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 or for any indirect, special or consequential damages.

Copyright Notice

Copyright © 1994-2002 Synplicity, Inc. All Rights Reserved.

Synplicity software products contain certain confidential information of Synplicity, Inc. Use of this copyright notice is precautionary and does not imply publication or disclosure. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the prior written permission of Synplicity, Inc. 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, Synplify, Amplify, Certify, Behavior Extracting Synthesis Technology, Embedded Synthesis, HDL Analyst, SCOPE, Simply Better Results, Simply Better Synthesis, and Synthesis Constraints Optimization Environment are registered trademarks, and B.E.S.T., DST, Direct Synthesis Technology, Partition-Driven Synthesis, Physical Optimizer, Total Optimization Physical Synthesis, and TOPS 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. Synopsys is a registered trademark of Synopsys, Inc. UNIX is a registered trademark of UNIX Systems Laboratories, Inc. Innoveda and Visual Elite are trademarks of Innoveda, Inc.
All other product names mentioned herein are the trademarks or registered trademarks of their respective owners.

Synplicity products are protected under U.S. Patent No. 6,182,268.

**Restricted Rights Legend**

Government Users: Use, reproduction, release, modification, or disclosure of this commercial computer software, or of any related documentation of any kind, is restricted in accordance with FAR 12.212 and DFARS 227.7202, and further restricted by the Synplicity Software License Agreement. Synplicity, Inc., 935 Stewart Drive, Sunnyvale, CA 94085, U.S.A

Printed in the U.S.A
April 2002
Synplicity Software License Agreement

Important! READ CAREFULLY BEFORE PROCEEDING

This is a legal agreement between you, the user ("Licensee") and Synplicity, Inc. ("Synplicity") regarding the software program that is attached to or enclosed with this software license agreement, or that is the subject of this documentation (the "SOFTWARE"). The term "SOFTWARE" also includes related documentation (whether in print or electronic form) and, if Licensee is obtaining an update, any pre-existing software and data provided within earlier software releases (to the extent such earlier software and data is retained by, embodied in or in any way used or accessed by the upgraded SOFTWARE provided with this Agreement). If Licensee is a participant in the University Program or has been granted an Evaluation License, then some of the following terms and conditions may not apply (refer to the sections entitled, respectively, Evaluation License and University Program, below).

By opening the packaging of the SOFTWARE, or by installing or using the SOFTWARE, Licensee agrees to be bound by the terms of this Software License Agreement (the "Agreement"). If Licensee does not agree to the terms of this Agreement, then do not install the SOFTWARE and return the copy of the SOFTWARE to the place from which you obtained it.

Evaluation License. The following applies, in addition to the other terms and conditions and as a limit on the license, if Licensee has obtained an Evaluation License. If Licensee has obtained the SOFTWARE pursuant to an evaluation license, then the following additional terms, conditions, and 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 tests and other 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 standard license fee for the commercial version of the SOFTWARE.

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. 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, 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). 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.

Copy Restrictions. This SOFTWARE is protected by United States copyright laws and international treaty provisions and copying not in accordance with this Agreement is forbidden. 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.
Preface

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. Licensee shall not sublicense, transfer or assign this Agreement or any of the rights or licenses granted under this Agreement, except in the case of a merger or sale of all or substantially all of Licensee’s assets. Permitted transfers, sublicenses or assignments under this Section continue to be subject to the rules regarding Export of the SOFTWARE.

Ownership of the SOFTWARE. Synplicity retains all right, title, and interest in the SOFTWARE (including all copies), and 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 Synplicity-supplied data, circuit and logic elements, libraries, algorithms, search strategies, rule bases, 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 into the SOFTWARE, including all intellectual property rights embodied therein. Licensee acknowledges that Synplicity is in the business of licensing SOFTWARE which incorporates Design Techniques. Licensee agrees that in the event Licensee voluntarily discloses any design techniques to Synplicity without designating such as Licensee’s Confidential Information, Synplicity has an unrestricted, royalty-free right to incorporate those Design Techniques 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 source code of the SOFTWARE, and any included trade secrets (including any technology, idea, algorithm or information contained in the SOFTWARE, and specifically including Design Techniques); (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 given to the receiving party and designated as confidential in writing within 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; or (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.

Termination. Synplicity may terminate this Agreement in the event of breach or default by Licensee. Upon
termination Licensee will relinquish all rights under this Agreement, and must cease using the SOFTWARE
and return or destroy all copies (and partial copies) of the SOFTWARE 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 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, Bureau of Export Administration’s Entity List or otherwise prohibited
from obtaining goods of this sort from the United States. Licensee shall not allow the Synplicity SOFTWARE
to be sent or used in any country except in compliance with applicable U. S. laws and regulations.

Limited Warranty and Disclaimer. Synplicity warrants that the program portion of the SOFTWARE will per-
form 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
warranties 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 that does not meet such limited warranty. In either case, Lic-
ensee 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 on duration of an implied warranty, so the above limitation
may not apply. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES OR CONDI-
tIONS, EITHER EXPRESS OR IMPLIED, ARE MADE BY SYNPLECTY WITH RESPECT TO THE SOFT-
WARE AND THE ACCOMPANYING DOCUMENTATION (STATUTORY OR OTHERWISE), AND
SYNPLECTY 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. SYNPLECTY
DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET LIC-
ENSEE'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 exclu-
sion may not apply.

Limitation of Liability. IN NO EVENT SHALL SYNPLECTY OR ITS AGENTS BE LIABLE FOR ANY INDI-
RECT, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES WHATSOEVER (INCLUDING, WITH-
OUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTIONS, LOSS OF
BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OR INABIL-
ITY TO USE THESE SYNPLECTY PRODUCTS, EVEN IF SYNPLECTY HAS BEEN ADVISED OF THE POS-
SIBILITY OF SUCH DAMAGES. In no event will Synplicity 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.** If a claim alleging infringement of an intellectual property right arises concerning the SOFTWARE (including but not limited to patent, trade secret, copyright or trademark rights), Synplicity in its sole discretion may elect to defend or settle such claim. Synplicity in the event of such a claim may also in its sole discretion elect to terminate this Agreement and all rights to use the SOFTWARE, and require the return or destruction of the SOFTWARE, with a refund of the fees paid for use of the SOFTWARE less a reasonable allowance for use and shipping.

**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 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, Bureau of Export Administration’s Entity List or otherwise prohibited from obtaining goods of this sort from the United States.

**Miscellaneous.** If Licensee is a corporation, partnership or similar entity, then the license to the SOFTWARE that is granted under this Agreement is expressly conditioned upon acceptance by a person who is authorized to sign for and bind the entity. 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. 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 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.** The Software contains commercial computer software and commercial computer software documentation. In accordance with FAR 12.212 and DFARS 227.7202, use, duplication or disclosure is subject to restrictions under paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013, and further restricted by this Agreement. Synplicity, Inc., 935 Stewart Drive, Sunnyvale, CA 94085, U. S. A.

**SCF Reader.** The following terms and conditions apply only to the SCF reader, which provides the functionality to support input of designs in SCF format (the "Reader"), and which may be included with the Software. Such terms are in addition to the other terms in this Agreement. With respect to the Reader, Synplicity disclaims, on its own behalf and on behalf of any of its licensors or suppliers who have contributed to the Reader, any and all warranties and conditions, express or implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose. Synplicity hereby excludes, on its own behalf and on behalf of any of its licensors or suppliers who have contributed to the Reader, any and all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits. The foregoing applies notwithstanding the type of claim (whether contract, tort or otherwise) and notwithstanding the failure of essential purpose of any limited remedy. If there are any terms in this Agreement regarding the Reader that differ from those terms on which Synplicity obtained the Reader, then those terms are offered by Synplicity and not by any other party. If you would like a source code version of the Reader, please contact Synplicity at support@synplicity.com, and Synplicity will make it available on reasonable terms.

**University Program.** The following section applies only if Licensee is a participant in Synplicity's University
Preface

*Program; it does not replace the remainder of the Agreement and supersedes only those terms that directly conflict.*

**University Program: License.** Subject to the terms and conditions of this Agreement, Synplicity hereby grants to Licensee (a University) for the License Term (defined below), a non-exclusive license, only for purposes of course work or teaching in connection with a university-sponsored class, or for academic research either sponsored by or conducted under the auspices of Licensee, to (a) install and use the SOFTWARE, and (b) reproduce and distribute copies of the documentation included in the SOFTWARE subject only to payment for those copies (which may be based on the number of users, the number and type of copies, or both). If the SOFTWARE is licensed pursuant to a node-locked license, then the Licensee may install and use the SOFTWARE on the authorized workstations. If the SOFTWARE is licensed pursuant to a floating license, then the Licensee may install the SOFTWARE on the authorized server and use the SOFTWARE on up to the number of nodes for which Licensee has paid license fees and Synplicity has granted authorization.

**University Program: License Term and Termination.** For purposes of the University Program, “License Term” means one year unless otherwise agreed to in writing. This Agreement will terminate at the end of the License Term, unless earlier terminated in accordance with this Agreement.

**University Program: License Restrictions.** As Licensee, University may not (i) allow access to the SOFTWARE by any user not registered for a course or participating in an academic research project for which use of the SOFTWARE has been authorized; (ii) use the SOFTWARE to design any commercial products; or (iii) 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.

**University Program: Technical Liaison.** Licensee shall appoint a Technical Liaison who will serve as the single point of contact between Synplicity and Licensee with respect to the subject matter of this Agreement. The Technical Liaison will coordinate installation and maintenance of the SOFTWARE, communicate with Synplicity regarding license procedures, administer Licensee's obligations under this Agreement and respond to inquiries by Synplicity related to the subject matter of this Agreement.

**University Program: Technical Support in North America.** Unless otherwise agreed in writing, Synplicity will accept calls only from the appointed Technical Liaison. No technical support will be provided other than calls from the Technical Liaison relating to installation of the SOFTWARE. SOFTWARE upgrades may be obtained from the Synplicity Web Site.

**University Program: International Technical Support.** Technical support is provided through Synplicity's authorized distributors in accordance with their applicable policies.

revised 10/01
## Contents

### Chapter 1: Product Overview

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licensing</td>
<td>1-2</td>
</tr>
<tr>
<td>Getting Help</td>
<td>1-3</td>
</tr>
<tr>
<td>Finding Information</td>
<td>1-3</td>
</tr>
<tr>
<td>Contacting Customer Support</td>
<td>1-4</td>
</tr>
<tr>
<td>The Synplicity Products</td>
<td>1-5</td>
</tr>
<tr>
<td>Synplicity® Product Family</td>
<td>1-5</td>
</tr>
<tr>
<td>Product Audience</td>
<td>1-6</td>
</tr>
<tr>
<td>Overview of the Synplify Synthesis Tool</td>
<td>1-7</td>
</tr>
<tr>
<td>B.E.S.T. Technology</td>
<td>1-7</td>
</tr>
<tr>
<td>Synthesis Tool Features</td>
<td>1-7</td>
</tr>
<tr>
<td>Supported Platforms</td>
<td>1-8</td>
</tr>
<tr>
<td>User Interface</td>
<td>1-8</td>
</tr>
<tr>
<td>Projects, Implementations and Workspaces</td>
<td>1-11</td>
</tr>
<tr>
<td>Starting the Synthesis Tool</td>
<td>1-12</td>
</tr>
<tr>
<td>Starting the Synthesis Tool in Interactive Mode</td>
<td>1-12</td>
</tr>
<tr>
<td>Syntax for the synplify Command</td>
<td>1-13</td>
</tr>
<tr>
<td>Using the Synplify Synthesis Tool</td>
<td>1-14</td>
</tr>
<tr>
<td>Synthesis Software Flow</td>
<td>1-14</td>
</tr>
<tr>
<td>Synthesizing Your Design</td>
<td>1-15</td>
</tr>
<tr>
<td>Optimizing for Best Performance</td>
<td>1-17</td>
</tr>
<tr>
<td>Optimizing for Area</td>
<td>1-19</td>
</tr>
<tr>
<td>Setting Area/Delay Trade-offs</td>
<td>1-19</td>
</tr>
<tr>
<td>Setting Fanout Limits</td>
<td>1-20</td>
</tr>
<tr>
<td>FSM Compiler</td>
<td>1-21</td>
</tr>
<tr>
<td>What FSM Compiler Does</td>
<td>1-21</td>
</tr>
<tr>
<td>When to Use FSM Compiler</td>
<td>1-22</td>
</tr>
<tr>
<td>Where to Use FSM Compiler (Global vs. Local Use)</td>
<td>1-22</td>
</tr>
</tbody>
</table>
Files .............................................................. 1-23
  Project Files .................................................. 1-23
  HDL Source Files ............................................. 1-23
  Constraint Files ............................................. 1-25
  Output Files .................................................. 1-26
Reports and Messages ........................................ 1-27
  Log File ....................................................... 1-27
  Timing Report ................................................. 1-28
  Net Buffering Report ....................................... 1-29
  Resource Usage Report ..................................... 1-29
  Informational Files ......................................... 1-30
  Errors, Warnings, Notes and Information ................ 1-30

Chapter 2: User Interface Overview

Windows and Views .......................................... 2-2
  Project Window ............................................... 2-2
  RTL View ...................................................... 2-4
  Technology View ............................................. 2-5
  Text Editor View ............................................ 2-7
Using the Interface Effectively ......................... 2-10
  Using the Mouse ............................................. 2-10
  Setting Project View Display Preferences ............ 2-12
  Managing Views ............................................. 2-13
  Setting Text Editor Window Preferences ............. 2-14
  Customizing the RTL and Technology Views ......... 2-14

Chapter 3: Menus, Dialog Boxes . .

Command Access ............................................... 3-2
  Menubar .......................................................... 3-2
  Context-sensitive Popup Menus ............................ 3-2
  Toolbars .......................................................... 3-2
  Keyboard Shortcuts ......................................... 3-3
  Action Buttons ............................................... 3-3
  Tcl Command Equivalents .................................. 3-3
Menubar Menus ................................................... 3-4
  File Menu .......................................................... 3-4
  Edit Menu .......................................................... 3-16
  View Menu .......................................................... 3-22
  Project Menu ..................................................... 3-29
  Run Menu .......................................................... 3-42
  HDL Analyst Menu ............................................. 3-45
Menubar Menus (continued)
Format Menu .................................................. 3-51
Options Menu .................................................. 3-53
Window Menu .................................................. 3-62
Help Menu .................................................... 3-64

Popup Menus .................................................. 3-73
Project View Popup Menu .................................. 3-73
SCOPE Popup Menu ......................................... 3-84
Text Editor Popup Menu .................................... 3-84
RTL View and Technology View Popup Menus ............. 3-85

Toolbars ..................................................... 3-88
Project Toolbar ............................................. 3-88
Analyst Toolbar ............................................. 3-90
View Toolbar ................................................ 3-91
Edit Toolbar ............................................... 3-92

Keyboard Shortcuts ....................................... 3-93
Action Buttons and Options ............................... 3-97

Chapter 4: Tcl Commands and Scripts
Introduction to Tcl .......................................... 4-2
Tcl Conventions ............................................. 4-2
Creating Tcl Scripts ....................................... 4-3
Starting in Batch Mode Using a Project File ............... 4-4
Executing Tcl Scripts ...................................... 4-5
Including Command Files in a Project .................... 4-5

Batch Mode ................................................. 4-6
Starting in Batch Mode Using a Project File ............... 4-6
Starting in Batch Mode Using a Tcl Script ................. 4-7

Tcl Commands for Synthesis ................................ 4-9
add_file .................................................. 4-9
constraint_file ........................................... 4-10
get_env .................................................... 4-11
get_option ................................................ 4-12
project ..................................................... 4-12
project_file ............................................. 4-14
set_option ............................................... 4-15
Vendor-Specific Tcl Commands ............................ 4-20

Tcl Script Examples ....................................... 4-21
Chapter 5: Using HDL Analyst

About HDL Analyst ................................................................. 5-2
HDL Analyst Views .............................................................. 5-2
HDL Analyst Commands ...................................................... 5-3

Getting Started with HDL Analyst ........................................ 5-4
Setting Up HDL Analyst for Synplify ..................................... 5-4
Viewing Object Information in HDL Analyst .......................... 5-4
Finding Objects by Name ..................................................... 5-5
Finding Objects ................................................................. 5-6
Selecting Objects in HDL Analyst ......................................... 5-6
Setting Colors in HDL Analyst ............................................. 5-7
Changing Amount of Logic Displayed ................................. 5-7

Crossprobing ................................................................. 5-8
Crossprobing Across Views .............................................. 5-8
Crossprobing to Visual Elite ............................................. 5-8
Crossprobing with ModelSim ............................................. 5-11

Traversing Hierarchy ......................................................... 5-12
Viewing Objects Across Design Levels and Views .................. 5-12
Using the Hierarchy Browser ............................................ 5-13
Using Push/Pop Mode ....................................................... 5-13

Multisheet Schematics in HDL Analyst ................................. 5-15
Sheet Connectors ............................................................ 5-15
Viewing Other Sheets ....................................................... 5-15

Filtering Schematics in HDL Analyst .................................. 5-17

Analyzing Critical Paths in HDL Analyst ............................... 5-18
Viewing Critical Paths in HDL Analyst ................................. 5-18
Interpreting Critical Path Information .................................. 5-19
Handling Negative Slack ................................................. 5-21
Strategies ................................................................. 5-23

Chapter 6: Timing Constraints

About Timing Constraints .................................................. 6-2
Specifying Timing Constraints ........................................... 6-2

SCOPE Window .............................................................. 6-3
SCOPE Spreadsheet ......................................................... 6-3
Opening a Constraint File with the SCOPE Spreadsheet ........... 6-4
SCOPE Constraint Types .................................................. 6-5
Clocks Panel ................................................................. 6-6
Inputs/Outputs Panel ....................................................... 6-10
<table>
<thead>
<tr>
<th>SCOPE Window (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers Panel</td>
</tr>
<tr>
<td>Multi-Cycle Paths Panel</td>
</tr>
<tr>
<td>False Paths Panel</td>
</tr>
<tr>
<td>Attributes Panel</td>
</tr>
<tr>
<td>Other Panel</td>
</tr>
<tr>
<td>SCOPE File Conversions</td>
</tr>
<tr>
<td>SCOPE Constraint Wizards</td>
</tr>
<tr>
<td>Tcl Constraint Files</td>
</tr>
<tr>
<td>Guidelines for Creating Constraint Files</td>
</tr>
<tr>
<td>Object Naming Syntax</td>
</tr>
<tr>
<td>Individual Timing Constraints</td>
</tr>
<tr>
<td>define_clock</td>
</tr>
<tr>
<td>define_reg_input_delay</td>
</tr>
<tr>
<td>define_reg_output_delay</td>
</tr>
<tr>
<td>define_input_delay</td>
</tr>
<tr>
<td>define_output_delay</td>
</tr>
<tr>
<td>define_multicycle_path</td>
</tr>
<tr>
<td>define_false_path</td>
</tr>
<tr>
<td>syn_reference_clock</td>
</tr>
<tr>
<td>Black-Box Timing Models</td>
</tr>
<tr>
<td>Black-box Source Code Directives</td>
</tr>
<tr>
<td>Forward Annotation</td>
</tr>
<tr>
<td>Altera Max+Plus II</td>
</tr>
<tr>
<td>Altera Quartus II</td>
</tr>
<tr>
<td>Lattice ORCA</td>
</tr>
<tr>
<td>Xilinx</td>
</tr>
<tr>
<td>Support for DCM/DLL in Xilinx</td>
</tr>
<tr>
<td>Timing Report</td>
</tr>
<tr>
<td>Timing Report Header</td>
</tr>
<tr>
<td>Performance Summary</td>
</tr>
<tr>
<td>Clock Relationships</td>
</tr>
<tr>
<td>Interface Information</td>
</tr>
<tr>
<td>Detailed Clock Report</td>
</tr>
</tbody>
</table>
## Chapter 7: Synthesis Attributes and Directives

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes and Directives: Summary</td>
<td>7-2</td>
</tr>
<tr>
<td>Attributes and Directives: Differences, Specifying</td>
<td>7-2</td>
</tr>
<tr>
<td>Attribute and Directive Summary by Vendor</td>
<td>7-3</td>
</tr>
<tr>
<td>Attribute Summary (Alphabetical)</td>
<td>7-3</td>
</tr>
<tr>
<td>Directive Summary (Alphabetical)</td>
<td>7-7</td>
</tr>
<tr>
<td>Specifying Attributes in a Constraint File</td>
<td>7-9</td>
</tr>
<tr>
<td>Specifying Attributes with the SCOPE Spreadsheet</td>
<td>7-9</td>
</tr>
<tr>
<td>SCOPE Attributes Wizard</td>
<td>7-12</td>
</tr>
<tr>
<td>Specifying Directives (and Attributes) in HDL</td>
<td>7-14</td>
</tr>
<tr>
<td>Verilog Attribute and Directive Syntax</td>
<td>7-15</td>
</tr>
<tr>
<td>VHDL Attribute and Directive Syntax</td>
<td>7-16</td>
</tr>
<tr>
<td>Attributes</td>
<td>7-18</td>
</tr>
<tr>
<td>alslc</td>
<td>7-19</td>
</tr>
<tr>
<td>alspin</td>
<td>7-21</td>
</tr>
<tr>
<td>alspreserve</td>
<td>7-23</td>
</tr>
<tr>
<td>altera_auto_use_eab</td>
<td>7-25</td>
</tr>
<tr>
<td>altera_auto_use_esb</td>
<td>7-28</td>
</tr>
<tr>
<td>altera_chip_pin_lc</td>
<td>7-31</td>
</tr>
<tr>
<td>altera_implement_in_eab</td>
<td>7-33</td>
</tr>
<tr>
<td>altera_implement_in_esb</td>
<td>7-35</td>
</tr>
<tr>
<td>altera_io_opendrain</td>
<td>7-37</td>
</tr>
<tr>
<td>altera_io_powerup</td>
<td>7-39</td>
</tr>
<tr>
<td>din</td>
<td>7-41</td>
</tr>
<tr>
<td>dout</td>
<td>7-42</td>
</tr>
<tr>
<td>loc</td>
<td>7-43</td>
</tr>
<tr>
<td>lock</td>
<td>7-44</td>
</tr>
<tr>
<td>orca_padtype</td>
<td>7-45</td>
</tr>
<tr>
<td>orca_props</td>
<td>7-47</td>
</tr>
<tr>
<td>ql_padtype</td>
<td>7-49</td>
</tr>
<tr>
<td>ql_placement</td>
<td>7-51</td>
</tr>
<tr>
<td>syn_direct_enable</td>
<td>7-53</td>
</tr>
<tr>
<td>syn_edif_bit_format</td>
<td>7-55</td>
</tr>
<tr>
<td>syn_edif_name_length</td>
<td>7-58</td>
</tr>
<tr>
<td>syn_edif_scalar_format</td>
<td>7-60</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>7-62</td>
</tr>
<tr>
<td>syn_forward_io_constraints</td>
<td>7-65</td>
</tr>
<tr>
<td>syn_hier</td>
<td>7-67</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>7-70</td>
</tr>
<tr>
<td>syn_multstyle</td>
<td>7-72</td>
</tr>
<tr>
<td>syn_netlist_hierarchy</td>
<td>7-74</td>
</tr>
</tbody>
</table>

Synplify Reference Manual, April 2002
Attributes (continued)

