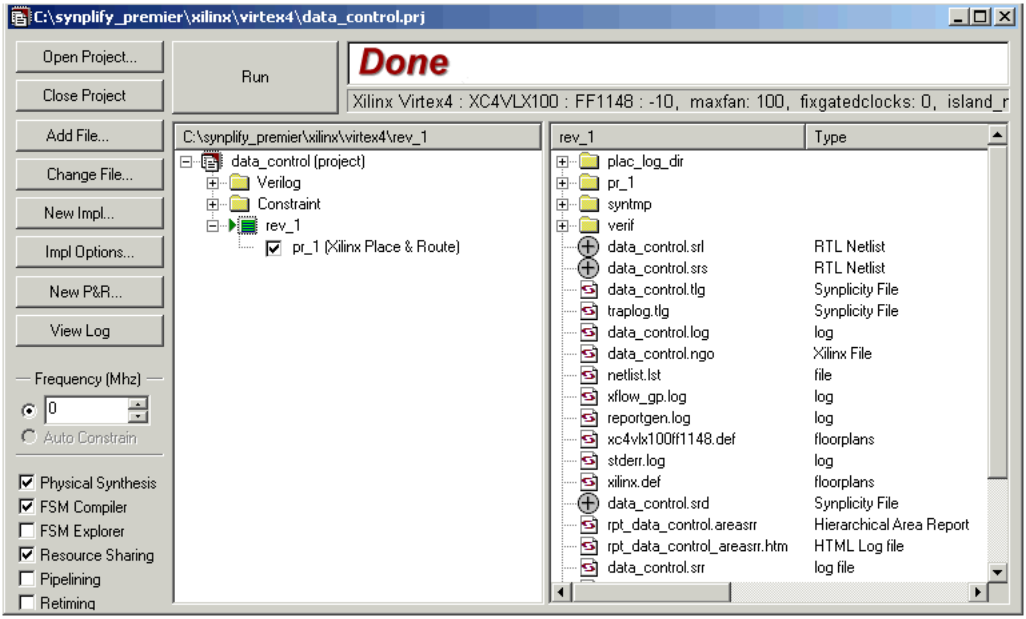
Check the Place and Route jobs to run following main synthesis flow.

For information about running place-and-route, see *Running Place-and-Route After Synthesis* on page 10-21.

2. Save the project file.

Synthesize the Design

Click Run in the Project view to start physical synthesis.



C:\synplify_premier\xilinx\virtex4\data_control.prj

Done

Xilinx Virtex4 : XC4VLX100 : FF1148 : -10, maxfan: 100, fixgatedclocks: 0, island_r

Frequency (Mhz)

● 0

○ Auto Constrain

☑ Physical Synthesis
☑ FSM Compiler
☐ FSM Explorer
☑ Resource Sharing
☐ Pipelining
☐ Retiming

C:\synplify_premier\xilinx\virtex4\rev_1

- data_control (project)
  - Verilog
  - Constraint
  - rev_1
    - ☑ pr_1 (Xilinx Place & Route)

rev_1	Type
plac_log_dir	
pr_1	
syntmp	
verif	
data_control.srl	RTL Netlist
data_control.srs	RTL Netlist
data_control.tlg	Synplicity File
traplog.tlg	Synplicity File
data_control.log	log
data_control.ngo	Xilinx File
netlist.lst	file
xflow_gp.log	log
reportgen.log	log
xc4vlx100ff1148.def	floorplans
stderr.log	log
xilinx.def	floorplans
data_control.srd	Synplicity File
rpt_data_control.areasrr	Hierarchical Area Report
rpt_data_control_areasrr.htm	HTML Log file
data_control.srr	log file

During this phase, mapping and physical synthesis are integrated. All optimizations are done with placement-aware synthesis. When physical synthesis completes, Done! appears in the status window of the Project view.

Analyze Results

You can display the physical synthesis results graphically using the HDL Analyst, Physical Analyst, and Island Timing Analyst tools.

Log File

Click the View Log button in the Project view and analyze results. This displays the log file in either text (`.srr`) or HTML (`.htm`) format. The log file contains default timing and area reports. See *Log File Command* on page 3-32 in the *Reference Manual* and *Analyze Results* on page 11-22 for information on analyzing the results.

HDL Analyst

The RTL and Technology views are schematic views used to graphically analyze your design.

To open an RTL view for a compiled design, do the following:

- Select HDL Analyst->RTL->Hierarchical View.
- Click the RTL View icon (⊕) (a plus sign inside a circle).
- Double-click the `.srs` file in the Implementation Results view.
- To open a flattened RTL view, select HDL Analyst->RTL->Flattened View.

To open a Technology view for a mapped (synthesized) design, do the following:

- Select HDL Analyst ->Technology->Hierarchical View.
- Click the Technology View icon (NAND gate icon ().
- Double-click the `.srm` file in the Implementation Results view.
- To open a flattened Technology view, select HDL Analyst-> Technology->Flattened View.

For more information about using the HDL Analyst views, see the following:

- *Basic Operations in the Schematic Views* on page 4-16
- *Exploring Design Hierarchy* on page 4-30
- *Finding Objects* on page 4-37
- *Analyzing With the HDL Analyst Tool* on page 4-56
- *Analyzing Timing* on page 4-73

Physical Analyst

The Physical Analyst tool provides a visual display of the floorplan, placement, and global routing of the design after design planning and place-and-route have been run. To display the Physical Analyst view, you can:

- 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.

The Physical Analyst view is capable of showing instances and nets. For more information, see Chapter 5, *Physical Analyst*.

Island Timing Analyst

Use the Island Timing Analyst to generate and display the Islands/Paths Summary and Details reports. You can also cross probe these critical paths to the HDL Analyst view. To invoke the Island Timing Analyst, you can either:

- Click on the Island Timing Analyst icon ( ).
- Select HDL Analyst->Island Timing Analyst from the menu.

For more information about using the Island Timing Analyst, see *The Island Timing Analyst* on page 4-88.

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 (Virtex-II Pro, Spartan-3, and Virtex-4), 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 2-22 for more information.

Index