- **syn_noarrayports** .................................................. 7-75
- **syn_noclockbuf** .................................................. 7-76
- **syn_pipeline** ..................................................... 7-78
- **syn_preserve_sr_priority** ...................................... 7-81
- **syn_props** .......................................................... 7-83
- **syn_radhardlevel** .................................................. 7-85
- **syn_ramstyle** ........................................................ 7-87
- **syn_reference_clock** .............................................. 7-91
- **syn_replicate** ...................................................... 7-92
- **syn_resources** ........................................................ 7-95
- **syn_romstyle (Altera)** ............................................. 7-97
- **syn_romstyle (Xilinx)** ............................................. 7-100
- **syn_srlstyle** .......................................................... 7-102
- **syn_tristatetomux** ................................................ 7-104
- **syn_useenables** ....................................................... 7-106
- **syn_useioff (Altera)** ............................................. 7-108
- **syn_useioff (Lattice ORCA)** .................................... 7-109
- **syn_useioff (QuickLogic)** ...................................... 7-112
- **syn_useioff (Xilinx)** ............................................. 7-114
- **xc_alias** .............................................................. 7-116
- **xc_clockbuftype** .................................................. 7-117
- **xc_fast** ............................................................... 7-119
- **xc_fast_auto** ......................................................... 7-121
- **xc_global_buffers** ................................................... 7-123
- **xc_loc** ................................................................. 7-124
- **xc_map** ................................................................. 7-125
- **xc_ncf_auto_relax** .................................................. 7-126
- **xc_nodelay** ............................................................ 7-128
- **xc_padtype** ............................................................ 7-129
- **xc_props** ............................................................... 7-131
- **xc_pullup / xc_pulldown** ......................................... 7-132
- **xc_rloc** ................................................................. 7-134
- **xc_slow** ............................................................... 7-136
- **xc_uset** ............................................................... 7-137

Directives .............................................................. 7-139

- **black_box_pad_pin** .................................................. 7-140
- **black_box_tri_pins** ................................................. 7-142
- **full_case** ............................................................. 7-144
- **parallel_case** .......................................................... 7-147
- **syn_black_box** ....................................................... 7-149
- **syn_enum_encoding** .................................................. 7-151
- **syn_isclock** .......................................................... 7-155
Directives (continued)
  syn_keep ........................................ 7-157
  syn_macro ...................................... 7-161
  syn_noprune .................................... 7-162
  syn_preserve ................................... 7-167
  syn_sharing .................................... 7-172
  syn_state_machine .............................. 7-173
  syn_tco<n> ..................................... 7-177
  syn_tpd<n> ..................................... 7-180
  syn_tristate ................................... 7-183
  syn_tsu<n> ..................................... 7-184
  translate_off/translate_on ...................... 7-187
  xc_isgsr ...................................... 7-189

Chapter 8: Verilog Language Support

Language Constructs ........................................ 8-2
  Supported Verilog Language Constructs ............ 8-2
  Unsupported Language Constructs ................... 8-3
  Ignored Language Constructs ........................ 8-3
  Verilog 2001 Support ................................ 8-3

Verilog Synthesis Guidelines ............................ 8-7
  General Synthesis Guidelines ....................... 8-7
  Verilog Language Guidelines ........................ 8-8

Verilog Module Template .................................. 8-9

Scalable Modules .......................................... 8-11
  Creating a Scalable Module ......................... 8-11
  Using Scalable Modules .............................. 8-11

Built-in Gate Primitives .................................. 8-14

Combinational Logic ..................................... 8-15
  Combinational Logic Examples ...................... 8-15
  always Blocks for Combinational Logic .......... 8-16
  Continuous Assignments for Combinational Logic . 8-18

Sequential Logic ........................................ 8-19
  Sequential Logic Examples .......................... 8-19
  Flip-flops Using always Blocks ................... 8-20
  Level-sensitive Latches ............................ 8-21
  Sets and Resets .................................... 8-24
VHDL Language Constructs (continued)
Sequential Logic ................................................. 9-24
Component Instantiation in VHDL .............................. 9-24
VHDL Selected Name Support ................................. 9-26
User-defined Function Support ............................... 9-30
Demand Loading ....................................................... 9-32
VHDL Synthesis Guidelines ..................................... 9-33
General Synthesis Guidelines ................................... 9-33
VHDL Language Guidelines ................................... 9-33
Model Template ......................................................... 9-34
Creating Flip-flops and Registers Using VHDL Processes .................. 9-36
Clock Edges ................................................................. 9-38
Defining an Event Outside a Process ......................... 9-39
Using a WAIT Statement Inside a Process ................... 9-39
Level-sensitive Latches Using Concurrent Signal Assignments ............ 9-40
Level-sensitive Latches Using VHDL Processes .................. 9-41
Sets and Resets ........................................................... 9-44
Asynchronous Sets and Resets .................................... 9-44
Synchronous Sets and Resets ..................................... 9-46
State Machines .......................................................... 9-49
State Machine Guidelines ...................................... 9-49
Using Enumerated Types for State Values .................... 9-52
Simulation Tips When Using Enumerated Types .................. 9-53
Asynchronous State Machines in VHDL ....................... 9-54
Hierarchical Designs ................................................ 9-56
Creating a Hierarchical Design in VHDL ...................... 9-56
Configuration Specification and Declaration ................... 9-60
Configuration Specification ....................................... 9-60
Configuration Declaration ........................................... 9-64
Scalable Designs ....................................................... 9-71
Creating Scalable Designs ....................................... 9-71
Creating a Scalable Design Using Unconstrained Vector Ports .......... 9-71
Creating a Scalable Design Using VHDL Generics ............... 9-72
Using a Scalable Architecture .................................... 9-73
Creating a Scalable Design Using Generate Statements .......... 9-75
RAM Inference ............................................................ 9-77
RAMs with Special Write Enables ............................... 9-78
ROM Inference ............................................................ 9-82
Synthesis Attributes and Directives ............................ 9-85
Instantiating Black Boxes in VHDL ............................... 9-86
Black-Box Timing Constraints ........................................ 9-86
Instantiating a Black Box ............................................. 9-87
Instantiating a Technology-vendor I/O ......................... 9-87

VHDL Synthesis Examples ........................................... 9-89
Combinational Logic Examples ..................................... 9-89
Sequential Logic Examples ........................................... 9-90

PREP VHDL Benchmarks .............................................. 9-91

Appendix A: Designing with Actel

Using Actel .......................................................... A-2
Actel Features ......................................................... A-2
Constraint Files ....................................................... A-2
Synthesis Reports ..................................................... A-3
Actel Device Mapping Options ...................................... A-3
Instantiating Macros and Black Boxes in Actel Designs ...... A-5

Actel-specific Tcl Command Options ............................... A-8
set_option Command for Actel ..................................... A-8

Actel Attribute and Directive Summary .......................... A-10

Appendix B: Designing with Altera

Altera Device-specific Support ...................................... B-2

General Guidelines ................................................... B-2
Specifying Constraints and Attributes to Obtain Quality Results B-3
Instantiating Verilog and VHDL Macros/Black Boxes ........ B-3
Instantiating LPMs in the Source Code .......................... B-3
Inferring General-purpose RAMs ................................... B-4
Inferring General-purpose ROMs ................................... B-6
Automatic Resource Management for ESBs/EABs ............... B-8
Place-and-route Tools and EDIF / VQM Filenames ............... B-9
Outputting Port Names for the Place-and-route Tools ........ B-9
Inferring Registers with the Compiler ............................ B-9
Mapping to ATOM Primitives ....................................... B-10
Selecting Implementation Options ................................. B-10
Viewing Synthesis Reports ......................................... B-11
Crossprobing with Quartus II ....................................... B-13
Appendix C: Designing with Atmel

General Guidelines .................................................. C-2
  Supported Atmel Device Families .............................. C-2
  Obtaining Quality Results Using Constraint Files .......... C-2
  Atmel Device Mapping Options ................................. C-2
Atmel-specific Tcl Command Options ............................ C-4
  set_option Command for Atmel ................................. C-4

Atmel Attribute and Directive Summary ........................ C-5

Appendix D: Designing with Cypress

Overview ................................................................. D-2
  Supported Cypress Device Families .......................... D-2
  Obtaining Quality Results Using Constraint Files .......... D-2
  Instantiating LPMs ................................................. D-2
Cypress-specific Options ........................................... D-11
  Cypress Device Mapping Options ............................... D-11
  set_option Command for Cypress ............................... D-11
Cypress Attribute and Directive Summary ........................ D-12
  Attribute and Directive .......................................... D-12

Appendix E: Designing with Lattice

Overview ................................................................. E-2
  Supported Lattice Device Families ............................ E-2
  Synthesis Output ................................................ E-2
  Constraint Files ................................................ E-2
  Instantiating Macros and Black Boxes ........................ E-2
Lattice ORCA Devices ................................................ E-4
  Device Mapping Options (ORCA) ............................... E-4
  set_option Command for Lattice ORCA ........................ E-6
  Inferring Registers with the Compiler ........................ E-6
  Forward annotation .............................................. E-8
  Supported Timing Constraints .................................. E-9
  Synthesis Reports ................................................. E-10
Lattice GAL and isp Devices ...................................... E-11
  Device Mapping Options (GAL and isp) ....................... E-11
  set_option Command for GAL and isp ........................ E-11
  GAL and isp Attributes ........................................... E-12
MACH Devices ................................................................. E-12
  Device Mapping Options (MACH) ..................................... E-12
  MACH Tcl Command Options ........................................ E-13
  MACH Attributes ...................................................... E-15

Lattice Attribute and Directive Summary ............................ E-16

Appendix F: Designing with QuickLogic

Synthesizing with QuickLogic ........................................... F-2
  Design Guidelines ..................................................... F-2
  Device Mapping Options (QuickLogic) ............................... F-3
  Running the Synthesis Tool From SpDE ............................ F-4

QuickLogic-specific Tcl Command Options ............................ F-6
  set_option Command for QuickLogic ............................... F-6

QuickLogic Attribute and Directive Summary ........................ F-7

QuickLogic SpDE Commands ............................................. F-9
  The .scp Command File ................................................ F-9
  QuickLogic portprop Command ...................................... F-9
  QuickLogic instprop Command ...................................... F-10
  QuickLogic include Directive ....................................... F-11

Appendix G: Designing with Triscend

Synthesizing Triscend Designs ......................................... G-2
  Supported Triscend Device Families ............................... G-2
  Obtaining Quality Results Using Constraint Files ............... G-2
  Device Mapping Options (Triscend) ............................... G-2

Triscend-specific Tcl Command Options ............................... G-3
  set_option Command for Triscend ................................. G-3

Triscend Attribute and Directive Summary ........................... G-4

Appendix H: Designing with Xilinx

Supported Xilinx Devices .............................................. H-2

Device Mapping Options .............................................. H-2
  Fanout Guide .......................................................... H-2
  Disable I/O Insertion ................................................ H-3
  Force GSR Usage ....................................................... H-4

Design Considerations ................................................ H-5
  Specifying Constraints and Attributes ........................... H-5
  Creating Mixed Schematic and HDL Designs ....................... H-5
Design Considerations (continued)

- Instantiating Startup Blocks ........................................ H-6
- Packing IOBs Using syn_useioff .................................... H-7
- Instantiating Macros and Black Boxes ............................. H-7
- Forward-annotating Timing Constraints to Placement and Routing H-8
- Viewing Resource Usage Reports ................................. H-9

Design Elements ......................................................... H-10
- Inferring RAMs ......................................................... H-10
- Mapping ROM into Block RAM ..................................... H-13
- Inferring Registers with the Compiler ........................... H-14
- Mapping Latches ....................................................... H-15
- Inferring Dynamic Shift Register Lookup (SRL) Tables ........ H-15

I/Os and Pin Locations .................................................. H-18
- Specifying Pin Locations and Inserting I/Os .................... H-18
- Specifying Relative Location .................................... H-25
- Controlling Port Names in EDIF ..................................... H-25

Using Spartan ........................................................... H-27
- Device Mapping Options (Spartan) ................................. H-27
- set_option Command for Spartan .................................. H-27
- project Command for Spartan ...................................... H-28

Using Virtex ............................................................. H-30
- Device Mapping Options (Virtex) ................................ H-30
- set_option Command for Virtex .................................. H-30
- project Command for Virtex ...................................... H-31
- Resource Usage ......................................................... H-32
- Interfacing with Xilinx Place-and-route Tools ................. H-32

Using XC4000 ............................................................. H-34
- Device Mapping Options (XC4000) ............................... H-34
- set_option Command for XC4000 ................................. H-34
- project Command for XC4000 .................................... H-35

Using XC3000 and XC5200 ............................................. H-37
- Device Mapping Options (XC3000 and XC5200) ................. H-37
- set_option Command for XC3000 and XC5200 ................ H-37
- project Command for XC3000 and XC5200 ..................... H-38

Using Xilinx CLPDs ..................................................... H-40
- Device Mapping Options (CPLDs) ................................ H-40
- set_option Command for Xilinx CPLDs .......................... H-40
- project Command for Xilinx CPLDs .............................. H-41

Xilinx Attribute and Directive Summary ............................ H-43
CHAPTER 1

Product Overview

This reference manual is part of a document set that also includes the Synplify User Guide. The reference manual describes the Synplify synthesis tool user interface, commands, and features. The User Guide contains “how-to” information on performing user tasks.

This chapter provides an introduction to Synplicity’s product family and the Synplify synthesis tool.

- Licensing on page 1-2
- Getting Help on page 1-3
- The Synplicity Products on page 1-5
- Overview of the Synplify Synthesis Tool on page 1-7
- Starting the Synthesis Tool on page 1-12
- Using the Synplify Synthesis Tool on page 1-14
- FSM Compiler on page 1-21
- Files on page 1-23
- Reports and Messages on page 1-27
Chapter 1: Product Overview

Licensing

For the most current and comprehensive licensing information, refer to the licensing and installation instructions that came with the Synplify synthesis software.

The following table summarizes common licensing tasks:

Table 1-1: Common licensing tasks

<table>
<thead>
<tr>
<th>To...</th>
<th>Do...</th>
</tr>
</thead>
<tbody>
<tr>
<td>View, enter, or change licensing information</td>
<td>Choose Help -&gt; License Wizard (see License Wizard Dialog Boxes on page 3-67). Choose Edit or enter license information received from Synplicity at the bottom of the dialog box, then click Next. Follow the instructions.</td>
</tr>
<tr>
<td>Request a trial license</td>
<td>Choose Help -&gt; License Wizard (see License Wizard Dialog Boxes on page 3-67). Choose Fill out a form to request a trial license from Synplicity, then click Next. Follow the instructions.</td>
</tr>
<tr>
<td>Check floating licenses in use</td>
<td>Choose Help -&gt; Floating License Usage to view the number of floating licenses currently being used, and their users.</td>
</tr>
<tr>
<td>Select a default license</td>
<td>Choose Help -&gt; Select Preferred License, then choose one of the licenses. Enable Save as default license type. Restart the Synplify synthesis tool. See Select Preferred License Dialog Box on page 3-71.</td>
</tr>
</tbody>
</table>
Getting Help

This section shows you where to find information and how to contact customer support.

Finding Information

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

Table 1-2: Getting help with Synplicity products

<table>
<thead>
<tr>
<th>For help with...</th>
<th>Refer to the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licensing</td>
<td>Copyright and License Agreement and the appropriate License Configuration and Setup document for your platform</td>
</tr>
<tr>
<td>Language and syntax</td>
<td>Synplify Reference Manual</td>
</tr>
<tr>
<td>Attributes and directives</td>
<td>Synplify Reference Manual</td>
</tr>
<tr>
<td>Tcl language</td>
<td>Online help (Help -&gt; Tcl Help)</td>
</tr>
<tr>
<td>Synthesis Tcl commands</td>
<td>Synplify Reference Manual</td>
</tr>
<tr>
<td>Using tool-specific features and attributes</td>
<td>Synplify User Guide</td>
</tr>
<tr>
<td>How to...</td>
<td>Synplify User Guide, and various application notes available on the Synplicity support web site</td>
</tr>
<tr>
<td>Flow information</td>
<td>Synplify User Guide and various application notes available on the Synplicity support web site</td>
</tr>
</tbody>
</table>
Contacting Customer Support

If you have a problem, question, or enhancement request, use one of these methods to contact us:

• E-mail us at support@synplicity.com

• Call us at (U.S.) +1 408 215-6000

• Write to us at this address:
  
  Synplicity, Inc.
  935 Stewart Drive
  Sunnyvale, CA 94085, USA

• FAX us at (U.S.) +1 408 990-0290
The Synplicity Products

Synplicity® Product Family

Synplicity’s products are based on core logic synthesis technology, and share a common look and feel.

- The Synplify and Synplify Pro products are logic synthesis tools for FPGAs (field programmable gate arrays) and CPLDs (complex programmable logic devices). The Synplify Pro tool is an advanced version of the Synplify tool, with many additional features.

  These tools accept high-level designs written in industry-standard hardware description languages (Verilog and VHDL) and, using Synplicity’s Behavior Extracting Synthesis Technology® (B.E.S.T.™), convert the designs into small, high performance design netlists for popular technology vendors. They can also write VHDL and Verilog netlists after synthesis, which you can then simulate in order to verify functionality.

- The Amplify Physical Optimizer™ product is the first physical synthesis tool for FPGAs, and is a separately licensed option to the Synplify Pro product. You use it to interactively assign physical constraints to a design, by dragging and dropping RTL objects into regions of the design. With this constraint information, the physical optimizer can derive more accurate timing estimates and use them to
perform additional optimizations, producing a more highly optimized circuit in fewer iterations.

- The Synplicity ASIC® product is an ASIC synthesis tool that is optimized for designer productivity. It is targeted to ASIC designers who generate gate-level netlists from RTL source code, and FPGA designers who are migrating toward ASIC designs by converting existing FPGA designs into ASIC designs. It offers single-pass ASIC synthesis of designs with up to 2 million gates, the development of cores and IP, and reoptimization of existing designs for die size reduction and performance improvements.

- The Certify® product is an electronic design automation (EDA) tool for system-design, system-on-a-chip (SoC), and ASIC teams that require hardware prototypes of their project early in the design phase. It produces a functional ASIC prototype – partitioned among multiple FPGAs – from RTL code (either Verilog or VHDL). It uses built-in synthesis technology to optimize the final prototype performance. Unlike traditional ASIC prototyping techniques, it enables prototype delivery before ASIC synthesis, in order to allow a design team to recognize and correct problems earlier in the design flow.

- The Certify SC™ product is a single-chip version of the Certify tool that supports ASIC prototyping using embedded synthesis technology. It supports the relocation of logic blocks from the FPGA to black boxes, and includes enhanced debugging capabilities with extensive probe functions.

Product Audience

The Synplicity products are targeted to FPGA and ASIC system developers. Familiarity with the following is recommended for using the tools:

- Design synthesis

- Register transfer level (RTL) code describing the detailed behavior of a design

- FPGAs or CPLDs

- Window-based user interfaces
Overview of the Synplify Synthesis Tool

This section provides an overview of the Synplify synthesis tool. The following are covered:

- B.E.S.T. Technology
- Synthesis Tool Features
- Supported Platforms
- User Interface
- Projects, Implementations and Workspaces

B.E.S.T. Technology

Behavior Extraction Synthesis Technology (B.E.S.T.) is the underlying proprietary technology that the Synplify synthesis tool uses to extract and implement your design structures.

During synthesis, the B.E.S.T. feature recognizes high-level abstract structures like RAMs, ROMs, FSMs and arithmetic operators, and it maintains them, instead of converting the design entirely to the gate level. It automatically maps these high-level structures to technology-specific resources using module generators. For example, it maps RAMs to target-specific RAMs, and adders to carry chains. The B.E.S.T. algorithms also optimize hierarchy automatically.

Synthesis Tool Features

The Synplify synthesis tool has the following built-in features:

- The HDL Analyst® RTL analysis and debugging environment, a graphical tool for analysis and crossprobing. See Chapter 5, Using HDL Analyst.

- The Text Editor window, with a language-sensitive editor for writing and editing HDL code. See Text Editor View on page 2-7.

- The SCOPE™ (Synthesis Constraint Optimization Environment™) tool, which provides a spreadsheet-like interface for managing timing constraints and design attributes. See SCOPE Window on page 6-3.
Chapter 1: Product Overview

Overview of the Synplify Synthesis Tool

- FSM Compiler, a symbolic compiler that performs advanced finite state machine (FSM) optimizations. See FSM Compiler on page 1-21.
- Other special windows, or views, for analyzing your design.

Supported Platforms

For information on supported platforms, consult the appropriate Release Notes document for the product.

User Interface

With the exception of the Synplify tool, the Synplicity family of products share a common graphical user interface (GUI), in order to ensure a cohesive look and feel across the different products. In the Synplify Pro
tool, you can choose whether to use this common GUI or to run the tool in Classic mode, which uses the basic Synplify tool GUI (plus the Tcl window and Log Watch window of the common GUI).

Figure 1-1: Synplify Tool GUI
The following table shows where you can find information about different parts of the GUI, some of which are not shown in the above figure. For more information, see the Synplify User Guide.

Table 1-3: User interface – references

<table>
<thead>
<tr>
<th>For information about...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project window</td>
<td>Project Window on page 2-2</td>
</tr>
<tr>
<td>RTL view</td>
<td>RTL View on page 2-4</td>
</tr>
<tr>
<td>Technology view</td>
<td>Technology View on page 2-5</td>
</tr>
<tr>
<td>Text Editor view</td>
<td>Text Editor View on page 2-7</td>
</tr>
<tr>
<td>SCOPE spreadsheet</td>
<td>SCOPE Window on page 6-3</td>
</tr>
<tr>
<td>Other views and windows</td>
<td>Windows and Views on page 2-2</td>
</tr>
<tr>
<td>Menubar menu commands and their dialog boxes</td>
<td>Menubar Menus on page 3-4</td>
</tr>
<tr>
<td>Toolbars</td>
<td>Toolbars on page 3-88</td>
</tr>
<tr>
<td>Action buttons</td>
<td>Action Buttons and Options on page 3-97</td>
</tr>
<tr>
<td>Context-sensitive popup menus and their dialog boxes</td>
<td>Popup Menus on page 3-73</td>
</tr>
<tr>
<td>Online help</td>
<td>Click F1 or Help in a dialog box.</td>
</tr>
</tbody>
</table>
Projects, Implementations and Workspaces

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

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

A workspace allows you to group related projects together. Although a workspace can contain a set of projects, only one implementation is active at a time. All commands operate on the active project and its implementation. You can open a project independently of the workspace it belongs to, if needed. In the Project view, a workspace is shown as a folder at one level above the project folder.
Chapter 1: Product Overview

Starting the Synthesis Tool

You can start the Synplify synthesis tool in graphic mode or batch mode. This section discusses the following:

- Starting the Synthesis Tool in Interactive Mode
- Syntax for the synplify Command

Starting the Synthesis Tool in Interactive Mode

Before you can start the Synplify synthesis tool, you must install it and set up the software license appropriately. How you start the tool depends on your environment. For details, see the installation instructions.

This section shows you ways to start interactive use of the Synplify synthesis tool. For information about using the tool in batch mode, see Batch Mode on page 4-6.

- To start the synthesis tool from the Microsoft® Windows® operating system, choose
  
  Start -> Programs -> Synplicity -> Synplify <version>

- To start the tool from a DOS command line, specify the executable:
  
  synplify_installation_dir\bin\synplify.exe

  The executable name is the name of the product followed by an “exe” file extension (.exe).

- To start the synthesis tool from a UNIX platform, type this command at the UNIX prompt: synplify
Syntax for the synplify Command

This command lets you run synthesis in batch mode. It starts the Synplify synthesis tool and opens the Project window. It opens the project file or Tcl file, if you specified it.

Syntax

```
synplify -batch [<projectFile>]
```

The following table describes the command options.

Table 1-4: synplify command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-batch</td>
<td>Starts the Synplify synthesis tool in batch mode, without opening the Project window. It opens any project file that you specified.</td>
</tr>
<tr>
<td>-log</td>
<td>Writes all output to the specified log file</td>
</tr>
</tbody>
</table>
Using the Synplify Synthesis Tool

This section gives you an overview of the synthesis software flow and describes how to run the synthesis tool and optimize synthesis results. It discusses the following:

- Synthesis Software Flow
- Synthesizing Your Design
- Optimizing for Best Performance
- Optimizing for Area
- Setting Fanout Limits

Synthesis Software Flow

When you run the Synplify synthesis tool, it performs logic synthesis. This consists of two stages: 1) logic compilation (HDL language synthesis) and optimization, and 2) technology mapping.

Logic compilation and optimization: The synthesis tool first compiles input HDL source code, which describes the design at a high level of abstraction, to known structural elements. Next, it optimizes the design, making it as small as possible and improving circuit performance. These optimizations are technology independent.
Technology mapping: During this stage, the tool optimizes the logic for the target technology, by mapping it to technology-specific components. It uses architecture-specific techniques to perform additional optimizations. Finally, it generates a design netlist for place-and-route.

Synthesizing Your Design

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

You perform the following actions to synthesize your design using the GUI. For details, see the Synplify User Guide, Chapter 7, Synplify Tutorial.

1. Access your design’s project: open an existing project or create a new one.

2. Specify the input source files to use. Right-click the project name in the Project view, then choose Add Source Files.

   Select the desired Verilog or VHDL file(s), then click OK. (See the examples in the directory installation_dir/examples, where installation_dir is the directory where the product is installed.)
You can also add source files in the Project view by dragging and dropping them there from a Windows® Explorer folder (Microsoft® Windows® only).

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

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

See Specifying Timing Constraints on page 6-2, for details on the SCOPE spreadsheet.

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

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

5. Synthesize the design by clicking the Run button.

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

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

6. After synthesis, do one of the following:
   - If there were no synthesis warnings or error messages (Done!), analyze your results in the RTL and Technology views. You can then resynthesize with different implementation options, or use the synthesis results to simulate or place-and-route your design.
If there were synthesis warnings (Done (warnings)) or error messages (Errors!), then check them in the log file. You can double-click an error or warning message in any of these places, to jump to the corresponding source code: the log file, the HDL file in the Project view, or the Tcl window. Use Next Error and Previous Error from the Run menu to navigate.

Correct any errors, then rerun synthesis.

Optimizing for Best Performance

When you optimize for improved timing and fast results, there are two issues to consider:

- **The timing/area trade-off.** When you optimize for timing, area used usually increases. See Optimizing for Area on page 1-19 and Setting Area/Delay Trade-offs on page 1-19 for additional information.

- **The correlation between synthesis timing and post-routing timing.** During timing analysis, the Synplify synthesis tool uses statistical modeling to determine routing delays. A given net delay in a routed design can actually be larger or shorter than the synthesis estimate.

The following lists some things you can try to improve performance.

- Set a global frequency goal in the Project view (Impl Options button -> Options/Constraints panel, Frequency (MHz) field).

- Use timing constraints in addition to setting the clock frequency. For details, see Chapter 6, Timing Constraints, or the appropriate vendor appendix.

  - If you have multiple clocks, override the default clock frequency with the define_clock timing constraint. Use the SCOPE spreadsheet to place timing constraints and attributes in a constraint file, and then add the file to the project’s source files list.

  - Use define_clock to specify clock groups.

  - If you know the delays outside the chip for inputs and outputs, set them with the define_input_delay and define_output_delay timing constraints.

  - Use define_reg_input_delay and define_reg_output_delay to define register delays.
– Use `define_multicycle_path` and `define_false_path` to identify multicycle paths and false paths, respectively.

– Use the following for black-box combinational delay, setup delay for input pins, and output delay through the black box, respectively: `syn_tpd<n>`, `syn_tsu<n>`, and `syn_tco<n>`.

• Improve the accuracy of the results by using the `-improve` and `-route` option with the timing constraints. The `-improve` option restructures the design to speed up the path and meet the frequency goal. The `-route` option accounts for the routing delay difference by adding or subtracting a value.

• Set the following implementation options using `Project -> Implementation Options`:
  – Choose a faster speed grade.
  – Decrease the fanout limit. See *Setting Fanout Limits on page 1-20* for information on setting a fanout limit. For very large fanout nets, consider instantiating a global buffer if it exists in the technology you are targeting. In addition to reducing delay for a large fanout net, it can free up routing resources for other signals.
  – Use vendor-specific options available for the technology you are using. You can also set vendor-specific or technology-specific attributes using the SCOPE spreadsheet.

• If the design contains state machines, consider enabling the FSM Compiler option to possibly improve timing.

• Analyze the design using HDL Analyst (RTL or Technology view) and other specialized views, looking for opportunities to improve the results. For example, HDL Analyst might display a very long path, and you might be able to modify the source code to avoid the long delay.

Double-check that synthesis constraints are set properly, in particular, false paths and multicycle paths, then synthesize the design and check the result. If you are within 5 to 10 percent of the desired frequency, placement and routing of the design will likely meet the goal. You may also need to set certain options in the place-and-route tools.
Optimizing for Area

Optimizing for area usually decreases design timing performance. See Setting Area/Delay Trade-offs, below, for additional information about this trade-off.

However, if you find that you do need to decrease area, you can try changing constraints, optimization switches such as resource sharing, or source code, then rerun synthesis. Analyze your results using the HDL Analyst RTL and Technology views and/or crossprobing.

Setting Area/Delay Trade-offs

Not all designs respond significantly to this area/delay trade-off control. You do not need this control for synthesizing FPGAs, but you can use it for CPLDs.

Area/Delay Trade-off for CPLDs

For CPLDs, you can control an area/delay trade-off by setting the percentage of the design you want optimized for timing.

1. Choose Project -> Implementation Options -> Device, and enable Percent of design to optimize for timing.

2. Enter a percentage (integer value) of the paths to which you want to apply timing optimizations.

   Set a percent value that is just high enough to reach the maximum optimization. Large values (greater than 50 percent) are usually counterproductive because the program can not make trade-offs. Start with the control set to low (for example, 5 percent), and gradually work up, 5 or 10 percent at a time, until you obtain the desired result.

3. Synthesize your design.

   The paths are first ordered through the design from longest to shortest. Then timing optimizations are first applied to the longest paths, and then other paths are optimized until the percentage of paths you specified has been optimized.
Area/Delay for FPGAs

An area/delay trade-off control is not necessary (or supported) for synthesis to FPGAs. Instead, set a clock frequency in the Project view so that the Synplify synthesis tool can target your timing goal.

Setting Fanout Limits

Large fanouts can increase delays and cause routability problems. You can set a fanout limit for the Synplify synthesis tool to use during technology mapping. The tool tries to keep the fanout under the limit, which is a guideline rather than a hard limit.

1. Choose Project -> Implementation Options, and set Fanout limit to an integer.

   The value represents the fanout guideline for a given driver in the design. If you do not set the fanout limit, it defaults to a number appropriate for the target architecture.

2. Synthesize the design.

   The synthesis tool first reduces fanout by replicating the driver of the high fanout net and splitting the net into segments. Replication can affect the number of register bits in the design. If replication is not possible, the synthesis tool buffers the signal. Buffering is more expensive in terms of intrinsic delay and consumption of resources, and is not used until a higher fanout limit is specified.

   Net timing affects the fanout limit for a net. Critical nets are replicated or buffered more aggressively.

3. Review the log file.

   The log file contains a net buffering report that shows how many nets were buffered or had their source replicated, and the number of segments created for the net. Click the View Log action button in the Project view to display the net buffering report.
FSM Compiler

What FSM Compiler Does

Other synthesis tools treat state machines as regular logic, but the Synplicity’s FSM Compiler performs advanced, proprietary state machine optimization techniques. You simply enable the FSM compiler; you do not need special directives or attributes to locate the state machines in the design. You can also, however, turn on the FSM compiler for selected individual state machines, using synthesis directives in the HDL description.

When you enable the FSM compiler, state machines are discovered and extracted into symbolic graph form, then special optimizations are performed. These include re-encoding state representations, which generates a better starting point for logic optimization.

The FSM compiler examines your design for state machines. It looks for registers with feedback that is controlled by the current value of the register, such as case or if-then-else statements that test the current value of a state register. It converts state machines to a symbolic form that provides a better starting point for logic optimization. Several proprietary optimizations are performed on each symbolic state machine.

Converting from an encoded state machine to a one-hot state machine often produces better results. However, one-hot implementations are not always the best choice, even in FPGAs and CPLDs. For example, one-hot state machines might result in higher speeds in CPLDs, but cause fitting problems because of the larger number of global signals. An example in which the one-hot implementation can be detrimental in an FPGA is when the state machine drives a large decoder, generating many output signals. In a 16-state state machine, for instance, the output decoder logic might reference eight signals in a one-hot implementation, but only four signals in an encoded representation.

During synthesis, a state encoding for an FSM is determined, based on certain predefined characteristics of the state machine. The optional FSM Explorer feature enhances this capability by automatically determining and using the best encoding styles for the state machines based on the design constraints and the area/delay requirements. You can force the use of a particular encoding style for a state machine by including the appropriate directive in the HDL description.
The log file contains a description of each state machine extracted, including a list of the reachable states and the state encoding used.

**When to Use FSM Compiler**

Use the symbolic FSM compiler to generate better results for the state machines, or to debug them. If you do not want to use the symbolic FSM compiler on the final circuit, you can decide to use it during just initial synthesis, in order to check that the state machines are described correctly. Many common state machine description errors result in unreachable states, which are then optimized away during synthesis, resulting in a smaller number of states than you may expect. Reachable states are reported in the log file.

To view a textual description of an FSM in terms of inputs, states, and transitions, select the state machine in the RTL view. Right-click, then choose View FSM Info File in the popup menu. You can view the same information graphically with the FSM viewer. The graphical description of a state machine makes it easier to verify behavior.

**Where to Use FSM Compiler (Global vs. Local Use)**

Enable the FSM Compiler check box in the Project view to turn on FSM synthesis. This allows the tool to recognize, extract and optimize the state machines in the design.

The following table summarizes the operations you can carry out. For more information, see the Synplify User Guide, Chapter 5, Design Optimization, section Using the Symbolic FSM Compiler.

<table>
<thead>
<tr>
<th>To...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globally enable (disable) the FSM Compiler</td>
<td>Enable (disable) the FSM Compiler check box in the Project view.</td>
</tr>
<tr>
<td>Enable (disable) the FSM compiler for a specific register</td>
<td>Enable (disable) the FSM Compiler check box, and set the Verilog syn_state_machine directive to 1 (0), or the VHDL syn_state_machine directive to true (false), for that instance of the state register.</td>
</tr>
</tbody>
</table>
Files

This section contains an overview of the files used by the Synplify synthesis tool.

- Project Files
- HDL Source Files
- Constraint Files
- Output Files
- Log File

Project Files

A project file (.prj) is a repository for all of the information required to complete a design. It is in Tcl format, and contains references to source files, compilation, mapping and optimization switches, specifications for target technology and other runtime options.

HDL Source Files

HDL source files can be in either VHDL or Verilog format, but they must all be one or the other.

VHDL

The Synplify synthesis tool supports a synthesizable subset of VHDL93 (IEEE 1076), and the following IEEE library packages:

- numeric_bit
- numeric_std
- std_logic_1164
The synthesis tool also supports the following industry standards in the IEEE libraries:

- std_logic_arith
- std_logic_signed
- std_logic_unsigned

The Synplify synthesis tool contains built-in macro libraries for vendor macros like gates, counters, flip-flops, and I/Os. If you use the built-in macro libraries, you can easily instantiate vendor macros directly into the VHDL designs, and forward-annotate them to the output netlist. Refer to the appropriate vendor support chapter for more information.

The Synplify synthesis tool library contains an attributes package (installation_dir/lib/vhd/synattr.vhd) of built-in attributes and timing constraints that you can use with VHDL designs. The package includes declarations for timing constraints (including black-box timing constraints), vendor-specific attributes and synthesis attributes. To access these built-in attributes, add the following two lines to the beginning of each of the VHDL design units that uses them:

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

For more information about the VHDL language and the synthesis commands and attributes you can include, see Chapter 9, VHDL Language Support. The following is a list of recommended VHDL reading:


Verilog

The Synplify synthesis tool supports a synthesizable subset of Verilog 2001 and Verilog95 (IEEE 1364). For more information about the Verilog language and the synthesis commands and attributes you can include, see Chapter 8, *Verilog Language Support*.

The Synplify synthesis tool contains built-in macro libraries for vendor macros like gates, counters, flip-flops, and I/Os. If you use the built-in macro libraries, you can easily instantiate vendor macros directly into the Verilog designs, and forward-annotate them to the output netlist. Refer to the appropriate vendor support chapter for more information.

Here is a list of recommended Verilog reading:


Constraint Files

Timing constraints help improve synthesis results. You can specify constraints in any of these ways:

- Attributes added to the source files. See Chapter 7, *Synthesis Attributes and Directives*.
- Constraints added interactively to a constraints file (.sdc), using the SCOPE spreadsheet. See *SCOPE Window on page 6-3* for more information.

Constraint files are optional. A constraint file contains information on constraints that you can set for design objects, such as clock parameters, I/O delays and multicycle paths. See *SCOPE Window on page 6-3* for more information.
Chapter 1: Product Overview

Files

- STAMP model files for black-box timing constraints. See the Synplify User Guide for more information.

For more information about constraints, see Chapter 6, Timing Constraints.

Output Files

In addition to reports about the synthesis is run, the Synplify synthesis tool generates files that you can use for simulation or placement and routing. For more information about the log file and the report files, see Reports and Messages on page 1-27.

You use the Implementation Results panel of the Options for Implementation dialog box to choose the kinds of output files you want to generate.

- Vendor-specific synthesized netlist files

  The synthesized netlist is written out in a format appropriate to the technology you are using, and to the place-and-route tool being used. Generally, this format is EDIF. This file is also referred to as the “results” file because it contains the synthesis results.

- Constraints file for forward annotation

  Depending on the vendor, you can forward-annotate some constraints to the place-and-route tool. Refer to the vendor chapters for specific information about the constraints you can forward-annotate.

- Mapped Verilog/VHDL netlist files

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

  Typically, you use this netlist for gate-level simulation, to verify your synthesis results. Some designers prefer to simulate before and after synthesis, and also after place-and-route. This approach helps them to isolate the stage of the design process where a problem occurred. Most place-and-route tools work best with structural Verilog netlists, so a structural VHDL netlist is generally only needed for simulation.
The Verilog and VHDL output files are for functional simulation only. When you input stimulus into a simulator for functional simulation, use a cycle time for the stimulus of 1000 time ticks.

- `.srr` – A synthesis log file that provides information on the synthesis run, as well as area and timing reports.
- `.srm` – A file output by the mapper stage of the process. It contains the actual technology-specific mapped design. This is the representation that is displayed graphically through the Technology view.
- `.srs` – A file output by the compiler stage of the synthesis process. It contains the RTL-level (schematic) view of the design. This is the representation that is displayed graphically through the RTL view.

### Reports and Messages

This section describes messages, such as errors and warnings, and report files that are generated by the Synplify synthesis tool.

- **Log File**
- **Timing Report**
- **Net Buffering Report**
- **Resource Usage Report**
- **Informational Files**
- **Errors, Warnings, Notes and Information**

### Log File

The Synplify synthesis tool writes messages and all reports on synthesis, timing and usage to the log file, which is located in the implementation directory (folder). This file, `project_name.srr`, is (re-)written each time you compile or synthesize (compile and map) the project. When you compile a project without mapping it the log file contains only compilation information, such as syntax errors and warnings.
(You can view the log file via the View Log action button in the Project view. See *Action Buttons and Options* on page 3-97.)

**Log File Contents**

The log file includes information like the following:

- List of compiled files compiled
- Syntax or synthesis warnings, errors, and notes
- List of user options set for synthesis
- State machine extraction information, including a list of reachable states, if the symbolic FSM compiler is turned on during synthesis
- Reports appropriate to the technology, including the following:
  - Timing Report
  - Net Buffering Report
  - Resource Usage Report

**Log File Color Coding**

The log file uses different colors to identify different types of message:

<table>
<thead>
<tr>
<th>Color</th>
<th>Message Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>Information (@I)</td>
<td>@I::&quot;D:\alu.v&quot;</td>
</tr>
<tr>
<td></td>
<td>Notes (@N)</td>
<td>@N:&quot;D:\p.v&quot;:58:0:58:5</td>
</tr>
<tr>
<td>Purple</td>
<td>Warnings (@W)</td>
<td>@W:&quot;D:\e.v&quot;:53:9:53:12</td>
</tr>
<tr>
<td>Red</td>
<td>Errors(@E)</td>
<td>@E:&quot;D:\e.v&quot;:53:9:53:12</td>
</tr>
</tbody>
</table>

**Timing Report**

A timing report on the critical path is written to the log file *(project_name.srr)* in the “START TIMING REPORT” section. See *Timing Report* on page 6-46 for detailed information about the timing report.
Net Buffering Report

Net buffering reports are generated for most of the supported FPGAs and CPLDs. This information is written in the log file (project_name.srr).

The net buffering report provides the following information:

- The nets that were buffered, or had their source replicated
- The number of segments created for that net
- The total number of buffers added during buffering
- The number of registers and look-up tables (or other cells) added during replication

Example: Net Buffering Report

Net buffering Report:
Badd_c[2] - loads: 24, segments 2, buffering source
Badd_c[1] - loads: 32, segments 2, buffering source
Badd_c[0] - loads: 48, segments 3, buffering source
Aadd_c[0] - loads: 32, segments 3, buffering source
Added 10 Buffers
Added 0 Registers via replication
Added 0 LUTs via replication

Resource Usage Report

A resource usage report is written in the log file (project_name.srr) every time you compile or synthesize (compile and map). The format of the resource usage report varies, depending on the architecture you are using.

The report provides the following information:

- The total number of cells, and the number of combinational and sequential cells in the design
- The number of clock buffers and I/O cells
- Details of how many of each type of cell in the design
Informational Files

Depending on your design, additional report files may be generated. These are listed in the Project view, after synthesis.

- 
- 
- .info contains detailed information about design components like state machines or ROMs.
- .fse contains information about state machine encodings.

Errors, Warnings, Notes and Information

Errors, warnings and notes detected during synthesis are reported in the Project view, the log file and the Tcl window.

- Error messages begin with “@E”.
- Warning messages begin with “@W”.
- Notes begin with “@N”.

CHAPTER 2

User Interface Overview

This chapter presents tools and technologies that are built into the Synplify synthesis software to enhance your productivity. This presentation does not treat the HDL Analyst or the SCOPE spreadsheet features, which are described in Chapter 5, Using HDL Analyst, and Chapter 6, Timing Constraints, respectively. Individual commands, menus and dialog boxes are detailed in Chapter 3, Menus, Dialog Boxes . . .

- Windows and Views on page 2-2
- Using the Interface Effectively on page 2-10
Windows and Views

The Synplify synthesis tool’s GUI provides windows and views that help you manage input and output files, direct the synthesis process, and analyze your design and its results. This section presents these windows and views, except for the SCOPE tool window, which is described in SCOPE Spreadsheet on page 6-3.

- Project Window on page 2-2
- RTL View on page 2-4
- Technology View on page 2-5
- Text Editor View on page 2-7

Project Window

The Project view (window) lets you create or open projects, create new implementations, set device options and synthesize designs. This is the main working window. It contains a list of files, and also has action buttons for quick access to common commands. For information about setting display options in the Project view, see Setting Project View Display Preferences on page 2-12.

The Project window has three main parts:

- **Action buttons** and user options give you immediate access to some of the more common commands. (See Action Buttons and Options on page 3-97.)

- The **Project Tree** view lists the workspaces and projects, and their associated HDL source files and constraint files (see Projects, Implementations and Workspaces on page 1-11).

- The **Implementation Results** view lists the result files for the current (active) implementation.

  You can only view one set of implementation results at a time. You click an implementation in the Project Tree view to make it active and view its result files.

  The Implementation Results view lists the names and types of the result files, and the dates they were last modified. You can click a
column heading to sort the file list accordingly. Click again to reverse the sort direction. An arrow in the column header displays the sorting direction.

Figure 2-1: Project view
Chapter 2: User Interface Overview

Windows and Views

RTL View

The RTL view includes the RTL Schematic view and the corresponding Hierarchy Browser. The RTL view is only available after a design has been successfully compiled. To display it, do one of the following:

- Choose a hierarchical or flattened view from the HDL Analyst -> RTL menu.
- Click the RTL View toolbar icon ( ).

The RTL view shows your design as a high-level, technology-independent schematic. The RTL view has two panes (you can drag the pane divider left and right with the mouse): the Hierarchy Browser on the left, and the RTL Schematic view on the right. The bar at the top displays the name of the view, the current level (sheet), and the total number of sheets in the schematic. The schematic offers a graphic abstraction of your design. The design is represented using technology-independent components like variable width adders, registers, large muxes, and state machines.

![RTL View Image]

Figure 2-2: RTL view

The RTL view offers the following analysis and debugging features:

- Hierarchy browser – A hierarchical breakdown providing easy access to the inputs, outputs, nets, gates, and registers, and their quick identification via symbols and names.
• Push/pop hierarchy – Incremental traversal of the design hierarchy, pushing down level by level, all the way to the primitive gate level.

• Sheet views – Quick access to a specific sheet in a multisheet project.

• Net drivers – Easy identification of what drives selected nets.

• Driven instance selection – Quick selection of logic instances that are driven by a given net.

• Cross probing of your design between different views (text, schematic).

• Schematic filtering – Lets you isolate logic between inputs, outputs and registers, to help you refine your design.

• Searching for instances, symbols, nets, ports and properties in your schematic, by name.

See the following for additional information:

• Customizing the RTL and Technology Views on page 2-14

• Managing Views on page 2-13

• Chapter 5, Using HDL Analyst

**Technology View**

The Technology view is only available after a design has been synthesized (compiled and mapped). To display the Technology view, do one of the following:

• Choose a view from the HDL Analyst -> Technology menu.

• Click the Technology View toolbar icon ( ).

The Technology view displays the design as a low-level, technology-specific schematic. Like the RTL view, it consists of the Hierarchy Browser on the left and a schematic view on the right. The bar at the top displays the name of the view, the current level, and the total number of sheets in the schematic. The Technology Schematic view shows technology-specific components, such as look-up tables, cascade and carry chains, F and H Maps, muxes, and flip-flops.
Figure 2-3: Technology view

The Technology view offers the following features:

- Critical path selection – Selection and highlighting of instances and nets in the critical path of your design, with display of associated individual timings.
- Schematic filtering – isolation of selected logic elements, using various criteria.
- Searching for instances, symbols, nets, ports, and properties in your schematic, by name.
- Design views – Various views in which to analyze your design: hierarchical, partially or totally flattened (to the gate level), and critical path (hierarchical or flattened).

See the following for additional information:

- *Customizing the RTL and Technology Views* on page 2-14
- *Managing Views* on page 2-13
- Chapter 5, *Using HDL Analyst*
Text Editor View

The Text Editor view displays text files. These can be constraint files, source code files, log files, or other informational files and reports. You can enter and edit text in the window. You use this window to update source code and fix syntax or synthesis errors. You can also use it to crossprobe the design.

Figure 2-4: Text Editor view

Starting the Text Editor

To open the Text Editor to edit an existing file, do one of the following:

- Double-click a source code file (.v or .vhdl) in the Project view.
- Choose File -> Open. In the dialog box displayed, double-click a file to open it.
With the Microsoft® Windows® operating system, you can instead
drag and drop a source file from a Windows folder into the gray
background area of the GUI (not into any particular view).

To open the Text Editor on a new file, do one of the following:

- Choose File -> New, then specify the kind of text file you want to create.
- Click the HDL toolbar icon ( ).

The Text Editor colors HDL source code keywords such as module and
output blue, and comments green. For information about display options in
this window, refer to Setting Text Editor Window Preferences on page 2-14.

Text Editor Features

The Text Editor has the following features:

Table 2-1: Text Editor features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color coding</td>
<td>Keywords are blue, comments green, and strings red. All other text is black.</td>
</tr>
<tr>
<td>Editing text</td>
<td>You can use the Edit menu or keyboard shortcuts for basic editing operations like Cut, Copy, Paste, Find, Replace, and Goto.</td>
</tr>
<tr>
<td>Completing keywords</td>
<td>A keyword is automatically completed if you type enough characters for it to be unique, then press the Esc key.</td>
</tr>
<tr>
<td>Indenting a block of text</td>
<td>The Tab key indents a selected block of text to the right. Shift-Tab indents text to the left.</td>
</tr>
<tr>
<td>Inserting a bookmark</td>
<td>Click the line you want to bookmark. Choose Edit -&gt; Toggle Bookmark, type Ctrl-F2, or click the Toggle Bookmark icon ( ) on the Edit toolbar. The line number is highlighted to indicate that there is a bookmark at the beginning of the line.</td>
</tr>
<tr>
<td>Deleting a bookmark</td>
<td>Click the line with the bookmark. Choose Edit -&gt; Toggle Bookmark, type Ctrl-F2, or click the Toggle Bookmark icon ( ) on the Edit toolbar. The line number is no longer highlighted, after the bookmark is deleted.</td>
</tr>
</tbody>
</table>
### Table 2-1: Text Editor features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deleting all bookmarks</td>
<td>Choose <strong>Edit -&gt; Delete all Bookmarks</strong>, type Ctrl-Shift-F2, or click the <strong>Clear All Bookmarks</strong> icon (🪚) on the <strong>Edit</strong> toolbar. The line numbers are no longer highlighted, after the bookmarks are deleted.</td>
</tr>
<tr>
<td>Editing columns</td>
<td>Press and hold <strong>Alt</strong>, then drag the mouse down a column of text, to select it.</td>
</tr>
<tr>
<td>Commenting out code</td>
<td>Choose <strong>Edit -&gt; Advanced -&gt; Comment Code</strong>. The rest of the current line is commented out: the appropriate comment prefix is inserted at the current text cursor position.</td>
</tr>
<tr>
<td>Checking syntax</td>
<td>Use <strong>Run -&gt; Syntax Check</strong> to highlight syntax errors, such as incorrect keywords and punctuation, in source code. If the active window shows an HDL file, then only that file is checked. Otherwise, the entire project is checked.</td>
</tr>
<tr>
<td>Checking synthesis</td>
<td>Use <strong>Run -&gt; Synthesis Check</strong> to highlight hardware-related errors in source code, like incorrectly coded flip-flops. If the active window shows an HDL file, then only that file is checked. Otherwise, the entire project is checked.</td>
</tr>
</tbody>
</table>
Using the Interface Effectively

This section describes some techniques and procedures for using the user interface more effectively. It covers the following topics:

- Using the Mouse on page 2-10
- Setting Project View Display Preferences on page 2-12
- Managing Views on page 2-13
- Setting Text Editor Window Preferences on page 2-14
- Customizing the RTL and Technology Views on page 2-14

Using the Mouse

The mouse button operations in Synplicity’s products are fairly standard, so it is likely that you are already familiar with them; Using The Mouse Buttons on page 2-11 contains a summary, in case you are not.

Synplify also has support for a mouse wheel; Using The Mouse Wheel (MS Windows Only) on page 2-11 describes this.

Note: Terminology —
“Click” in the Synplicity documentation means click with the left mouse button: press and release it without moving the mouse.
“Double-click” means click twice rapidly, without moving the mouse.
“Right-click” means click with the right mouse button.
“Drag” means press the left mouse button, hold it down while moving the mouse, then release it. Dragging initiated over an object often moves it to where the mouse is released; then, releasing is sometimes called “dropping”. Dragging initiated when the mouse is not over an object often traces a selection rectangle, or box (also called a “lasso”), whose diagonal corners are at the press and release positions.
“Press” and “hold” are sometimes used as abbreviations for “press and hold”.
Using The Mouse Buttons

- You can select an (unselected) object by clicking it. You can unselect a (selected) object by clicking it.

- You can select and unselect multiple objects by pressing and holding the Control key (Ctrl) while clicking each of them.

- You can select all of the objects in a region by tracing a selection rectangle around them (lassoing).

- You can select text by dragging the mouse over it. You can alternatively select a single “word” (text surrounded by whitespace such as spaces) by double-clicking it.

- You can access a contextual popup menu by right-clicking or pressing and holding the right mouse button. The menu displayed is specific to the current context, including the object or window under the mouse. For example, right-clicking a project name in the Project view displays a popup menu with operations appropriate to the project file. Right-clicking a source (HDL) file in the Project view displays a popup menu with operations applicable to source files.

- Double-clicking often both selects an object and immediately initiates a default action with it. For example, double-clicking a source file in the Project view opens the file in a Text Editor window.

Using The Mouse Wheel (MS Windows Only)

If your mouse has a wheel, you can use it to scroll and zoom, as follows:

- Whenever a horizontal scroll bar is visible, rotating the wheel scrolls the window horizontally.

- Whenever a vertical scroll bar is visible, rotating the wheel while pressing and holding the Shift key scrolls the window vertically.

- In a window that can be zoomed, such as a graphics window, rotating the wheel while pressing and holding the Ctrl key zooms the window.
Setting Project View Display Preferences

You can customize the organization and display of project files, as follows.

1. Choose Options -> Project View Options.

The Project View Options dialog box opens (see Project View Options Dialog Box on page 3-84).

2. To organize different kinds of files in folders, enable View Project Files in Folders in the dialog box.

This creates separate folders in the Project view for constraint files and source files.

View Project Files in Folders enabled

View Project Files in Folders disabled
3. To automatically display all the files, enable Show Project Library. If this is disabled, the Project view does not display the names of files inside a folder until you click its plus symbol ( + ) to expand the folder.

4. To determine how filenames are displayed, choose (click) one of the radio buttons in the Project File Name Display zone of the dialog box. You can choose to display the filename only, the relative path, or the absolute path.

Managing Views

As you work on a project, you move between different views of the design. The following guidelines can help you manage the different views you have open.

1. Enable View ->Workbook Mode.

   Below the Project view, you then see tabs like the following, one for each open view. The icons accompanying the view name on the tab indicate the kind of view.

   ![Project 1 and Project 2]

2. To bring an open view to the front and make it current (active), click any visible part of the window, or click the window’s tab.

   If you previously minimized the view, it will be activated but will remain minimized. To raise (display) it, you double-click a minimized window.

3. To bring the next window to the front, type Ctrl-F6. Repeating this cycles through all the open windows.

4. To close a view, type Ctrl-F4 in that window, or choose File -> Close.
5. You can rearrange open windows using the Window menu: you can cascade them (stack them, slightly offset) or tile them horizontally or vertically.

**Setting Text Editor Window Preferences**

You can use the Editor Options dialog box to customize the Text Editor so that it displays fonts and colors you choose (see *Editor Options Dialog Box* on page 3-56).

**Customizing the RTL and Technology Views**

You can customize the colors used in the RTL view and Technology view schematics, and specify the number of objects to display in a schematic sheet.

1. Edit the .ini file. On the Microsoft® Windows® operating system, this file is in the \WINDOWS\ (or \WINNT\) directory. On UNIX workstations, it is in the Windows subdirectory of your home directory (~/windows, where ~ is your home directory, which can be set via environment variable $HOME).

2. Adjust the RGB (red, green, blue) values in the “[Schematics]” section of the file.

   Here are some examples of RGB values:

   0 0 0 = black
   255 255 255 = white
   255 0 0 = red
   200 200 200 = gray
3. To change the amount of logic displayed on a single schematic sheet, use the \texttt{PartitionSize=value} statement.

   – To see your entire design on a single page, set the \textit{value} to a number larger than the total number of components in the design. A larger \textit{value} means displaying takes longer.

   – To see a smaller area of your design on each page, set the \textit{value} to a lower number.
CHAPTER 3

Menus, Dialog Boxes . . .

This chapter describes the different ways the graphical user interface (GUI) lets you access commands, the commands themselves, and their associated dialog boxes.

- Command Access on page 3-2
- Menubar Menus on page 3-4
- Popup Menus on page 3-73
- Toolbars on page 3-88
- Keyboard Shortcuts on page 3-93
- Action Buttons and Options on page 3-97
Command Access

Interactive commands can be accessed by menubar menus, context-sensitive popup menus, toolbar icons, or action buttons. Keyboard shortcuts are also available for commonly used commands, and Tcl command equivalents are defined for most of the interactive commands available in the Project view.

Menubar

The menubar consists of pulldown menus which vary with the active view. The set of commands available on a given menubar menu varies according to view, design status, task to be performed, and selected object. For example, the File menu commands in the Project view differ slightly from those in the RTL view File menu. Menu items that are not available for the current context are grayed out (dimmed).

![Figure 3-1: Project view menubar](image)

Context-sensitive Popup Menus

Popup menus, available by right-clicking, offer access to commonly used commands that are specific to the current context. For example, if you right-click a project name in the Project Tree view, you get a popup menu with project file commands. Many popup menu commands are also available from the menubar.

Toolbars

Toolbars contain active icons (iconic action buttons) associated with commonly used commands. See Toolbars on page 3-88 for details. You can also access the toolbar commands from the menubar.
Keyboard Shortcuts

Keyboard shortcuts are available for commonly used commands. The shortcut is displayed next to the command in the menu. See Keyboard Shortcuts on page 3-93 for details.

Action Buttons

The Project view has action buttons for quick access to commonly used commands and options. See Action Buttons and Options on page 3-97 for details.

Tcl Command Equivalents

Tcl commands can be included in Tcl scripts that you can run in batch mode. For information about Tcl commands, see Chapter 4, Tcl Commands and Scripts.
Menubar Menus

Most commands are accessible via menubar menus. Some of these menus vary, according to the current context. For example, the View menubar menu contains the command Show Line Numbers in the Text Editor view, but not in other views. Similarly, the View menu in the RTL and Technology views contains several commands that are not in the View menu of other views.

The individual menubar menus, their commands (menu items) and the associated dialog boxes, are described in the following sections:

- File Menu on page 3-4
- Edit Menu on page 3-16
- View Menu on page 3-22
- Project Menu on page 3-29
- Run Menu on page 3-42
- HDL Analyst Menu on page 3-45
- Format Menu on page 3-51
- Options Menu on page 3-53
- Window Menu on page 3-62
- Help Menu on page 3-64

File Menu

You use the File menubar menu for such tasks as opening, creating, saving and closing projects and files. The available File menu commands vary according to the current view (Project, Text Editor, RTL or Technology view) and the synthesis technology being used. The following figure shows the File menu in the Project view.
The File menu commands and their associated dialog boxes are described in the following sections:

- *File Menu Commands on page 3-5*
- *New Dialog Box on page 3-7*
- *Open Dialog Box on page 3-9*
- *Save As Dialog Box on page 3-10*
- *Select Files to Add to Project Dialog Box on page 3-11*
- *Open Project Dialog Box on page 3-12*
- *Select Projects to Include in Workspace Dialog Box on page 3-14*
- *Print Dialog Box on page 3-14*
- *Page Setup Dialog Box on page 3-15*
- *Print Setup Dialog Box on page 3-16*

### File Menu Commands

The following table describes the File menu commands, beginning with those commands that appear in every view, followed by those that appear only in certain views.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Menu Commands Available in All Views</strong></td>
<td></td>
</tr>
<tr>
<td>New...</td>
<td>Displays a dialog box for opening a new file for your project. Available file types include text, Tcl script, VHDL, Verilog, constraint, and project. See <em>New Dialog Box on page 3-7.</em></td>
</tr>
<tr>
<td>Open...</td>
<td>Displays a dialog box where you choose an existing file to open. See <em>Open Dialog Box on page 3-9.</em></td>
</tr>
<tr>
<td>Close</td>
<td>Closes the active view and any associated files.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the contents of the current view to its associated file. The first time you save a file, you use the <em>Save As</em> dialog box to specify the name, type and directory of the file to be saved.</td>
</tr>
</tbody>
</table>
Table 3-1: File menu commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save As...</td>
<td>Displays the Save As dialog box, where you specify the name, type and directory of the file to be saved. See <em>Save As Dialog Box on page 3-10</em>.</td>
</tr>
<tr>
<td>Save All</td>
<td>Saves all open files.</td>
</tr>
<tr>
<td>Build Project...</td>
<td>Creates a project based on the file open in the Text Editor (if active), or lets you choose files to add to a new project. See <em>Select Files to Add to Project Dialog Box on page 3-11</em>.</td>
</tr>
<tr>
<td>Open Project...</td>
<td>Opens a project. See <em>Open Project Dialog Box on page 3-12</em>.</td>
</tr>
<tr>
<td>Close Project</td>
<td>Closes the current project.</td>
</tr>
<tr>
<td>Recent Projects</td>
<td>Lists recently accessed projects. Choose a project listed in the submenu to open it.</td>
</tr>
<tr>
<td>Recent files</td>
<td>The last six files you opened, as separate menu items. Choose a file to open it.</td>
</tr>
<tr>
<td>Exit</td>
<td>Exits the session.</td>
</tr>
</tbody>
</table>

**File Menu Commands Available in Specific Views**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print...</td>
<td>Available in all views except the Project view. Displays a standard Print dialog box. See <em>Print Dialog Box on page 3-14</em>. For more information about printing, see the appropriate License Configuration and Set Up document for your platform.</td>
</tr>
<tr>
<td>Print Preview</td>
<td>Available in all views except the Project view. Previews print results. In the Text Editor, if any text is selected (highlighted), then only the lines of the selected text are previewed for printing.</td>
</tr>
<tr>
<td>Page Setup...</td>
<td>Available only in the Text Editor view, but settings apply to all views that support printing. Sets header and footer information. See <em>Page Setup Dialog Box on page 3-15</em>.</td>
</tr>
<tr>
<td>Print Setup...</td>
<td>Available in all views except the Project view. Displays the standard Print Setup dialog box, where you specify printer parameters. See <em>Print Setup Dialog Box on page 3-16</em>.</td>
</tr>
</tbody>
</table>
Table 3-1: File menu commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print Image</td>
<td>Available only in the HDL Analyst RTL and Technology views. Prints an image of a region you define in the view. You define the region to be printed by dragging the camera pointer, then release the mouse button to display the standard Print dialog box.</td>
</tr>
<tr>
<td>Save Project</td>
<td>Available in all views except the Project view. Saves the current project.</td>
</tr>
<tr>
<td>New Project</td>
<td>Available only in the Project view. Opens a new project.</td>
</tr>
<tr>
<td>New Workspace...</td>
<td>Available only in the Project view. Creates a project workspace. Prompts you to select projects to add to the workspace. See Select Projects to Include in Workspace Dialog Box on page 3-14.</td>
</tr>
</tbody>
</table>

New Dialog Box

You use File -> New to create a new file of the selected type and add it to your project. This displays the New dialog box:

![New dialog box (File -> New)](image)

Figure 3-2: New dialog box (File -> New)
### Table 3-2: New dialog box features

<table>
<thead>
<tr>
<th><strong>Feature</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Type</strong></td>
<td>A list of supported file types. Click a type to select it. Double-clicking a type immediately creates the new file, without adding it to the project [you are prompted for a filename when you close the file].</td>
</tr>
<tr>
<td><strong>Add to Project</strong></td>
<td>This check box appears when a File Type other than Project is selected with a single click. Turn this box on to add the new file to the project when you click OK.</td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>The name of the file to be created. Do not enter a file type extension; the appropriate extension for the selected file type is added automatically.</td>
</tr>
<tr>
<td><strong>File Location</strong></td>
<td>The directory where the file is to be created.</td>
</tr>
<tr>
<td><strong>Full Path</strong></td>
<td>A read-only field that shows the full path of the file. It is updated automatically, when you enter the filename.</td>
</tr>
<tr>
<td><strong>OK</strong></td>
<td>Opens the new file. If the Add to Project check box is enabled, the file is also added to the project.</td>
</tr>
<tr>
<td><strong>Cancel</strong></td>
<td>Cancels the file creation, and closes the dialog box.</td>
</tr>
</tbody>
</table>
Open Dialog Box

You use File -> Open to select a file and open it. This displays the Open dialog box:

![Open dialog box](image)

**Table 3-3: Open dialog box features**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look in</td>
<td>The directory where the file to be opened is located. You can use the pulldown directory list and the Up One Level button to choose the desired directory.</td>
</tr>
<tr>
<td>List of files and subdirectories (large unnamed zone in the center)</td>
<td>The files and directories currently in the chosen directory. The files are those that match the file type(s) specified in the field Files of type. Select a file, then click Open to open it. Alternatively, double-click a file to open it.</td>
</tr>
<tr>
<td>File name</td>
<td>The name of the file to open. If you enter a name using the keyboard, then you must include the file-type extension.</td>
</tr>
</tbody>
</table>
Save As Dialog Box

You use File -> Save As to save a file, specifying its name, type, and directory. This displays the Save As dialog box:

![Save As dialog box](image)

**Figure 3-4: Save As dialog box (File -> Save As)**
Menubar Menus

Chapter 3: Menus, Dialog Boxes

Table 3-4: Save As dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save in</td>
<td>The directory where the file is to be saved. You can use the pulldown directory list and the Up One Level button to choose the directory.</td>
</tr>
<tr>
<td>List of files and subdirectories (large unnamed zone in the center)</td>
<td>The files and directories currently in the chosen directory. The files are those that match the file type(s) specified in the field Save as type.</td>
</tr>
<tr>
<td>File name</td>
<td>The name of the file to be saved. This field initially contains the name of the file associated with the active tool view or window. You can enter a new name for the file. Do not include a file-type extension.</td>
</tr>
<tr>
<td>Save as type</td>
<td>The type of the file to be saved. This also determines which existing files are listed, by wildcard matching.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the named file in the specified directory. If you changed the File name, then you are first asked if the newly named file is to replace the original file in the project files list.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels the file saving, and closes the dialog box.</td>
</tr>
</tbody>
</table>

Select Files to Add to Project Dialog Box

You use File -> Build Project to build a new project. When an active Text Editor window displaying an HDL file is open, File -> Build Project creates a project with the same name as that of the open file; no dialog box is displayed. When no Text Editor window is active, File -> Build Project prompts you to add files to the project, using the Select Files to Add to Project dialog box. This is the same dialog box displayed via Project -> Add Source File (see Select Files to Add to Project Dialog Box on page 3-38).

The name of the new project is the name of the first HDL file added.
Open Project Dialog Box

You use File -> Open Project to open an existing project, a new project, a new workspace, or a Project Wizard.

![Open Project dialog box](image)

Figure 3-5: Open Project dialog box (File -> Open Project)

Table 3-5: Open Project dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recent Projects</td>
<td>A list of recent projects (maintained in your .ini file).</td>
</tr>
<tr>
<td>Existing Project...</td>
<td>Displays the Open dialog box for opening an existing project.</td>
</tr>
<tr>
<td>New Project</td>
<td>Creates a new project, and places it in the Project view.</td>
</tr>
<tr>
<td>New Workspace...</td>
<td>displays the Select Projects to Include in Workspace dialog box, which you use to create a new workspace.</td>
</tr>
<tr>
<td>Project Wizard...</td>
<td>Calls up the project wizard, which helps you set up a new project or workspace. There are different kinds of projects to choose from. See Project Wizard Dialog Boxes on page 3-13 for more information.</td>
</tr>
<tr>
<td>OK</td>
<td>Opens the project selected in the list of Recent Projects (equivalent to double-clicking the project name).</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels opening a project, and closes the dialog box</td>
</tr>
</tbody>
</table>
Project Wizard Dialog Boxes

The Project Wizard walks you through the steps needed to create a project or workspace. In the first dialog box displayed (New Project), select the type of project (Synthesis or Workspace Project), give the project a name, then click Next.

![New Project dialog box (File -> Open Project -> Project Wizard)](image)

Figure 3-6: New Project dialog box (File -> Open Project -> Project Wizard)

The second dialog box displayed depends on the type of project selected, as follows:

<table>
<thead>
<tr>
<th>Project Type</th>
<th>Steps to Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis</td>
<td>Click Add Files, add the files to the project, and click OK. Click Next. Set any desired implementation, constraint, and VHDL / Verilog options on the subsequent dialog boxes. Click Finish when you are done.</td>
</tr>
<tr>
<td>Workspace Project</td>
<td>Click Add Files and add project files to the workspace. Click Finish.</td>
</tr>
</tbody>
</table>
Select Projects to Include in Workspace Dialog Box

You use File -> New Workspace to create a new project workspace, named “workspace.” This opens the Select Projects to Include in Workspace dialog box, which you use to add project files to the new project workspace.

This is the same dialog box displayed via the Project view popup menu item Insert Project; see Select Projects to Include in Workspace Dialog Box on page 3-80, for details.

After a workspace is created, you can change its name by right-clicking it in the Project view, then selecting Save Workspace from the popup menu; that displays the Save Project As dialog box, where you can specify a new name (see Save Project As Dialog Box on page 3-84).

Print Dialog Box

You use File -> Print to display the Print dialog box, where you can print the current view. You can use File -> Print Preview to preview the result, before actually printing.)

![Print dialog box](File -> Print)

Figure 3-7: Print dialog box (File -> Print)

**Note:** For more information about printing, see the appropriate License Configuration and Set Up document for your platform.
Page Setup Dialog Box

You use File -> Page Setup to set page header and footer information. This displays the Page Setup dialog box, where you provide variables whose values are to be included in the Header or Footer. For variables that specify the date and/or time, you can choose whether to use the File Time (when the file was last modified) or System Time (the current time, when printing).

![Page Setup dialog box](image)

Figure 3-8: Page Setup dialog box (File -> Page Setup)

These are the variables you can enter in the header and footer fields:

Table 3-7: Page Setup header and footer variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;p</td>
<td>Page number.</td>
</tr>
<tr>
<td>&amp;f</td>
<td>File name, including full path.</td>
</tr>
<tr>
<td>%c</td>
<td>Date (mo/day/yr) and time (hr:min:sec).</td>
</tr>
<tr>
<td>%#c</td>
<td>Long form of date and time (Sunday, October 31, 1999, 12:42:30).</td>
</tr>
<tr>
<td>%x</td>
<td>Date (mo/day/yr).</td>
</tr>
<tr>
<td>%X</td>
<td>Time (hr:min:sec).</td>
</tr>
<tr>
<td>%#x</td>
<td>Long form of date (Sunday, October 31, 1999).</td>
</tr>
</tbody>
</table>
Print Setup Dialog Box

You use File -> Print Setup to specify printing parameters. This displays the Print Setup dialog box:

![Print Setup dialog box](image)

Figure 3-9: Print Setup dialog box (File -> Print Setup)

Edit Menu

You use the Edit menubar menu to edit project and source files. This includes cutting, copying, pasting, finding and replacing text, manipulating bookmarks, and commenting-out code lines. Which Edit menu commands are available at any time depends on which window or view (Project, Text Editor, SCOPE spreadsheet, RTL, Technology) is active. This is the Edit menu in the Text Editor view:

The Edit menu commands and their associated dialog boxes are described in the following sections:

- *Edit Menu Commands* on page 3-17
- *Find Dialog Box* on page 3-19
- *Replace Dialog Box* on page 3-20
- *Goto Dialog Box* on page 3-21
Edit Menu Commands

The following table describes the Edit menu commands available in different views and windows:

Table 3-8: Edit menu commands

**Edit Menu Commands in the Text Editor View**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Undoes the last action.</td>
</tr>
<tr>
<td>Redo</td>
<td>Performs the action undone by Undo.</td>
</tr>
<tr>
<td>Cut</td>
<td>Removes the selected text, and makes it available to be Pasted.</td>
</tr>
<tr>
<td>Copy</td>
<td>Duplicates the selected text, and makes it available to be Pasted.</td>
</tr>
<tr>
<td>Paste</td>
<td>Places a copy of the most recently Cut or Copy’d text at a selected location.</td>
</tr>
<tr>
<td>Select All</td>
<td>Selects all text in the file.</td>
</tr>
<tr>
<td>Find...</td>
<td>Searches the file for text matching a given search string. See <em>Find Dialog Box on page 3-19</em>.</td>
</tr>
<tr>
<td>Replace...</td>
<td>Finds and replaces text. See <em>Replace Dialog Box on page 3-20</em>.</td>
</tr>
<tr>
<td>Goto</td>
<td>Goes to specific line number. See <em>Goto Dialog Box on page 3-21</em>.</td>
</tr>
<tr>
<td>Advanced -&gt; Comment Code</td>
<td>Inserts the appropriate comment prefix at the current text cursor location.</td>
</tr>
<tr>
<td>Advanced -&gt; Uppercase</td>
<td>Makes the selected string all upper case.</td>
</tr>
<tr>
<td>Advanced -&gt; Lowercase</td>
<td>Makes the selected string all lower case.</td>
</tr>
<tr>
<td>Toggle bookmark</td>
<td>Alternately inserts and removes a bookmark to the line containing the text cursor.</td>
</tr>
<tr>
<td>Next bookmark</td>
<td>Takes you to the next bookmark.</td>
</tr>
<tr>
<td>Previous bookmark</td>
<td>Takes you to the previous bookmark.</td>
</tr>
<tr>
<td>Delete all bookmarks</td>
<td>Removes all bookmarks from the Text Editor window.</td>
</tr>
</tbody>
</table>
Table 3-8: Edit menu commands (Continued)

**Edit Menu Commands in the SCOPE Window**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Undo</strong></td>
<td>Undoes the last action. The full name of the menu item reflects the specific action to be undone; for example, <em>Undo Insert Rows</em> stands for undoing the action of inserting rows.</td>
</tr>
<tr>
<td><strong>Redo</strong></td>
<td>Performs the action undone by <em>Undo</em>. The full name of the menu item reflects the specific action to be redone; for example, <em>Redo Insert Rows</em> stands for redoing the action of inserting rows.</td>
</tr>
<tr>
<td><strong>Cut</strong></td>
<td>Removes the selected text, and makes it available to be <em>Pasted</em>.</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>Duplicates the selected text, and makes it available to be <em>Pasted</em>.</td>
</tr>
<tr>
<td><strong>Paste</strong></td>
<td>Places a copy of the most recently <em>Cut</em> or <em>Copy</em>’d text at a selected location.</td>
</tr>
<tr>
<td><strong>Clear</strong></td>
<td>Clears a row of information.</td>
</tr>
<tr>
<td><strong>Fill Down</strong></td>
<td>Copies the information in a cell to other cells below it in the same column. Click the cell you want copied, highlight the cells below it where you want the information filled in, then click <em>Fill Down</em>.</td>
</tr>
<tr>
<td><strong>Remove Rows</strong></td>
<td>Removes selected rows from the SCOPE spreadsheet.</td>
</tr>
<tr>
<td><strong>Insert Row</strong></td>
<td>Inserts a new row above the selected row, or, when no row is selected, at the bottom of the spreadsheet.</td>
</tr>
<tr>
<td><strong>Insert Wizard...</strong></td>
<td>Opens a wizard to help you define constraints for a particular SCOPE pane, or attributes for an object. See <em>SCOPE Constraint Wizards</em> on page 6-18, for descriptions of the constraint wizards, and <em>SCOPE Attributes Wizard</em> on page 7-12 for information about the attributes wizard.</td>
</tr>
<tr>
<td><strong>Insert Quick</strong></td>
<td>Automatically inserts constraints for all objects that apply to the current SCOPE pane (for example all clocks in the <em>Clocks</em> panel).</td>
</tr>
</tbody>
</table>
Menubar Menus

Chapter 3: Menus, Dialog Boxes . . .

Table 3-8: Edit menu commands (Continued)

Find... Searches the current SCOPE column for text matching a given search string. This is similar to the Find command in the Text Editor (see Find Dialog Box on page 3-19), but in the SCOPE window you cannot use a regular expression for the search string, there is no wrap-around searching, and there is no line-number highlighting (Mark All).

Replace... Finds and replaces text. This is similar to the Replace command in the Text Editor (see Replace Dialog Box on page 3-20), but in the SCOPE window you cannot use a regular expression for the search string, there is no wrap-around search, and the search direction is always down.

Find again Continues the search initiated by the last Find.

Edit Menu Commands in RTL/Technology Views

Copy Image Lets you select and copy elements from your existing schematic in the RTL or Technology view.

Find Dialog Box

Search for text in the Text Editor.

Figure 3-10: Find dialog box (Edit -> Find)
Table 3-9: Find dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find What</td>
<td>Search string matching the text to be found. You can use the pulldown list to view and reuse search strings used previously in the current session.</td>
</tr>
<tr>
<td>Match Case</td>
<td>When enabled, searching is case sensitive.</td>
</tr>
<tr>
<td>Regular expression</td>
<td>When enabled, wildcard characters (*) and (?) can be used in the search string: ? matches any single character; * matches any string of characters, including the empty string.</td>
</tr>
<tr>
<td>Wrap around search</td>
<td>When enabled, searching starts again after reaching the end (Down) or the beginning (Up) of the file.</td>
</tr>
<tr>
<td>Direction</td>
<td>Radio buttons for selecting the search direction (Up or Down).</td>
</tr>
<tr>
<td>Find Next</td>
<td>Initiates a search for the search string (see Find What).</td>
</tr>
<tr>
<td>Mark All</td>
<td>Highlights the numbers of all lines containing text matching the search string. Closes the dialog box.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels searching and closes the dialog box.</td>
</tr>
</tbody>
</table>

Replace Dialog Box

Finds and optionally replaces text in the Text Editor.

![Replace dialog box](image)

Figure 3-11: Replace dialog box (Edit -> Replace)
### Table 3-10: Replace dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find What</td>
<td>Search string matching the text to be found. You can use the pulldown list to view and reuse search strings used previously in the current session.</td>
</tr>
<tr>
<td>Replace With</td>
<td>Replacement text, which will replace found text. You can use the pulldown list to view and reuse replacement text used previously in the current session.</td>
</tr>
<tr>
<td>Match Case</td>
<td>When enabled, searching is case sensitive.</td>
</tr>
<tr>
<td>Regular expression</td>
<td>When enabled, wildcard characters (*) and (?) can be used in the search string: ? matches any single character; * matches any string of characters, including the empty string.</td>
</tr>
<tr>
<td>Wrap around search</td>
<td>When enabled, searching starts again after reaching the end (Down) or the beginning (Up) of the file.</td>
</tr>
<tr>
<td>Direction</td>
<td>Radio buttons for selecting the search direction (Up or Down).</td>
</tr>
<tr>
<td>Find Next</td>
<td>Initiates a search for the search string (see Find What).</td>
</tr>
<tr>
<td>Replace</td>
<td>Replaces the found text with the replacement text, and locates the next match.</td>
</tr>
<tr>
<td>Replace All</td>
<td>Replaces all text that matches the search string.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels replacement and closes the dialog box.</td>
</tr>
</tbody>
</table>

### Goto Dialog Box

Goes to a specified line number in the Text Editor.

![Goto Dialog Box](image)

**Figure 3-12:** Goto dialog box (Edit -> Goto)
View Menu

The View menubar menu varies with the active view. You use this menu to set the display and viewing options, choose which toolbars are to be visible (Project view), and display result files.

The View menu commands and their associated dialog boxes are described in the following sections:

- View Menu Commands on page 3-22
- Toolbars Dialog Box on page 3-24
- Customize Dialog Box on page 3-26
- Sheet Selection Dialog Box on page 3-27
- Display Settings Dialog Box on page 3-28

View Menu Commands

The following table describes the View menu commands, beginning with the commands common to all views, and followed by the commands unique to specific views.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common View Menu Commands</td>
<td></td>
</tr>
<tr>
<td>Toolbars...</td>
<td>Defines which toolbars to display.</td>
</tr>
<tr>
<td>Status Bar</td>
<td>When enabled, context-sensitive information is displayed in the lower left corner of the main window as you move your pointer over design elements. This information includes element identification.</td>
</tr>
<tr>
<td>Workbook Mode</td>
<td>When enabled, the various views may be accessed by clicking corresponding tabs near the bottom of the Project window.</td>
</tr>
<tr>
<td>View Log File...</td>
<td>Displays the log file, which includes a net report and a performance summary on your design's speed.</td>
</tr>
<tr>
<td>View Result File...</td>
<td>Displays a detailed netlist report.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>View Menu Commands in RTL/Technology Views</strong></td>
<td></td>
</tr>
<tr>
<td>Push/Pop Hierarchy</td>
<td>Toggles traversing the hierarchy. When an vertical arrow pointer is active, clicking pushes down (down arrow) or pops up (up arrow) through the hierarchy at the pointer position, to show the next hierarchical level below or above.</td>
</tr>
<tr>
<td>Previous Sheet</td>
<td>Displays the previous sheet of a multiple-sheet schematic.</td>
</tr>
<tr>
<td>Next Sheet</td>
<td>Displays the next sheet of a multiple-sheet schematic.</td>
</tr>
<tr>
<td>View Sheets</td>
<td>Displays a dialog box where you can select a sheet to display from a list of all sheets. See Sheet Selection Dialog Box on page 3-27.</td>
</tr>
<tr>
<td>Select All -&gt; Instances</td>
<td>Selects all instances, nets, or ports on the current schematic or sheet.</td>
</tr>
<tr>
<td>Select All -&gt; Nets</td>
<td></td>
</tr>
<tr>
<td>Select All -&gt; Ports</td>
<td></td>
</tr>
<tr>
<td>Goto Net Driver</td>
<td>Displays a net driver for a selected net.</td>
</tr>
<tr>
<td>Select Net Driver</td>
<td>Selects the driver of a selected net.</td>
</tr>
<tr>
<td>Select Net Instances</td>
<td>Selects instances driven by the net.</td>
</tr>
<tr>
<td>Unselect All</td>
<td>Unselects all elements in the view.</td>
</tr>
<tr>
<td>Zoom In</td>
<td>Toggles zooming in. When the Z-shaped pointer is active, you can zoom in on the view by clicking or dragging a box around (lassoing) the region. Clicking zooms in on the center of the view; lassoing zooms in on the lassoed region. You can also right-click to remove the Z-shaped pointer and exit zooming mode.</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>Toggles zooming out. When the Z-shaped pointer is active, you can click to zoom out from the view. You can also right-click to remove the Z-shaped pointer and exit zooming mode.</td>
</tr>
<tr>
<td>Pan</td>
<td>Pans (scrolls) the view to show elements that might not otherwise be visible. Click in the schematic, then drag and release in the direction to pan.</td>
</tr>
<tr>
<td>Pan Center</td>
<td>Centers the window on the design.</td>
</tr>
</tbody>
</table>
Table 3-11: View menu commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full View</td>
<td>Zooms the active view so that it shows the entire the design.</td>
</tr>
<tr>
<td>Normal View</td>
<td>Zooms the active view, when you click it, to full (normal) size.</td>
</tr>
<tr>
<td>Zoom Lock</td>
<td>When enabled, if you resize the window the displayed schematic is resized proportionately, so that it occupies the same portion of the window.</td>
</tr>
</tbody>
</table>

View Menu Commands in Text Editor Window

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Line Numbers</td>
<td>Displays consecutive line numbers for lines of code in the file.</td>
</tr>
</tbody>
</table>

View Menu Commands in SCOPE Window

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom In</td>
<td>Zooms in on the spreadsheet a small amount.</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>Zooms out from the spreadsheet a small amount.</td>
</tr>
<tr>
<td>Normal View</td>
<td>Zooms the spreadsheet to full (normal) size.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Locks the SCOPE spreadsheet, to prevent changes.</td>
</tr>
<tr>
<td>Properties...</td>
<td>Sets the display settings for the SCOPE spreadsheet. See Display Settings Dialog Box.</td>
</tr>
</tbody>
</table>

Toolbars Dialog Box

Toolbar icons (iconic action buttons) exist for several commands in the File, Edit, View, and HDL Analyst menus (see Toolbars on page 3-88). You can add icons to any of the existing toolbars or you can create additional, custom toolbars with your own choice of icons.
You use View -> Toolbars to choose which toolbars are displayed, and to specify their icons.

![Toolbars dialog box](image)

**Figure 3-13: Toolbars dialog box (View -> Toolbars)**

**Table 3-12: Toolbars dialog box features**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolbars</td>
<td>Lists the standard and custom toolbars. You enable just those toolbars that you want to display.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the dialog box.</td>
</tr>
<tr>
<td>New...</td>
<td>Prompts for the name of a new custom toolbar. You enter a name, then click OK. This displays a new, empty toolbar, and opens the Customize dialog box, where you can add icons to the toolbar. See Customize Dialog Box on page 3-26.</td>
</tr>
<tr>
<td>Customize...</td>
<td>Displays the Customize dialog box, where you can specify which icons belong to each toolbar (standard or custom). See Customize Dialog Box on page 3-26.</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets a selected standard (predefined) toolbar to its default set of icons. (Only available when a standard toolbar is selected.)</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes a selected custom toolbar. (Only available when a custom toolbar is selected.)</td>
</tr>
<tr>
<td>Toolbar name</td>
<td>Displays the name of the selected toolbar. You can edit the names of custom toolbars.</td>
</tr>
</tbody>
</table>
Table 3-12: Toolbars dialog box features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Tooltips</td>
<td>When enabled, a descriptive tooltip is displayed whenever you position the pointer over a toolbar icon.</td>
</tr>
<tr>
<td>Cool Look</td>
<td>When enabled, toolbar icons are outlined only when you pass the mouse pointer over them; otherwise, they blend with the background. When disabled, toolbar icons are always outlined.</td>
</tr>
<tr>
<td>Large Buttons</td>
<td>When enabled, large toolbar icons are used.</td>
</tr>
</tbody>
</table>

**Customize Dialog Box**

The Customize dialog box is displayed when you click Customize... in the Toolbars dialog box, or when you enter a custom toolbar name after clicking New in the Toolbars dialog box. You use the Commands panel of the dialog box to define which icons belong to which toolbars. You do this by dragging and dropping individual icons to or from any displayed toolbars (standard or custom). The Toolbars panel of the dialog box offers the same operations as the Toolbars dialog box (see Toolbars Dialog Box on page 3-24).

![Figure 3-14: Toolbars panel, Customize dialog box](image-url)
Sheet Selection Dialog Box

View -> View Sheets displays the Sheet Selection dialog box, which you use to select a particular sheet for display or print. This dialog box is only available in an RTL or Technology view, and only when a multiple-sheet design is present.
Display Settings Dialog Box

View -> Properties is only available in the SCOPE view. It displays the Display Settings dialog box, where you can edit SCOPE view settings: row, line, and button settings, and colors for table lines.

![Display Settings dialog box](image)

**Figure 3-17**: Display Settings dialog box (View -> Properties, in SCOPE view)

**Table 3-13**: Display Settings dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preview</td>
<td>A preview of the current dialog box settings, so you can see their effect before accepting them by clicking OK.</td>
</tr>
<tr>
<td>3D-Buttons</td>
<td>When enabled, row numbers and column headings are displayed as apparently 3-dimensional buttons.</td>
</tr>
<tr>
<td>Vertical Lines</td>
<td>When enabled, vertical lines are shown between columns.</td>
</tr>
<tr>
<td>Horizontal Lines</td>
<td>When enabled, horizontal lines are shown between rows.</td>
</tr>
<tr>
<td>Mark Current Row</td>
<td>When enabled, the selected row’s number is highlighted; for example, its 3D button appears depressed.</td>
</tr>
</tbody>
</table>
Menubar Menus

Chapter 3: Menus, Dialog Boxes . . .

Project Menu

You use the Project menubar menu to set implementation options, add or remove files from a project, change project filenames, and create new implementations. The Project menu commands are the same in all views.

The Project menu commands and their associated dialog boxes are described in the following sections:

- *Project Menu Commands* on page 3-30
- *Options for implementation Dialog Box* on page 3-31
- *Select Files to Add to Project Dialog Box* on page 3-38
- *Source File Dialog Box* on page 3-40
- *File Options Dialog Box (VHDL)* on page 3-41
- *Options for implementation Dialog Box (New Implementation)* on page 3-41

---

Table 3-13: Display Settings dialog box features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark Current Column</td>
<td>When enabled, the selected column’s heading is highlighted; for example, its 3D button appears depressed.</td>
</tr>
<tr>
<td>Color</td>
<td>Sets the color for the selected spreadsheet feature. For example, if Grid Lines is selected, then horizontal and vertical grid lines are set to appear in the selected color.</td>
</tr>
<tr>
<td>User Properties</td>
<td>Sets the value of the indicated Attribute to the Value you choose in the pulldown list.</td>
</tr>
</tbody>
</table>
Project Menu Commands

The following table describes the Project menu commands and functions.

Table 3-14: Project menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Options...</td>
<td>Displays the Options for implementation dialog box, which you use to set options for implementing your design. See Options for implementation Dialog Box on page 3-31.</td>
</tr>
<tr>
<td>Add Source File...</td>
<td>Displays the Select Files to Add to Project dialog box, where you can add Verilog (.v), VHDL (.vhd), Tcl script (.tcl) and constraint (.sdc) files to your design. See Select Files to Add to Project Dialog Box on page 3-38.</td>
</tr>
<tr>
<td>Remove Files From Project</td>
<td>Removes selected files from your project.</td>
</tr>
<tr>
<td>Change File...</td>
<td>Lets you replace a file in your project. See Source File Dialog Box on page 3-40.</td>
</tr>
<tr>
<td>Set VHDL Library...</td>
<td>Displays the File Options dialog box, where you choose the library (Library Name) where your VHDL files are to be synthesized. The default library is called work. See File Options Dialog Box (VHDL) on page 3-41.</td>
</tr>
<tr>
<td>New Implementation...</td>
<td>Creates a new implementation for current design. Each implementation pertains to the same design, but it can have different options settings and/or constraints for synthesis runs. Same as Project -&gt; Implementation Options (see Options for implementation Dialog Box (New Implementation) on page 3-41).</td>
</tr>
</tbody>
</table>
Options for implementation Dialog Box

You use the Options for implementation dialog box to define the selected project’s implementation options. You can access this dialog box via Project -> Implementation Options, as well as a few other ways. This is one of the most important and frequently used dialog boxes.

The dialog box includes the following panels:

- **Device Panel** on page 3-32
- **Options/Constraints Panel** on page 3-33
- **Implementation Results Panel** on page 3-33
- **Timing Report Panel** on page 3-35
- **VHDL and Verilog Panels** on page 3-36
Device Panel

You use the Device panel to set mapping options for the selected technology. See the Synplify User Guide for details. See also the vendor appendices of this reference manual, for information on options for specific technologies.

![Device panel, Options for Implementation dialog box](image)

Figure 3-19: Device panel, Options for Implementation dialog box
Options/Constraints Panel

You use the Options/Constraints panel to specify optimization switches, target frequency, and timing constraint files for design compilation. See the Synplify User Guide for details.

![Options/Constraints panel](image)

**Figure 3-20:** Options/Constraints panel, Options for Implementation dialog box

Implementation Results Panel

You use the Implementation Results panel to specify the implementation name (default: rev_1), the results directory, and the name and format of the top-level output netlist file (result file). You can also specify output constraint and netlist files. See the Synplify User Guide for more information.
The results directory is a subdirectory of the project file directory. You can change the location of the results directory, but its name must be identical to the implementation name.

Enable one or more of the optional output file check boxes to generate the corresponding Verilog netlist, VHDL netlist or vendor constraint files. To show these files in the Project files list, enable Show all files in results directory under Options -> Project View Options (see *Project View Options Dialog Box* on page 3-55).

![Implementation Results panel, Options for Implementation dialog box](image)

**Figure 3-21**: Implementation Results panel, Options for Implementation dialog box
Timing Report Panel

You use the Timing Report panel to set criteria for the output timing report. You specify the number of start and end points, and the number of critical paths to appear in the timing report.

![Timing Report Panel](image)

Figure 3-22: Timing Report panel, Options for implementation dialog box
VHDL and Verilog Panels

You use the VHDL and Verilog panels in the Options for implementation dialog box to specify various language-related options, as described in the tables below. With mixed HDL designs, both panels are present.

Figure 3-23: VHDL and Verilog panels, Options for Implementation dialog box

Table 3-15: Verilog panel features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top Level Module</td>
<td>The name of the top-level module of your design.</td>
</tr>
<tr>
<td>Include Path Order</td>
<td>The search path to be used by the include commands in the project's Verilog design files.</td>
</tr>
</tbody>
</table>
Example: *Push Tristates across Process/Block boundaries check box*

When this option is enabled (default), the tristate is pushed across the flip-flop. When disabled, it is implemented *before* the flip-flop.
Select Files to Add to Project Dialog Box

You use \texttt{Project -> Add Source File} to add source files to your project. This displays the Select Files to Add to Project dialog box. (There are other ways to access this dialog box.)
Figure 3-26: Select Files to Add to Project dialog box

Table 3-17: Select Files to Add to Project dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look in</td>
<td>The directory to look in for the file to be added. You can use the pulldown directory list and the Up One Level button to choose the directory.</td>
</tr>
<tr>
<td>List of files and subdirectories (large unnamed zone)</td>
<td>The files and directories currently in the chosen directory. The files are those that match the file type(s) specified in the field Save as type. Select a file to be added, then click &lt;- Add to add it to the list of Files to Add to Project. Alternatively, double-click a file to immediately add only it to a new project named after it (this closes the dialog box).</td>
</tr>
<tr>
<td>File name</td>
<td>The name of a file to add to the project. If you enter a name using the keyboard, then you must include the file-type extension.</td>
</tr>
</tbody>
</table>
Table 3-17: Select Files to Add to Project dialog box features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files of type</td>
<td>A description of file types, with wildcard expressions to match the file to add. Only files in the chosen directory that match the expressions are displayed in the list of files. You can use the pulldown list to choose the file types to match.</td>
</tr>
<tr>
<td>Files To Add To Project</td>
<td>The files to be added to the project. You add files to this list with the &lt;- Add and &lt;- Add All action buttons. You remove files from this list with the Remove -&gt; and Remove All -&gt; buttons.</td>
</tr>
<tr>
<td>&lt;- Add All</td>
<td>Adds all of the files currently displayed in the directory to the Files to Add to Project list.</td>
</tr>
<tr>
<td>&lt;- Add</td>
<td>Adds the file named in the File name field to the Files to Add to Project list.</td>
</tr>
<tr>
<td>Remove -&gt;</td>
<td>Removes a selected file from the Files to Add to Project list.</td>
</tr>
<tr>
<td>Remove All -&gt;</td>
<td>Removes all of the files from the Files to Add to Project list.</td>
</tr>
<tr>
<td>OK</td>
<td>Adds the files named in the Files to Add to Project list to a new project named for the first file in the list, and closes the dialog box.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels adding files, and closes the dialog box.</td>
</tr>
</tbody>
</table>

**Source File Dialog Box**

You use Project -> Change File to replace a file in the project files list with another of the same type. You must first select the file to be replaced, in the Project view, before you can use this command. This displays the Source File dialog box, where you specify the replacement file.
Figure 3-27: Source File dialog box (Project -> Change File)

**File Options Dialog Box (VHDL)**

You use Project -> Set VHDL Library to view VHDL file properties and specify the VHDL library name. This displays the File Options dialog box, where you provide this information. This is the same dialog box displayed by right-clicking a VHDL filename in the Project view and choosing File -> Options (see *File Options Dialog Box on page 3-78*).

**Options for implementation Dialog Box (New Implementation)**

You use Project -> New Implementation to create a new implementation for the selected project. This displays the Options for implementation dialog box, where you define the project’s implementation options.
This is the same dialog box displayed by Project -> Implementation Options (see Options for implementation Dialog Box on page 3-31), except that there is no list of implementations to the right of the tabbed panels.

Figure 3-28: Options for implementation dialog box

**Run Menu**

You use the Run menubar menu to perform tasks such as the following:

- Compile and synthesize a design
- Check design syntax and synthesis
- Run Tcl scripts
- Check the task status
- Navigate to the next and previous source code errors in the Text Editor window.
The Run menu commands and their associated dialog boxes are described in the following sections:

- Run Menu Commands on page 3-43
- Open Dialog Box on page 3-44
- Job Status Dialog Box on page 3-45

**Run Menu Commands**

In addition to the Run menu commands described below, there are popup menu commands available by right-clicking in the Project view. See Project View Popup Menu on page 3-73, for details.

The following table describes the Run menu commands.

Table 3-18: Run menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesize</td>
<td>Compiles and maps the design, which you can then view in the RTL and Technology views.</td>
</tr>
<tr>
<td>Compile Only</td>
<td>Compiles the design into technology-independent high-level structures. You can view the result in the RTL view.</td>
</tr>
<tr>
<td>Syntax Check</td>
<td>Runs a syntax check on your design’s coding. The status bar at the bottom of the window displays any error messages. If the active window shows an HDL file, then only that file is checked; otherwise, the entire project is checked.</td>
</tr>
<tr>
<td>Synthesis Check</td>
<td>Runs a synthesis check on your design’s coding. This includes a syntax check and a check to see if the synthesis tool could map the design to the hardware. No optimizations are carried out. The status bar at the bottom of the window displays any error messages. If the active window shows an HDL file, then only that file is checked; otherwise, the entire project is checked.</td>
</tr>
<tr>
<td>Run Tcl Script...</td>
<td>Runs a selected Tcl script. See Open Dialog Box on page 3-44.</td>
</tr>
</tbody>
</table>
Table 3-18: Run menu commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Status</td>
<td>Tells you the name of the job, and gives you the runtime and directory location of your design, during compilation. This option is enabled during synthesis. See Job Status Dialog Box on page 3-45.</td>
</tr>
<tr>
<td>Next Error</td>
<td>Shows the next error in your source code file.</td>
</tr>
<tr>
<td>Previous Error</td>
<td>Shows the previous error in your source code file.</td>
</tr>
</tbody>
</table>

Open Dialog Box

You use Run -> Run Tcl Script to run a Tcl script. This displays the Open dialog box, where you specify the Tcl script file to use. The File name area is prefilled with the wildcard string “*.tcl”, corresponding to Tcl files.

This dialog box is the same as that displayed via File -> Open, except that no Open as read-only check box is present. See Open Dialog Box on page 3-9, for an explanation of the features in the Open dialog box.

Figure 3-29: Open dialog box (Run -> Run Tcl Script)
Job Status Dialog Box

You use Run -> Job Status to monitor the synthesis jobs that are running, their run times, and their associated commands. This information is displayed in the Job Status dialog box.

You can cancel a displayed job by selecting it in the dialog box and clicking Cancel Job.

![Job Status dialog box]

To cancel a job, select it, then click the Cancel button.

Figure 3-30: Job Status dialog box

HDL Analyst Menu

The HDL Analyst menubar menu gives you multiple schematic views of your project. It has the following submenus:

- **RTL** – shows your design at a register-transfer level (see RTL Submenu on page 3-46).
- **Technology** – shows your design at a technology-primitive level (see Technology Submenu on page 3-46).
- **Visual Elite Cross Probing** – lets you access Visual Elite™, provided that you have set up that software correctly. Visual Elite is a GUI that you can use to create and edit HDL designs. Cross probing is available, in both directions, between Synplicity’s synthesis tools and Visual Elite. Visual Elite is a product of Innoveda™; for more information see http://www.innoveda.com/products/datasheets_HTML/visualelite.asp.
The HDL Analyst menu commands and the associated dialog box are described in the following sections:

- *Hdl Analyst Menu Commands* on page 3-46
- *Object Query Dialog Box* on page 3-49

**Hdl Analyst Menu Commands**

**RTL Submenu**
Before you can access the RTL view, you must successfully compile your design. The RTL submenu then offers the following views:

- Hierarchical View
- Flattened View

**Technology Submenu**
Before you can access the Technology view, you must synthesize (compile and map) your design. The Technology submenu then offers the following views:

- Hierarchical View
- Flattened View
- Flattened to Gates View
- Hierarchical Critical Path
- Flattened Critical Path

The following table describes the HDL Analyst menu commands.

---

**Note:** RTL submenu commands are available only after your design is compiled. Technology submenu commands are available only after your design is synthesized (compiled and mapped).
Table 3-19: HDL Analyst menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL -&gt; Hierarchical View</td>
<td>Shows a hierarchical schematic view of the design.</td>
</tr>
<tr>
<td>RTL -&gt; Flattened View</td>
<td>Removes the hierarchy from the schematic.</td>
</tr>
<tr>
<td>Technology -&gt; Hierarchical View</td>
<td>Displays a hierarchical schematic of the design.</td>
</tr>
<tr>
<td>Technology -&gt; Flattened View</td>
<td>Removes the hierarchy from the schematic.</td>
</tr>
<tr>
<td>Technology -&gt; Flattened to Gates View</td>
<td>Flattens the hierarchy to the primitive gate level.</td>
</tr>
<tr>
<td>Technology -&gt; Hierarchical Critical Path</td>
<td>Highlights the critical path, without flattening the hierarchy.</td>
</tr>
<tr>
<td>Technology -&gt; Flattened Critical Path</td>
<td>Highlights the critical path, after flattening the hierarchy.</td>
</tr>
<tr>
<td>Set Slack Margin...</td>
<td>Displays a dialog box where you set the slack margin value. This is subtracted from the slack time to get a slack range. Only instances within the range are listed in the log file.(^a)</td>
</tr>
<tr>
<td>Show Critical Path</td>
<td>Displays the most critical path in your design.(^b)</td>
</tr>
<tr>
<td>Show Timing Information</td>
<td>Annotates the schematic view with timing numbers.(^b)</td>
</tr>
<tr>
<td>Filter Schematic</td>
<td>Filters the selected objects and groups them together on a new schematic.(^a)</td>
</tr>
<tr>
<td>Flatten current schematic</td>
<td>Displays a flattened view of the current schematic.(^a)</td>
</tr>
<tr>
<td>Flatten current schematic to gates</td>
<td>Displays a flattened view, at the gate level, of the current schematic.(^b)</td>
</tr>
</tbody>
</table>
Table 3-19: HDL Analyst menu commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| Find...                        | Displays the Object Query dialog box, which lets you search for instances, symbols, nets, and ports, by name. See Object Query Dialog Box on page 3-49.  
|                                | a. Available only in the RTL and Technology views.                          |
| Expand                         | Expands a selected input or output net in a filtered view, to show the adjacent object in the path.  
|                                | a. Available only in the Technology view.                                   |
| Expand to register/port        | Expands a selected input or output net in a filtered view, to show everything in the path, until a register or a port is reached.  
|                                | a. Available only in the Technology view.                                   |
| Expand paths                   | Shows all logic between two or more instances.                              |
| Visual Elite Cross Probing     | Connects the Synplify synthesis tool to Visual Elite, or disconnects it if already connected. Visual Elite must be set up correctly for this to be available. See Crossprobing Across Views on page 5-8, for details.  
| -> Connect to Visual Elite     | b. Available only in the Technology view.                                   |
| -> Disconnect from Visual Elite|                                                                             |
| Visual Elite Cross Probing -> Cross Probing Enabled | When turned on, enables crossprobing between the synthesis tool and Visual Elite. See Crossprobing Across Views on page 5-8, for details.  
|                                | a. Available only in the Technology view.                                   |
| External Cross Probing Engaged | Enables crossprobing to an external text file. See Crossprobing with ModelSim on page 5-11 for an example.  
|                                | b. Available only in the Technology view.                                   |

a. Available only in the RTL and Technology views.
b. Available only in the Technology view.
Object Query Dialog Box

You use HDL Analyst -> Find to display the Object Query dialog box, which lets you search for instances, symbols, nets, and ports, by name. The located elements are highlighted in red in the RTL or Technology view. The elements available for searching are those in the current hierarchy level; the located elements can appear at any hierarchy level, and on multiple sheets.

Figure 3-31: Object Query dialog box (HDL Analyst -> Find)

Table 3-20: Object Query dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances, Symbols,</td>
<td>Tabbed panels for finding different kinds of objects. Each panel shows a list of all objects of the corresponding type at the current hierarchy level (in the Unhighlighted zone). Choose a panel for a given object type by clicking its tab.</td>
</tr>
<tr>
<td>Nets, Ports</td>
<td></td>
</tr>
<tr>
<td>Unhighlighted</td>
<td>Names of all objects of the current panel's type. To select an object for highlighting, click its name in the list (select multiple names by pressing the Ctrl or Shift key while clicking) or use Highlight Wildcard (see below) to select a set of matching objects. The number of objects selected for highlighting, and the total number listed, are displayed above the list: # selected / # total.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3-20: Object Query dialog box features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highlight Wildcard (*)</td>
<td>Selects names in the Unhighlighted area, based on a match string you input. The string can contain the pattern-matching characters *, which matches any sequence of characters, and ?, which matches any single character. You can reuse previously entered match strings via the pulldown list.</td>
</tr>
<tr>
<td>-&gt;</td>
<td>Moves the selected names from the Unhighlighted area to the Highlighted area, and highlights their objects in the RTL and Technology views.</td>
</tr>
<tr>
<td>&lt;-</td>
<td>Moves the selected names from the Highlighted area to the Unhighlighted area, and unhighlights their objects in the RTL and Technology views.</td>
</tr>
<tr>
<td>All -&gt;</td>
<td>Moves all names from the Unhighlighted to the Highlighted area, and highlights their objects in the RTL and Technology views.</td>
</tr>
<tr>
<td>&lt;- All</td>
<td>Moves all names from the Highlighted to the Unhighlighted area, and unhighlights their objects in the RTL and Technology views.</td>
</tr>
<tr>
<td>Highlighted</td>
<td>Complementary and analogous to the Unhighlighted area. You select object names here as candidates for moving to the Unhighlighted list, which unhighlights their corresponding objects.</td>
</tr>
<tr>
<td>Unhighlight Wildcard (*)</td>
<td>Complementary and analogous to the Unhighlight Wildcard area: selects names in the Highlighted area, based on a match string you input.</td>
</tr>
<tr>
<td>Jump Location</td>
<td>When enabled, the target objects are displayed.</td>
</tr>
<tr>
<td>OK</td>
<td>Closes the Object Query dialog box.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels any changes to object highlighting made since opening the dialog box, and closes it.</td>
</tr>
</tbody>
</table>
Format Menu

The Format menubar menu is available only when the SCOPE view is active. You can use it to configure SCOPE spreadsheet cells, rows and columns, and to set text alignment and style.

The Format menu commands and their associated dialog boxes are described in the following sections:

- *Format Menu Commands* on page 3-51
- *Cells Dialog Box* on page 3-52
- *Styles Dialog Box* on page 3-52

Format Menu Commands

The following table describes the Format menu commands.

Table 3-21: Format menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells...</td>
<td>Specifies the font and color to use, and adds borders to the cells. See <em>Cells Dialog Box</em> on page 3-52.</td>
</tr>
<tr>
<td>Resize Rows</td>
<td>Changes the row height to fit the text.</td>
</tr>
<tr>
<td>Resize Columns</td>
<td>Changes the column width to fit the text.</td>
</tr>
<tr>
<td>Cover Cells</td>
<td>Temporarily hides cells from view.</td>
</tr>
<tr>
<td>Remove Covering</td>
<td>Removes the covering previously placed on cells by <em>Cover Cells</em>, so they become visible again.</td>
</tr>
<tr>
<td>Styles...</td>
<td>Changes the styles of columns, rows and warnings. See <em>Styles Dialog Box</em> on page 3-52.</td>
</tr>
<tr>
<td>Align</td>
<td>Aligns text in columns and rows to the left, center, or right.</td>
</tr>
<tr>
<td>Style</td>
<td>Makes text bold, italic, or underlined.</td>
</tr>
</tbody>
</table>
Cells Dialog Box

You access the Cells dialog box via Format > Cells. You use the Cells dialog box to set options like font, color, border, and alignment for the SCOPE spreadsheet cells. The dialog box has panels for the different options.

![Cells Dialog Box](image)

Figure 3-32: Cells dialog box (Format -> Cells)

Styles Dialog Box

You access the Styles dialog box via Format -> Styles. You use the Styles dialog box to change the styles of your columns, rows, and warnings. Select the item to change, then click the Change button to display a dialog box for the selected item that is equivalent to the Cells dialog box (see Cells Dialog Box on page 3-52).
Options Menu

You use the Options menubar menu to configure the VHDL and Verilog compilers, customize toolbars, and set options for the schematic display, Project view and Text Editor. When using certain technologies, additional menu items let you run the vendor's software from this menu.

The Options menu commands and their associated dialog boxes are described in the following sections:

- *Options Menu Commands* on page 3-54
- *Options for implementation Dialog Box – VHDL* on page 3-54
- *Options for implementation Dialog Box – Verilog* on page 3-55
- *Customize Dialog Box* on page 3-55
- *Project View Options Dialog Box* on page 3-55
- *Editor Options Dialog Box* on page 3-56
- *Schematic Options Dialog Box* on page 3-59
Options Menu Commands

The following table describes the Options menu commands.

Table 3-22: Options menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure VHDL Compiler...</td>
<td>Lets you set the encoding method for enumerated types, and the top-level entity. See Options for implementation Dialog Box – VHDL on page 3-54.</td>
</tr>
<tr>
<td>Configure Verilog Compiler...</td>
<td>Lets you specify the top-level module and the 'include search path. See Options for implementation Dialog Box – Verilog on page 3-55.</td>
</tr>
<tr>
<td>Customize...</td>
<td>Customizes your toolbars. See Customize Dialog Box on page 3-55.</td>
</tr>
<tr>
<td>Project View Options...</td>
<td>Sets options for organizing files in the Project view. See Project View Options Dialog Box on page 3-55.</td>
</tr>
<tr>
<td>Editor Options...</td>
<td>Sets your syntax coloring, font, and tabs, for the Text Editor. See Editor Options Dialog Box on page 3-56.</td>
</tr>
<tr>
<td>Schematic Options...</td>
<td>Sets display preferences for the RTL and Technology views. See Schematic Options Dialog Box on page 3-59.</td>
</tr>
<tr>
<td>Quartus (Altera only)</td>
<td>Lets you set place-and-route options or run Quartus II placement and routing in either foreground or background.</td>
</tr>
<tr>
<td>Xilinx (Xilinx only)</td>
<td>Runs Design Manager, Floorplanner, or ISE Project Navigator from the Synplicity environment.</td>
</tr>
</tbody>
</table>

Options for implementation Dialog Box – VHDL

You use Options -> Configure VHDL Compiler to configure the VHDL compiler. This displays the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31), where you use the VHDL panel (see VHDL and Verilog Panels on page 3-36) to set the top-level entity and the encoding method for enumerated types. Note that state-machine encoding is automatically determined by the FSM compiler, or you can specify it explicitly using the syn_encoding attribute.
Options for implementation Dialog Box – Verilog

You use Options -> Configure Verilog Compiler to configure the Verilog compiler. This displays the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31), where you use the Verilog panel (see VHDL and Verilog Panels on page 3-36) to specify the top-level module and the ‘include search path.

Customize Dialog Box

You use Options -> Customize to customize and enable toolbars. This displays the Customize dialog box (see Customize Dialog Box on page 3-26), which has two panels, Toolbars and Commands. You use the Toolbars panel to select and enable existing toolbars, and create new, custom toolbars. You use the Commands panel to customize the selected toolbar. For more information on customizing toolbars, see Toolbars on page 3-88.

Project View Options Dialog Box

You use Options -> Project View Options to define how projects appear and are organized. This displays the Project View Options dialog box, which lets you work on multiple projects and define how files are arranged in them.

Figure 3-34: Project View Options dialog box (Options -> Project View Options)
The following table describes the Project View Options dialog box features.

Table 3-23: Project View Options dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description: When Feature is Enabled . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Project File Library</td>
<td>When enabled, displays the corresponding VHDL library next to each source filename, in the Project Tree view of the Project view. (See also File Options Dialog Box (VHDL) on page 3-41, for how to change a file's library.)</td>
</tr>
<tr>
<td>Beep when a job completes</td>
<td>An audible signal is given whenever a project finishes running.</td>
</tr>
<tr>
<td>View project files in folders</td>
<td>Project files are organized into separate folders, by type.</td>
</tr>
<tr>
<td>Show file name only</td>
<td>Displays only filenames.</td>
</tr>
<tr>
<td>Show relative file path</td>
<td>Displays relative paths, as well as filenames.</td>
</tr>
<tr>
<td>Show full file path</td>
<td>Displays full paths and filenames.</td>
</tr>
</tbody>
</table>

For more information on using the Project View Options dialog box, see Setting Project View Display Preferences on page 2-12.

**Editor Options Dialog Box**

You use Options -> Editor Options to choose either an internal or an external text editor. This displays the Editor Options dialog box, where you specify the editor to use.
Synplify Reference Manual, April 2002

Table 3-24: Editor Options dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Editor</td>
<td>Sets the Synplicity Text Editor as the default text editor.</td>
</tr>
<tr>
<td>Options</td>
<td>(Only available when Internal Editor is enabled.) Lets you define the whitespace format, syntax coloring, and font to use for selected file types. See Editor Color, Font, and Tab Options on page 3-57.</td>
</tr>
<tr>
<td>External Editor</td>
<td>Uses the specified external text editor program to view text files from within the Synplicity tool. Note that files opened with an external editor cannot be crossprobed.</td>
</tr>
<tr>
<td>...</td>
<td>You can use this button to browse and select an external editor program.</td>
</tr>
</tbody>
</table>

Editor Color, Font, and Tab Options

The Options action button of the Editor Options dialog box (Internal Editor only) lets you define various text editing preferences for the built-in Synplicity Text Editor.
You can use this editor for the following types of files:

Verilog       VHDL       Constraint
Tcl            Log         C/C++
Cyn++          default

Select a file type, then click OK. A second dialog box then opens (also named Editor Options). You can use this box to define the whitespace format, syntax coloring, and font to use for the selected file type. See the Synplify User Guide for more information.

After you choose a filetype, use Editor Options to set syntax coloring and font.

Figure 3-36: Syntax Coloring and Font dialog boxes
Schematic Options Dialog Box

You use Options -> Schematic Options to define your preferences for the HDL Analyst schematic windows (RTL and Technology views). This displays the Schematic Options dialog box, where you specify what you want.

The result is the same as setting preferences in your .ini file. Some preferences take effect immediately; others only take effect in the next view you open. For details see the Synplify User Guide.

Figure 3-37: Schematic Options dialog box
The following table describes the Schematic Options dialog box features.

Table 3-25: Schematic Options dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Hierarchy Browser</td>
<td>When enabled, the Hierarchy Browser pane is displayed, on the left side of</td>
</tr>
<tr>
<td></td>
<td>the RTL and Technology views.</td>
</tr>
<tr>
<td>Enhanced Text Crossprobing</td>
<td>When enabled, you can crossprobe to a place-and-route report file. See</td>
</tr>
<tr>
<td></td>
<td><em>Using the Enhanced Text Crossprobing Option</em> on page 3-61 for more</td>
</tr>
<tr>
<td></td>
<td>information.</td>
</tr>
<tr>
<td>Show Cell Interior</td>
<td>When enabled, the internal logic of cells is displayed.</td>
</tr>
<tr>
<td>Show Sheet Connector Index</td>
<td>When enabled, connecting sheet page numbers are displayed inside</td>
</tr>
<tr>
<td></td>
<td>sheet-connector symbols.</td>
</tr>
<tr>
<td>Compress Buses</td>
<td>When enabled, the display of buses is compressed, to reduce clutter.</td>
</tr>
<tr>
<td>No Buses in Technology View</td>
<td>When enabled, buses are not displayed; they are only indicated as bits in</td>
</tr>
<tr>
<td></td>
<td>the Technology View.</td>
</tr>
<tr>
<td>Maximum Instances</td>
<td>Defines the maximum number of instances to display on a single sheet.</td>
</tr>
<tr>
<td>Maximum Filtered Instances</td>
<td>Defines the maximum number of instances to display on a single sheet in a</td>
</tr>
<tr>
<td></td>
<td>filtered view. This must be at least the <strong>Maximum Instances</strong> value.</td>
</tr>
<tr>
<td>Maximum Sheet Pins</td>
<td>Defines the maximum number of pins on a sheet.</td>
</tr>
<tr>
<td>Maximum Instance Ports</td>
<td>Defines the maximum number of ports on an instance.</td>
</tr>
<tr>
<td>Show Text</td>
<td>Enables the display of labels. Which labels are actually displayed is</td>
</tr>
<tr>
<td></td>
<td>governed by the other <strong>Show * features</strong>, below.</td>
</tr>
<tr>
<td>Show Instance Name</td>
<td>When enabled, instance names are displayed.</td>
</tr>
<tr>
<td>Show Port Name</td>
<td>When enabled, port names are displayed.</td>
</tr>
<tr>
<td>Show Symbol Name</td>
<td>When enabled, symbol names are displayed.</td>
</tr>
<tr>
<td>Show Pin Name</td>
<td>When enabled, pin names are displayed.</td>
</tr>
<tr>
<td>Show Conn Name</td>
<td>When enabled, connectivity names are displayed.</td>
</tr>
</tbody>
</table>
Using the Enhanced Text Crossprobing Option

To use the Enhanced Text Crossprobing option, follow these steps:

1. Select a path in a text file – If the objects in the path are in a column, hold down the Alt key and drag the mouse to select the column. The column is highlighted.

2. Right-click.

3. To select all the objects in the highlighted path, choose Select All from the popup menu.

4. To further filter the objects to crossprobe:
   – Choose Select From from the popup menu.
   – In the displayed Select element list dialog box, choose the objects that you want to cross probe, then click OK. The objects are then highlighted where they appear in other views.

Table 3-25: Schematic Options dialog box features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font</td>
<td>Shows the current text font. To change it, click Change, then choose a new font from the displayed list.</td>
</tr>
<tr>
<td>Size</td>
<td>Shows the current font size. To change it, click Change, and choose a new font size from the displayed list.</td>
</tr>
<tr>
<td>Change</td>
<td>Displays a dialog box where you can change Font Options (Font and Size).</td>
</tr>
</tbody>
</table>
Window Menu

You use the Window menubar menu to do tasks such as the following:

• create a new window as a copy (clone) of the current window (except the Project View),
• arrange windows, by cascading or tiling them,
• arrange minimized windows (icons),
• close all windows,
• quickly raise and activate a window.

Note: The windows manipulated with the Window menu do not include the Tcl window or the Log Watch window.

Window Menu Commands

The following table describes the Window menu commands.

Table 3-26: Window menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Window</td>
<td>Creates a new window as a copy (clone) of the current (active) window. Not available in the Project view.</td>
</tr>
<tr>
<td>Cascade</td>
<td>Stacks all windows so that they are slightly offset from each other. See Cascade on page 3-63.</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>Tiles views horizontally, so that they do not overlap. See Tile Horizontally on page 3-63.</td>
</tr>
<tr>
<td>Tile Vertically</td>
<td>Tiles views vertically, so that they do not overlap. See Tile Vertically on page 3-64.</td>
</tr>
<tr>
<td>Arrange Icons</td>
<td>Arranges minimized windows along the bottom of the overall application window.</td>
</tr>
<tr>
<td>Close All</td>
<td>Closes all windows except the Project view.</td>
</tr>
</tbody>
</table>

Commands Cascade, Tile Horizontally, and Tile Vertically determine how existing windows are arranged.
Cascade

Figure 3-38: Cascading windows display

Window -> Cascade stacks windows on top of each other, offsetting them slightly.

Tile Horizontally

Figure 3-39: Horizontally tiled windows display

Window -> Tile Horizontally arranges windows horizontally, in such a way that they do not overlap.
Tile Vertically

Figure 3-40: Vertically tiled windows display

Window -> Tile Vertically arranges windows vertically, in such a way that they do not overlap.

Help Menu

The Help menubar menu changes according to the view—SCOPE spreadsheet, Project, RTL, Technology or Text Editor. Menu commands give information about the help system, additional products, finding online documentation, licensing, and using the Synplify synthesis tool. Some commands are only available in certain views.

The Help menu commands and their associated dialog boxes are described in the following sections:

- *Help Menu Commands on page 3-65*
- *Help Topics Dialog Box on page 3-66*
- *License Wizard Dialog Boxes on page 3-67*
- *License Editor Dialog Box on page 3-69*
- *Select Preferred License Dialog Box on page 3-71*
- *Tip of the Day Dialog Box on page 3-72*
Help Menu Commands

The following table describes the Help menu commands.

Table 3-27: Help menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help</td>
<td>Displays hyperlinked online help for the product.</td>
</tr>
<tr>
<td>Additional Products</td>
<td>Displays a form that you can fill out, fax, or e-mail to Synplicity, to request product information.</td>
</tr>
<tr>
<td>How to Use Help</td>
<td>Displays online help on how to use the online help (!).</td>
</tr>
<tr>
<td>Online Documents</td>
<td>Displays hyperlinked PDF documentation on the product: release notes, user guide, reference manual, and configuration and setup help. You need Adobe’s Acrobat Reader to view the PDF files.</td>
</tr>
<tr>
<td>Synplify vs. Synplify Pro</td>
<td>Details the differences between these products. Only available in the Project view.</td>
</tr>
<tr>
<td>TCL Help</td>
<td>Displays help for Tcl commands.</td>
</tr>
<tr>
<td>License Agreement</td>
<td>Displays the Synplify software license agreement.</td>
</tr>
<tr>
<td>License Wizard</td>
<td>Opens the license wizard, to help you request or install your license. See License Wizard Dialog Boxes on page 3-67 for details.</td>
</tr>
<tr>
<td>License Editor</td>
<td>Lets you edit your current (node-locked) license information. See License Editor Dialog Box on page 3-69 for details. Only available in the Project view.</td>
</tr>
<tr>
<td>Floating License Usage</td>
<td>Specifies the number of floating licenses currently being used, and their users.</td>
</tr>
<tr>
<td>Preferred License Selection</td>
<td>Displays the floating licenses that are available for your selection. See Select Preferred License Dialog Box on page 3-71.</td>
</tr>
<tr>
<td>Tip of the Day</td>
<td>Displays a daily tip on how to use the Synplify synthesis tools better. See Tip of the Day Dialog Box on page 3-72.</td>
</tr>
<tr>
<td>About this program</td>
<td>Specifies the synthesis tool product name, license expiration date, version number, and copyright.</td>
</tr>
</tbody>
</table>
Help Topics Dialog Box

Most of the online help is provided via the Help Topics dialog box. This documents different things, depending on how it is accessed:

- help on the Synplify synthesis tool, via Help -> Help
- help on how to use the online help itself, via Help -> How to Use Help
- help on Tcl/Tk (standard Tcl commands), via Help -> TCL Help

In all cases, the Help Topics dialog box, has three panels: Contents, Index, and Find. For more information, try Help -> How to Use Help.

Contents Panel

Displays the table of contents for the online help. Double-click a closed book ( ) icon to list its contents. Double-click an open book ( ) icon to close it. Double-click a document icon ( ) to display its document.

Figure 3-41: Help Topics dialog box, Contents panel
Index Panel
Displays an alphabetical index of principal topics. Type the first few letters of the topic you are searching for, or scroll through the index, then double-click the topic of interest to display it.

Figure 3-42: Help Topics dialog box, Index panel

Find Panel
Searches the help database for a text string. Enter a string, then click one of the matching topics to display it.

License Wizard Dialog Boxes
You use the License Wizard to do either of the following:

- Fill out a form to request a trial license from Synplicity
- Edit or enter license information received from Synplicity
You choose which to do in the License Wizard Introduction dialog box, accessed via Help -> License Wizard. Choose one of these options by enabling the corresponding radio button, then follow the directions in subsequently displayed License Wizard dialog boxes.

![Figure 3-43: Introduction dialog box (Help -> License Wizard)](image)

If you choose Fill out a form to request a trial license from Synplicity, you will be asked for additional information regarding your context, such as name, company, host IDs, platforms, vendors, languages, and so on. You can either fax or e-mail the request to Synplicity.

If you choose Edit or enter license information received from Synplicity, then the License Editor dialog box is displayed, where you can view and edit the information received. See License Editor Dialog Box on page 3-69, for details.
License Editor Dialog Box

You use Help > License Editor to view and edit your current license information (node-locked license only), via the License Editor dialog box. Refer to the product installation notes for the most current information.

Figure 3-44: License Editor dialog box (Help -> License Editor)

Follow these steps to edit your current license information:

1. If you are using the Microsoft® Windows® operating system, and your license file is not in the recommended directory, then specify the location with the variable SYNPLICITY_LICENSE_FILE or LM_LICENSE_FILE (Environment Variable Used option).
If you are using a UNIX system, then specify the location of the license file with the variable LM_LICENSE_FILE.

2. If your licensing information (Synplicity Software Authorization form) is in the form of an e-mail or a text file, then enter the information automatically as follows:

Table 3-28: Entering licensing information automatically

<table>
<thead>
<tr>
<th>Format</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail</td>
<td>Copy the licensing information. Click the Clipboard action button. The licensing information from the copied document is automatically pasted into the dialog box.</td>
</tr>
<tr>
<td>Text file</td>
<td>Click the Text File action button, then specify the name of the file. The licensing information in the file is automatically pasted into the dialog box.</td>
</tr>
</tbody>
</table>

If the information is not in the form of an e-mail or a text file, then enter the licensing information manually in the appropriate fields.

```
# FEATURE synplify_pc synplctyd 2.590 31-dec-99 0 12345678901234567890 \
# VENDOR_STRING=fpga HOSTID=SKEY=FFFFFFFF ck=00
```

- To add a value to VENDOR_STRING, click the Add icon (Add) and type the feature name.
- To delete an entry, click the Delete icon (Delete).
- To reorder entries, select them, then use the Move Up (Move Up) and Move Down (Move Down) arrow icons to move them.
– Place HDL Analyst last on the list in a field. (The other feature names can be in any order).

3. Click Validate. If all information has been correctly entered, a happy face will display.

4. Click Save.

Select Preferred License Dialog Box

You use Help > Preferred License Selection to choose a preferred license, when multiple licenses exist. This displays the Select Preferred License dialog box, which lists the available licenses for you to choose from. Select a license from the License Type column, click Save, close the Synplify synthesis tool, then restart it. The new session will use the preferred license you selected.

Figure 3-45: Select Preferred License dialog box
Tip of the Day Dialog Box

You use Help -> Tip of the Day to display the Tip of the Day dialog box, with a daily tip on how to best use the Synplify synthesis tool. This box also opens automatically when you first start the tool. To prevent it from redisplaying at product startup, just disable Show Tips at Startup.

Figure 3-46: Tip of the Day dialog box (Help -> Tip of the Day)
Popup Menus

Popup menus, available by right-clicking, offer quick ways to access commonly used commands that are specific to the view where you click, or are otherwise dependent on the current context. Which popup menu items appear at any given time vary, depending on that context. Items that are shown grayed out (dimmed) are currently inaccessible. Most popup menu items are also available from a menubar menu.

The individual popup menus, their commands (menu items) and the associated dialog boxes, are described in the following sections:

- Project View Popup Menu on page 3-73
- SCOPE Popup Menu on page 3-84
- Text Editor Popup Menu on page 3-84
- RTL View and Technology View Popup Menus on page 3-85

Project View Popup Menu

The popup menu commands available in the Project view vary, depending on what, if anything, is currently selected. The commands and their associated dialog boxes are described in the following sections:

- Project View Popup Menu Commands on page 3-74
- File Options Dialog Box on page 3-78
- Project Properties Dialog Box on page 3-80
- Select Projects to Include in Workspace Dialog Box on page 3-80
- Select Files to Add to Project Dialog Box on page 3-82
- Open Project Dialog Box on page 3-83
- Build Workspace Dialog Box on page 3-83
- Project View Options Dialog Box on page 3-84
- Save Project As Dialog Box on page 3-84
## Project View Popup Menu Commands

The Project view popup menu varies, depending on which object in the view is selected, if any, and where in the view you open the popup menu (by right-clicking). The following tables describe the Project view popup menu commands in these various contexts.

### Table 3-29: Project view popup menu commands, nothing selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Project...</td>
<td>Opens a new or existing project. Same as File -&gt; Open Project (see <em>Open Project Dialog Box on page 3-12</em>).</td>
</tr>
<tr>
<td>New Project...</td>
<td>Creates a new project, and places it in the Project view.</td>
</tr>
<tr>
<td>Build Workspace...</td>
<td>Creates a project workspace. In the Project view, select existing projects that you want to include in the project workspace. Refer to <em>Build Workspace Dialog Box on page 3-83</em>.</td>
</tr>
<tr>
<td>Project View Options...</td>
<td>Lets you define how projects are organized in the Project view. Same as Options -&gt; Project View Options (see <em>Project View Options Dialog Box on page 3-55</em>).</td>
</tr>
<tr>
<td>Refresh</td>
<td>Cleans up the window display.</td>
</tr>
</tbody>
</table>

### Table 3-30: Project view popup menu commands, HDL source file selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Options...</td>
<td>Displays the File Options dialog box, which shows general properties of the selected file such as its modification date, path, and type. (See <em>File Options Dialog Box on page 3-78.</em>)</td>
</tr>
<tr>
<td>Open</td>
<td>Opens the selected file.</td>
</tr>
<tr>
<td>Syntax Check</td>
<td>Checks source file syntax. Same as Run -&gt; Syntax Check (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
<tr>
<td>Synthesis Check</td>
<td>Checks synthesis. Same as Run -&gt; Synthesis Check (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
</tbody>
</table>
### Table 3-30: Project view popup menu commands, HDL source file selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy File...</td>
<td>Displays the Copy File dialog box, where you copy the selected file and add it to the current project. You specify a new name for the file. (See Copy File Dialog Box on page 3-79.)</td>
</tr>
<tr>
<td>Change File...</td>
<td>Displays the Source File dialog box, where you choose a file to replace the selected file. Same as Project -&gt; Change File (see Source File Dialog Box on page 3-40).</td>
</tr>
<tr>
<td>Remove File from Project...</td>
<td>Removes the selected file from the project.</td>
</tr>
</tbody>
</table>

### Table 3-31: Project view popup menu commands, constraint file selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Options...</td>
<td>Displays the File Options dialog box, which shows general properties of the selected file such as its modification date, path, and type. (See File Options Dialog Box on page 3-78.)</td>
</tr>
<tr>
<td>Open</td>
<td>Opens the selected file.</td>
</tr>
<tr>
<td>Open as Text</td>
<td>Opens the selected file in the Text Editor.</td>
</tr>
<tr>
<td>Copy File...</td>
<td>Displays the Copy File dialog box, where you copy the selected file and add it to the current project. You specify a new name for the file. (See Copy File Dialog Box on page 3-79.)</td>
</tr>
<tr>
<td>Change File...</td>
<td>Displays the Source File dialog box, where you choose a file to replace the selected file. Same as Project -&gt; Change File (see Source File Dialog Box on page 3-40).</td>
</tr>
<tr>
<td>Remove File from Project...</td>
<td>Removes the selected file from the project.</td>
</tr>
</tbody>
</table>
### Table 3-32: Project view popup menu commands, results file selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Options...</td>
<td>Displays the File Options dialog box, which shows general properties of the selected file such as its modification date, path, and type. (See File Options Dialog Box on page 3-78.)</td>
</tr>
<tr>
<td>Open</td>
<td>Opens the selected file.</td>
</tr>
<tr>
<td>Open as Text</td>
<td>Opens the selected file in the Text Editor, even if the file is of a type, such as .sdc or .info, that is normally viewed otherwise.</td>
</tr>
<tr>
<td>Delete File</td>
<td>Removes the selected file from the Implementation Results view.</td>
</tr>
</tbody>
</table>

### Table 3-33: Project view popup menu commands, project selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Options...</td>
<td>Displays project properties such as name and location. You can click a selected implementation to make it current. Refer to Project Properties Dialog Box on page 3-80.</td>
</tr>
<tr>
<td>Open as Text</td>
<td>Opens the selected file in the Text Editor, even if the file is of a type, such as .sdc or .info, that is normally viewed otherwise.</td>
</tr>
<tr>
<td>Add Source File...</td>
<td>Adds a source file to the selected project. Same as Project -&gt; Add Source File (see Select Files to Add to Project Dialog Box on page 3-11).</td>
</tr>
<tr>
<td>New Implementation...</td>
<td>Creates a new implementation for your design.</td>
</tr>
<tr>
<td>Synthesize</td>
<td>Synthesizes (compiles and maps) your design.</td>
</tr>
<tr>
<td>Compile Only</td>
<td>Compiles your design, without mapping it.</td>
</tr>
</tbody>
</table>
Table 3-33: Project view popup menu commands, project selected (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Project</td>
<td>Saves the selected project.</td>
</tr>
<tr>
<td>Remove Project From Workspace</td>
<td>Removes the selected project from the selected workspace, upon confirmation. Available only if selected project is within a workspace.</td>
</tr>
<tr>
<td>Close Project</td>
<td>Closes the current project. Same as File -&gt; Close Project (see File Menu Commands on page 3-5). Not available if the selected project is within a workspace—use Close Workspace then, instead.</td>
</tr>
</tbody>
</table>

Table 3-34: Project view popup menu commands, workspace selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Options...</td>
<td>Displays workspace properties such as name and location. You can click a selected implementation to make it current. Refer to Project Properties Dialog Box on page 3-80.</td>
</tr>
<tr>
<td>Open as Text</td>
<td>Opens the selected file in the Text Editor, even if the file is of a type, such as .sdc or .info, that is normally viewed otherwise.</td>
</tr>
<tr>
<td>Insert Project...</td>
<td>Adds project files to the project workspace. Refer to Select Projects to Include in Workspace Dialog Box on page 3-80.</td>
</tr>
<tr>
<td>Run All Projects</td>
<td>Synthesizes the current implementation of each project in the workspace.</td>
</tr>
<tr>
<td>Save Workspace</td>
<td>Saves the selected workspace and its projects to a file you name. Refer to Save Project As Dialog Box on page 3-84.</td>
</tr>
<tr>
<td>Close Workspace</td>
<td>Closes the project workspace and its projects.</td>
</tr>
</tbody>
</table>
Chapter 3: Menus, Dialog Boxes . . .

Table 3-35: Project view popup menu commands, implementation selected

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Options...</td>
<td>Same as Project -&gt; Implementation Options (see Options for implementation Dialog Box on page 3-31).</td>
</tr>
<tr>
<td>Change Implementation Name...</td>
<td>Displays the Implementation Name dialog box, where you rename the selected implementation. (See Implementation Name Dialog Box on page 3-79.)</td>
</tr>
<tr>
<td>Copy Implementation...</td>
<td>Copies the selected implementation, and adds it to the current project. Displays the Implementation Name dialog box, where you specify a name for the copy.</td>
</tr>
<tr>
<td>Remove Implementation</td>
<td>Removes the selected implementation from the project.</td>
</tr>
<tr>
<td>Run</td>
<td>Runs synthesis on a selected implementation.</td>
</tr>
</tbody>
</table>

File Options Dialog Box

You use File Options in the Project view popup menu to view or specify various properties of a file. This displays the File Options dialog box, where you can specify the file type, and whether to save the file relative to the project or with an absolute path. It is in the project file (.prj) that the path type is used: the file is either listed with an absolute path or with no path.

For a Verilog file, you can specify the Verilog Standard to use for the file (Verilog 95 or Verilog 2001), or choose to use the project’s default Verilog standard. The default standard for the project is set via the Verilog 2001 checkbox on the Verilog panel of the Options for implementation dialog box (see VHDL and Verilog Panels on page 3-36).

For a VHDL file, you can specify the Library Name, which must be compatible with VHDL simulators. In the case of a VHDL file, the dialog box is the same as that accessed via Project -> Set VHDL Library, (see File Options Dialog Box (VHDL) on page 3-41).
Copy File Dialog Box

You use Copy File in the Project view popup menu to copy the selected file and add it to the current project. This displays the Copy File dialog box, where you specify the name of the new file.

Implementation Name Dialog Box

You use Change Implementation Name in the Project view popup menu to rename the selected implementation. This displays the Implementation Name dialog box, where you specify the new name.
Project Properties Dialog Box

You use the Project Options and Workspace Options popup menu items to change which of a project’s implementations is active. This menu item is only accessible when a project or workspace is selected in the Project view.

Choosing this item displays the Project Properties dialog box, which indicates the Project Name and Project Location (full path and filename). In the dialog box, select an implementation in the Implementations list, then click OK or Apply, to make it the project’s active implementation.

![Project Properties dialog box](image)

Figure 3-50: Project Properties dialog box

Select Projects to Include in Workspace Dialog Box

You use the Insert Project popup menu item to add projects to a workspace. This menu item is only accessible when a workspace is selected in the Project view. Both projects and workspaces can be inserted into a workspace.
Choosing this menu item displays the Select Projects to Include in Workspace dialog box, where you select the project files to add to the workspace. This is the same dialog box as that displayed via File -> New Workspace (see Select Projects to Include in Workspace Dialog Box on page 3-14).

Figure 3-51: Select Projects to Include in Workspace dialog box

Table 3-36: Select Projects to Include in Workspace dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look in</td>
<td>The directory to look in for the projects to be included. You can use the pulldown directory list and the Up One Level button to choose the directory.</td>
</tr>
<tr>
<td>File name</td>
<td>The name of a project file to add to the workspace. If you enter a name using the keyboard, then you must include the file-type extension here.</td>
</tr>
</tbody>
</table>
Chapter 3: Menus, Dialog Boxes . . .

Table 3-36: Select Projects to Include in Workspace dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files of type</td>
<td>A description of file types, with wildcard expressions to match the file to add. Only files in the chosen directory that match the expressions are displayed in the list of files. You can use the pulldown list to choose the file types to match. However, only files of type .prj can be added to a workspace.</td>
</tr>
<tr>
<td>Files To Add To Project</td>
<td>The project files to be added to the workspace. You add files to this list with the &lt;- Add and &lt;- Add All action buttons. You remove files from this list with the Remove -&gt; and Remove All -&gt; buttons.</td>
</tr>
<tr>
<td>&lt;- Add All</td>
<td>Adds all of the files currently displayed in the directory to the Files to Add to Project list.</td>
</tr>
<tr>
<td>&lt;- Add</td>
<td>Adds the file named in the File name field to the Files to Add to Project list.</td>
</tr>
<tr>
<td>Remove -&gt;</td>
<td>Removes a selected file from the Files to Add to Project list.</td>
</tr>
<tr>
<td>Remove All -&gt;</td>
<td>Removes all of the files from the Files to Add to Project list.</td>
</tr>
<tr>
<td>OK</td>
<td>Adds the project files named in the Files to Add to Project list to the workspace, and closes the dialog box.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels adding files, and closes the dialog box.</td>
</tr>
</tbody>
</table>

Select Files to Add to Project Dialog Box

You use the Add Source File popup menu item to add files to an existing project. This menu item is only accessible when a project is selected in the Project view.

Choosing this item displays the Select Files to Add to Project dialog box, where you select the files to add to the project. This is the same dialog box displayed via Project -> Add Source File (see Select Files to Add to Project Dialog Box on page 3-38).

Options for implementation Dialog Box

You use the New Implementation popup menu item to define the selected project’s implementation options. This menu item is only accessible when a project is selected in the Project view.
Choosing this menu item displays the Options for implementation dialog box. This is the same dialog box displayed via Project -> Implementation Options (see Options for implementation Dialog Box on page 3-31).

**Open Project Dialog Box**

You use the Open Project popup menu item to open a project or workspace. This menu item is only accessible when nothing is selected in the Project view.

Choosing this menu item displays the Open Project dialog box, where you can open an existing project, a new project, a new workspace, or a project wizard. This is the same dialog box displayed via File -> Open Project (see Open Project Dialog Box on page 3-12).

**Build Workspace Dialog Box**

You use the Build Workspace popup menu item to build a project workspace. This menu item is only accessible when nothing is selected in the Project view. Choosing the item displays the Build Workspace dialog box, where you select the projects to include in the workspace. If no projects exist yet, click OK to build an empty workspace.

![Build Workspace dialog box](image)

Figure 3-52: Build Workspace dialog box
Project View Options Dialog Box

You use the Project View Options popup menu item to define how projects are organized in the Project view. This menu item is only accessible when nothing is selected in the Project view.

Choosing this menu item displays the Project View Options dialog box, where you define how projects are organized in the Project view. This is the same dialog box displayed via Options -> Project View Options (see Project View Options Dialog Box on page 3-55).

Save Project As Dialog Box

You use the Save Workspace popup menu item to save a workspace. This menu item is only accessible when a workspace is selected in the Project view. Choosing this menu item displays the Save Project As dialog box, where you specify the workspace’s filename and location.

This is the same as the Save As dialog box (see Save As Dialog Box on page 3-10). The Save as type zone is predefined as Project Files (Project) (*.prj).

SCOPE Popup Menu

The popup menu in the SCOPE window contains commonly used commands from the menubar Edit menu (see Edit Menu Commands in the SCOPE Window on page 3-18): Undo, Cut/Disable, Copy, Paste, Clear/Enable, Remove Rows, Insert Row, Fill Down, Find, and Insert Wizard. The Cut/Disable and Clear/Enable commands act just like the Edit menu Cut and Clear commands, respectively, except when they are used in the SCOPE window Enabled column. Then, they simply turn the Enabled check box on and off.

Text Editor Popup Menu

The popup menu in the Text Editor window contains commonly used text-editing commands from the menubar Edit menu (see Edit Menu Commands in the Text Editor View on page 3-17): Undo, Redo, Cut, Copy, Paste, and Toggle Bookmark.
RTL View and Technology View Popup Menus

The popup menus in the RTL and Technology views are the same. The commands are the same as commands in either the View menubar menu (see View Menu Commands on page 3-22) or the HDL Analyst menubar menu (see Hdl Analyst Menu Commands on page 3-46), with the following exceptions, described below:

- Dissolve Selected Instances
- View FSM Info File
- the SCOPE submenu: SCOPE -> Edit Attributes

Commands Shared with the View Menu

The following commands are shared with the View menubar menu:

Push/Pop Hierarchy, Previous Sheet, Next Sheet, View Sheets, Select All, Unselect All, Zoom In, Zoom Out, Pan, Pan Center, Full View, Normal View, Zoom Lock. See View Menu Commands in RTL/Technology Views on page 3-23.

Commands Shared with the HDL Analyst Menu

The following commands are shared with the HDL Analyst menubar menu:


Commands Occurring Only in the Popup Menus

The following commands occur only in the RTL view and Technology view popup menus:

Dissolve Selected Instances, View FSM Info File, the SCOPE submenu commands.

Dissolve Selected Instances

The Dissolve Selected Instances command replaces instances that you have selected by their equivalents at the next lower level of the hierarchy. In effect, each selected instance is replaced, in the current view (RTL or...
Chapter 3: Menus, Dialog Boxes . . .

Menu, Dialog Boxes . . .

Technology), by what you would see if you were to push into it using the Push/Pop Hierarchy down arrow pointer. The rest of the view, which was not selected, remains unchanged.

View FSM Info File
To access the View FSM Info File command, you must first select a finite state machine module in the RTL or Technology view.

- View FSM Info File – Displays information about the selected finite state machine, including the number of states and inputs and a table of the states and transitions.

SCOPE Submenu
The Edit Attributes command of the popup menu’s SCOPE submenu displays the SCOPE Attributes dialog box, which you use to edit the attributes of an instance selected in the RTL or Technology view.

Table 3-37: SCOPE Attributes dialog box features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Enables the Attribute on the same line.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Clicking the Attribute column displays a pulldown list of available attributes. You choose an attribute, then define its Value. The attribute’s type and description are indicated at the bottom of the dialog box.</td>
</tr>
<tr>
<td>Value</td>
<td>Clicking a cell in the Value column displays its attribute’s default value. You can use the pulldown list to assign one of the available values.</td>
</tr>
</tbody>
</table>
Table 3-37: SCOPE Attributes dialog box features (Continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>You can enter a comment here.</td>
</tr>
<tr>
<td>OK</td>
<td>Updates the attribute information in the constraint file, and closes the dialog box.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancels any changes you may have made, and closes the dialog box.</td>
</tr>
<tr>
<td>Add Row</td>
<td>Adds a new, empty row above the current row in the table.</td>
</tr>
<tr>
<td>Remove Row</td>
<td>Removes the selected row from the table.</td>
</tr>
<tr>
<td>Undo</td>
<td>Undoes the last action.</td>
</tr>
<tr>
<td>Redo</td>
<td>Performs the action undone by Undo.</td>
</tr>
</tbody>
</table>
Chapter 3: *Menus, Dialog Boxes*. . .  Toolbars

Toolbars

Toolbars provide a quick way to access common menu commands. They have icons (iconic action buttons) grouped by related function.

You can toggle the display of individual toolbars, move them around, and dock them along the top or side of the application (main) window. You can also change the size (large or small) of the toolbar icons. See *Customize Dialog Box on page 3-26* for details.

The following toolbars are available.

- **Project Toolbar** — contains icons for project control and file manipulation.
- **Analyst Toolbar** — contains icons for manipulating RTL and Technology views.
- **View Toolbar** — controls viewing and hierarchy navigation in different views.
- **Edit Toolbar** — contains text editing and bookmarking icons, for the Text Editor view.

Project Toolbar

The Project toolbar provides the following icons, by default:

![Project Toolbar Image](image)

Figure 3-54: Project toolbar
The following table describes the Project toolbar icons.

Table 3-38: Project toolbar icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open project</td>
<td>Displays the Open Project dialog box to create a new project or open an existing project. A project wizard can help you specify and arrange project files.</td>
</tr>
<tr>
<td>New HDL file</td>
<td>Opens the Text Editor window with a new, empty source file.</td>
</tr>
<tr>
<td>New constraint file (SCOPE)</td>
<td>Opens the SCOPE spreadsheet with a new, empty constraint file.</td>
</tr>
<tr>
<td>Open</td>
<td>Displays the Open dialog box, to open a file.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the current file. If the file has not yet been saved, this displays the Save As dialog box, where you to specify the filename. The kind of file depends on the active view.</td>
</tr>
<tr>
<td>Save all</td>
<td>Saves all files associated with the current design.</td>
</tr>
<tr>
<td>Print</td>
<td>Prints the active view (RTL, Technology or Text Editor).</td>
</tr>
<tr>
<td>Cut</td>
<td>Cuts text or graphics from the active view.</td>
</tr>
<tr>
<td>Paste</td>
<td>Pastes previously cut or copied text or graphics to the active view.</td>
</tr>
<tr>
<td>Undo</td>
<td>Undoes the last action taken.</td>
</tr>
<tr>
<td>Redo</td>
<td>Performs the action undone by Undo.</td>
</tr>
<tr>
<td>Find</td>
<td>Finds text in the Text Editor or objects in an RTL or technology view.</td>
</tr>
</tbody>
</table>
Analyst Toolbar

The Analyst toolbar becomes active once a design has been compiled.

![Analyst toolbar icons]

Figure 3-55: Analyst toolbar

The following table describes the Analyst toolbar icons.

Table 3-39: Analyst toolbar icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL View</td>
<td>Opens an RTL Schematic view of the compiled design, with associated Hierarchy Browser.</td>
</tr>
<tr>
<td>Technology View</td>
<td>Opens a Technology Schematic view of the mapped (synthesized) design, with associated Hierarchy Browser.</td>
</tr>
<tr>
<td>Show Critical Path</td>
<td>Highlights the critical path in an active Technology view.</td>
</tr>
<tr>
<td>Filter on Selected Gates</td>
<td>Redisplays the Technology and RTL views, showing only the selected gates. Click the toolbar icon again to redisplay the view with no filtering.</td>
</tr>
</tbody>
</table>
Toolbars

Chapter 3: Menus, Dialog Boxes . . .

View Toolbar

The View toolbar allows you to zoom in and out of your schematic, push up and down to traverse your hierarchy, and view your design sheet by sheet.

![View toolbar icons](image)

Figure 3-56: View toolbar

The following table describes the View toolbar icons.

Table 3-40: View toolbar icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom 100%</td>
<td>Zooms the active view, when you click, to full (normal) size. Same as View -&gt; Normal View (see View Menu Commands on page 3-22).&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>Zoom In</td>
<td>Toggles zooming in. Same as View -&gt; Zoom In (see View Menu Commands on page 3-22).&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>Toggles zooming out. Same as View -&gt; Zoom Out (see View Menu Commands on page 3-22).&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>Zoom Full</td>
<td>Zooms the active view so that it shows the entire design. Same as View -&gt; Full View.&lt;sup&gt;b&lt;/sup&gt;</td>
</tr>
<tr>
<td>Push/Pop Hierarchy</td>
<td>Toggles traversing the hierarchy. Same as View -&gt; Push/Pop Hierarchy (see View Menu Commands on page 3-22).&lt;sup&gt;b&lt;/sup&gt;</td>
</tr>
<tr>
<td>Previous Sheet</td>
<td>Displays the previous sheet of a multiple-sheet schematic.</td>
</tr>
<tr>
<td>Next Sheet</td>
<td>Displays the next sheet of a multiple-sheet schematic.</td>
</tr>
</tbody>
</table>

<sup>a</sup> Available only in SCOPE spreadsheet, RTL and Technology views.

<sup>b</sup> Available only in RTL and Technology views.
Chapter 3: Menus, Dialog Boxes . . .

Toolbars

Edit Toolbar

The Edit toolbar is active whenever the Text Editor is. You use it to edit bookmarks in the file.

![Edit Toolbar Icons]

Figure 3-57: Edit toolbar

The following table describes the Edit toolbar icons.

Table 3-41: Edit toolbar icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle Bookmark</td>
<td>Alternately inserts and removes a bookmark to the line containing the text cursor. Same as Edit -&gt; Toggle bookmark (see Edit Menu Commands on page 3-17).</td>
</tr>
<tr>
<td>Next Bookmark</td>
<td>Takes you to the next bookmark. Same as Edit -&gt; Next bookmark (see Edit Menu Commands on page 3-17).</td>
</tr>
<tr>
<td>Previous Bookmark</td>
<td>Takes you to the previous bookmark. Same as Edit -&gt; Previous bookmark (see Edit Menu Commands on page 3-17).</td>
</tr>
<tr>
<td>Clear All Bookmarks</td>
<td>Removes all bookmarks from the Text Editor window. Same as Edit -&gt; Delete all bookmarks (see Edit Menu Commands on page 3-17).</td>
</tr>
</tbody>
</table>
Keyboard Shortcuts

Keyboard equivalents, or shortcuts, are key sequences that you type in order to run a command. Menus list keyboard equivalents next to the corresponding commands. For example, to check syntax, you can press and hold the Shift key while you type the F7 key, instead of picking the menu item Run -> Syntax Check.

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-1</td>
<td>Zooms the active view, when you click, to full (normal) size. Same as View -&gt; Normal View (see View Menu Commands on page 3-22). Available only in RTL and Technology views.</td>
</tr>
<tr>
<td>Ctrl-F2</td>
<td>Alternately inserts and removes a bookmark to the line containing the text cursor. Same as Edit -&gt; Toggle bookmark (see Edit Menu Commands on page 3-17).</td>
</tr>
<tr>
<td>Ctrl-F4</td>
<td>Closes the current window. Same as File -&gt; Close (see File Menu Commands on page 3-5).</td>
</tr>
<tr>
<td>Ctrl-F6</td>
<td>Activates the next window.</td>
</tr>
<tr>
<td>Ctrl-a</td>
<td>Centers the window on the design. Same as View -&gt; Pan Center (see View Menu Commands on page 3-22).</td>
</tr>
<tr>
<td>Ctrl-c</td>
<td>Copies the selected object. Same as Edit -&gt; Copy (see Edit Menu Commands on page 3-17).</td>
</tr>
</tbody>
</table>
Table 3-42: Keyboard shortcuts (Continued)

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-f</td>
<td>Finds the selected object. Same as Edit -&gt; Find in the Text Editor (see <em>Edit Menu Commands</em> on page 3-17). Same as HDL Analyst -&gt; Find in an RTL or Technology view (see <em>Object Query Dialog Box</em> on page 3-49).</td>
</tr>
<tr>
<td>Ctrl-g</td>
<td>Jumps to the specified line in the Text Editor. Same as Edit -&gt; GoTo (see <em>Edit Menu Commands</em> on page 3-17).</td>
</tr>
<tr>
<td>Ctrl-h</td>
<td>In the Text Editor, replaces text: same as Edit -&gt; Replace (see <em>Edit Menu Commands</em> on page 3-17). In an RTL or Technology view, enables/disables crossprobing to Visual Elite: same as HDL Analyst -&gt; Visual Elite Crossprobing -&gt; Connect to Visual Elite / Disconnect from Visual Elite (see <em>Hdl Analyst Menu Commands</em> on page 3-46).</td>
</tr>
<tr>
<td>Ctrl-k</td>
<td>Enables crossprobing to windows in other tools. Same as HDL Analyst -&gt; External Crossprobing Engaged (see <em>Hdl Analyst Menu Commands</em> on page 3-46).</td>
</tr>
<tr>
<td>Ctrl-l</td>
<td>When enabled, if you resize the window the displayed schematic is resized proportionately, so that it occupies the same portion of the window. Same as View -&gt; Zoom Lock. Available only in RTL and Technology views.</td>
</tr>
<tr>
<td>Ctrl-n</td>
<td>Opens a new file or project. Same as File -&gt; New (see <em>File Menu Commands</em> on page 3-5).</td>
</tr>
<tr>
<td>Ctrl-o</td>
<td>Opens an existing file or project. Same as File -&gt; Open (see <em>File Menu Commands</em> on page 3-5).</td>
</tr>
<tr>
<td>Ctrl-p</td>
<td>Prints the current view. Same as File -&gt; Print (see <em>File Menu Commands</em> on page 3-5).</td>
</tr>
<tr>
<td>Ctrl-r</td>
<td>Displays a net driver for a selected net. Same as View -&gt; Goto Net Driver (see <em>View Menu Commands</em> on page 3-22).</td>
</tr>
<tr>
<td>Ctrl-s</td>
<td>In the Project View, saves the file: same as File -&gt; Save (see <em>File Menu Commands</em> on page 3-5). In an RTL or Technology view, selects the sheet number in a multipage schematic: same as View -&gt; View Sheets (see <em>View Menu Commands</em> on page 3-22).</td>
</tr>
<tr>
<td>Ctrl-v</td>
<td>Pastes the last object copied or cut. Same as Edit -&gt; Paste (see <em>Edit Menu Commands</em> on page 3-17).</td>
</tr>
<tr>
<td>Ctrl-x</td>
<td>Cuts the selected object or objects. Same as Edit -&gt; Cut (see <em>Edit Menu Commands</em> on page 3-17).</td>
</tr>
<tr>
<td>Keyboard Shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Ctrl-y</td>
<td>Performs the action undone by Undo. Same as Edit -&gt; Redo (see <em>Edit Menu Commands</em> on page 3-17).</td>
</tr>
<tr>
<td>Ctrl-z</td>
<td>Undoes the last action. Same as Edit -&gt; Undo (see <em>Edit Menu Commands</em> on page 3-17).</td>
</tr>
<tr>
<td>Ctrl-Shift-F2</td>
<td>Removes all bookmarks from the Text Editor window. Same as Edit -&gt; Delete all bookmarks (see <em>Edit Menu Commands</em> on page 3-17).</td>
</tr>
<tr>
<td>Ctrl-Shift-i</td>
<td>Selects all instances in an RTL or Technology view. Same as View -&gt; Select all -&gt; Instances (see <em>View Menu Commands in RTL/Technology Views</em> on page 3-23).</td>
</tr>
<tr>
<td>Ctrl-Shift-n</td>
<td>Selects all nets in an RTL or Technology view. Same as View -&gt; Select all -&gt; Nets (see <em>View Menu Commands in RTL/Technology Views</em> on page 3-23).</td>
</tr>
<tr>
<td>Ctrl-Shift-p</td>
<td>Selects all ports in an RTL or Technology view. Same as View -&gt; Select all -&gt; Ports (see <em>View Menu Commands in RTL/Technology Views</em> on page 3-23).</td>
</tr>
<tr>
<td>d</td>
<td>Selects the driver for the selected net. Same as View -&gt; Select Net Driver in an RTL or Technology view (see <em>View Menu Commands in RTL/Technology Views</em> on page 3-23).</td>
</tr>
<tr>
<td>Delete</td>
<td>Removes the selected files from the project. Same as Project -&gt; Remove Files From Project (see <em>Project Menu Commands</em> on page 3-30).</td>
</tr>
<tr>
<td>e</td>
<td>Expands the path to include instances connected to the selected pin. Same as HDL Analyst -&gt; Expand in an RTL or Technology view (see <em>Hdl Analyst Menu Commands</em> on page 3-46).</td>
</tr>
<tr>
<td>E (Shift-e)</td>
<td>Expands the path in the selected direction, from the selected pin up to the next register or port. Same as HDL Analyst -&gt; Expand to Register/Port (see <em>Hdl Analyst Menu Commands</em> on page 3-46).</td>
</tr>
<tr>
<td>F1</td>
<td>Provides context-sensitive help. Same as Help -&gt; Help (see <em>Help Menu Commands</em> on page 3-65).</td>
</tr>
</tbody>
</table>
| F2               | In an RTL or Technology view, toggles traversing the hierarchy: same as View -> Push/Pop Hierarchy (see *View Menu Commands* on page 3-22).  
In the Text Editor, takes you to the next bookmark. |
### Table 3-42: Keyboard shortcuts (Continued)

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4</td>
<td>In the Project view, adds a file to the project: same as Project -&gt; Add Source File (see <em>Select Files to Add to Project Dialog Box on page 3-11</em>). In an RTL or Technology view, zooms the view so that it shows the entire design: same as View -&gt; Full View.</td>
</tr>
<tr>
<td>F5</td>
<td>Displays the next source file error. Same as Run -&gt; Next Error (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
<tr>
<td>F7</td>
<td>Compiles your design, without mapping it. Same as Run -&gt; Compile Only (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
<tr>
<td>F8</td>
<td>Synthesizes (compiles and maps) your design. Same as Run -&gt; Synthesize (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
<tr>
<td>F10</td>
<td>In an RTL or Technology view, pans the view in the direction you drag the pointer. Same as View -&gt; Pan (see <em>View Menu Commands on page 3-22</em>).</td>
</tr>
<tr>
<td>F11</td>
<td>Toggles zooming in. Same as View -&gt; Zoom In (see <em>View Menu Commands on page 3-22</em>).</td>
</tr>
<tr>
<td>F12</td>
<td>Filters the schematic to show only the selected objects and their connections. Same as HDL Analyst -&gt; Filter Schematic (see <em>Hdl Analyst Menu Commands on page 3-46</em>).</td>
</tr>
<tr>
<td>i</td>
<td>Selects instances connected to the selected net. Same as View -&gt; Select Net Instances in an RTL or Technology view (see <em>View Menu Commands on page 3-22</em>).</td>
</tr>
<tr>
<td>p</td>
<td>Shows all the logic between the selected instances. Same as HDL Analyst -&gt; Expand Paths (see <em>Hdl Analyst Menu Commands on page 3-46</em>).</td>
</tr>
<tr>
<td>Shift-F2</td>
<td>In the Text Editor, takes you to the previous bookmark.</td>
</tr>
<tr>
<td>Shift-F5</td>
<td>Displays the previous source file error. Same as Run -&gt; Previous Error (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
<tr>
<td>Shift-F7</td>
<td>Checks source file syntax. Same as Run -&gt; Syntax Check (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
<tr>
<td>Shift-F8</td>
<td>Checks synthesis. Same as Run -&gt; Synthesis Check (see <em>Run Menu Commands on page 3-43</em>).</td>
</tr>
</tbody>
</table>
Table 3-42: Keyboard shortcuts (Continued)

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift-F11</td>
<td>Toggles zooming out. Same as View -&gt; Zoom Out (see View Menu Commands on page 3-22).</td>
</tr>
<tr>
<td>Shift-Left Arrow</td>
<td>Displays the previous sheet of a multiple-sheet schematic.</td>
</tr>
<tr>
<td>Shift-Right Arrow</td>
<td>Displays the next sheet of a multiple-sheet schematic.</td>
</tr>
</tbody>
</table>

Action Buttons and Options

The Project view contains several action buttons and a few additional features that give you immediate access to some of the more common commands and user options.

![Figure 3-59: Project view action buttons]

The following table describes the Project View action buttons and options.

Table 3-43: Project view action buttons and options

<table>
<thead>
<tr>
<th>Button / Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (MHz)</td>
<td>You use this field to set the global frequency, which you can override locally with attributes. Same as setting the frequency with Project -&gt; Implementation Options (see <em>Project Menu Commands</em> on page 3-30).</td>
</tr>
<tr>
<td>Symbolic FSM Compiler</td>
<td>Turning this on enables special FSM optimizations. Same as enabling the FSM Compiler option with Project -&gt; Implementation Options (see <em>Project Menu Commands</em> on page 3-30).</td>
</tr>
<tr>
<td>Resource Sharing</td>
<td>When enabled, makes the synthesis use resource sharing techniques. Same as enabling the Resource Sharing option on the Options/Constraints panel of the Options for implementation dialog box (see <em>Options/Constraints Panel</em> on page 3-33). This produces the resource sharing report in the log file (see <em>Resource Usage Report</em> on page 1-29).</td>
</tr>
<tr>
<td>Run</td>
<td>Runs synthesis (compilation and mapping). Same as the Run -&gt; Synthesize command (see <em>Run Menu Commands</em> on page 3-43).</td>
</tr>
</tbody>
</table>
CHAPTER 4

Tcl Commands and Scripts

The following topics contain information about Tcl commands and scripts:

- Introduction to Tcl on page 4-2
- Batch Mode on page 4-6
- Tcl Commands for Synthesis on page 4-9
- Tcl Script Examples on page 4-21
Introduction to Tcl

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

Tcl scripts are text files that have a “.tcl” file extension (.tcl) and contain a set of Tcl commands designed to complete a task on set of tasks.

The Synplicity Tcl commands are described in this chapter. For information on the standard Tcl commands, syntax, language and conventions, refer to the Tcl online help provided in your installation directory (Help -> TCL Help).

Tcl Conventions

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

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

  designs/big_design/test.v
Creating Tcl Scripts

This section provides information on creating Tcl scripts. However, details on the commands used are not provided here. For information on command syntax and usage, see the individual command sections of this chapter. To create a Tcl script:

1. Open a text file for the script and specify a new project using the `project -new` command.

2. Add HDL source files to the script for your project by using the `add_file -verilog` or `add_file -vhdl` command.

3. Use synthesis control Tcl commands to set the target technology, design speed goal, enable the symbolic FSM compiler and to set other options with the `set_option` command.

4. Use vendor-specific Tcl commands to set your target technology, part, package and speed grade, and to change the default mapping options for your target technology such as fan-out limit. Refer to the section for your vendor for more information.

5. (Optional): Load a constraint file using the `add_file -constraint` command to set timing constraints and vendor-specific attributes.

6. Specify synthesis of your Project using the `project -run` command. If you want, you can leave out the `project -run` command and instead, click the Run button in the Project window after running your Tcl script.

7. Save the script using a “tcl” file extension (.tcl).
Starting in Batch Mode Using a Project File

Batch mode is only available if you have a floating license. You can run batch mode from a project file or a Tcl script; the former is easier. For information about using a Tcl script for batch mode, see *Starting in Batch Mode Using a Tcl Script* on page 4-7.

1. Create a project file either from the Project window, or by typing the appropriate Tcl commands into a `.prj` project file with an ASCII text editor. A project file is a Tcl file that contains Tcl synthesis commands.
   - Set implementation options (Project -> Implementation Options or the `set_option` Tcl command). See *Tcl Commands and Scripts* on page 4-1, for information about the Tcl commands you can include.
   - Add project files such as HDL source files, constraint files, and Tcl scripts (File -> Build Project or the `add_file` Tcl command).
   - Save the project file (File -> Save). If you are typing the commands into a text file, make sure the file has a “prj” extension (.prj).

2. From the command prompt, go to the directory where the project file is located.

3. Type the command that initiates batch mode for the tool you are using at a UNIX or DOS command line. For example:

   $$ \text{\textasciitilde syplify -batch project_name.prj} $$

   The Synplify synthesis tool starts in batch mode, and synthesizes the design using the options you set in the `.prj` project file.

4. Check the results in the `project_name.srr` file.

   If there are errors, see the stdout (usually the monitor) in UNIX or the stdout.log file in Microsoft Windows.
Executing Tcl Scripts

To execute a Tcl script:

1. Choose Run -> Run Tcl Script.
2. Choose your Tcl script file from the list.
3. Click Open.

Including Command Files in a Project

You can include a Tcl command file in the source files list of a project. To do this, right-click in the Project view, then choose Add Source File in the popup menu. From the source file list, select the Tcl command file you want to include.
Chapter 4: Tcl Commands and Scripts

Batch Mode

Starting in Batch Mode Using a Project File

Batch mode is only available if you have a floating license. You can run batch mode from a project file or a Tcl script; the former is easier. For information about using a Tcl script for batch mode, see Starting in Batch Mode Using a Tcl Script on page 4-7.

1. Create a project file, either from the Project window in the GUI, or by typing the appropriate Tcl commands into a .prj project file using an ASCII text editor. A project file is a Tcl file that contains Tcl synthesis commands.
   - Set implementation options as needed (Project -> Implementation Options or the set_option Tcl command). See Tcl Commands for Synthesis on page 4-9 for information about the Tcl commands you can include.
   - Add project files such as HDL source files, constraint files, and Tcl scripts (File -> Build Project, the Add File button in the Project view, or the add_file Tcl command).
   - Save the project file (File -> Save). If you are typing the commands into a text file, make sure the file has a “prj” extension (.prj).

2. From the command prompt, go to the directory where the project file is located.

3. Type the following command, which initiates batch mode, at a UNIX or DOS command line:

   ```bash
   % synplify -batch project_name.prj
   ```

   The Synplify synthesis tool starts in batch mode, and synthesizes the design using the options you set in the .prj project file.

4. Check the results in the project_name.srr file.

   If there are errors, see stdout (usually the monitor) in UNIX, or file stdout.log in Microsoft Windows.
Starting in Batch Mode Using a Tcl Script

Batch mode is only available if you have a floating license. You can run batch mode from a project file or a Tcl script. Running through a project file is generally easier (see Starting in Batch Mode Using a Project File on page 4-4 for information). Tcl (Tool command language) is a popular, easy-to-use scripting language for controlling software applications. A project file is also in the Tcl format, but it has a “prj” extension (.prj). A Tcl synthesis script has a “tcl” extension (.tcl).

You can use a Tcl script to do a single synthesis run, or for different runs using different technologies, timing goals and constraints. Advanced users, in particular, often use a single script to run multiple synthesis implementations with different options. You can run a Tcl script in batch mode as described here, from the Project view using Run->Run Tcl Script.

To run a Tcl script in batch mode:

1. Using an ASCII text editor, enter the beginning and ending Tcl commands.
   
   ```
   project -new
   #All other Tcl commands go here
   project -run
   exit
   ```

2. Enter the Tcl commands. For a complete list of Tcl commands, see Tcl Commands for Synthesis on page 4-9.

3. Save the file using a .tcl file extension in the same directory as the other project files, HDL source files, constraint files, and possibly Tcl scripts.

4. At the command prompt in a command or Tcl window, navigate to the directory where the Tcl files are located.

5. At the command prompt, type this command to initiate the batch process:
   
   ```
   synplify -batch Tcl_script_name.tcl
   ```

   This command runs the Synplify synthesis tool in batch mode, without opening the Project window. It synthesizes the design using the commands in the Tcl file.
6. Check the results in stdout and the file project_name.srr.

The synthesis tool reports process status via return codes, stdout and .srr files. For UNIX systems, stdout is usually directed to the monitor. For Microsoft Windows, file stdout.log is written.

Synthesis status (compilation and mapping) and errors are written to the .srr log file for each project. The synthesis tool also reports success and failure return codes: 0 for successful process completion; 1 for process failure or error conditions. Warnings return a successful process return code of 0. If the Tcl script incurs an error, check the stdout.
Tcl Commands for Synthesis

You use the following Tcl commands to create and synthesize projects, add and open files, and control synthesis. The commands are listed in alphabetical order:

- add_file on page 4-9
- constraint_file on page 4-10
- get_env on page 4-11
- get_option on page 4-11
- project on page 4-12
- project_file on page 4-13
- set_option on page 4-15
- Vendor-Specific Tcl Commands on page 4-19

add_file

Adds files to a project.

Syntax

```
add_file [-language] [-lib libraryName] [-include] fileName
```

where language is one of the following:

<table>
<thead>
<tr>
<th>language</th>
<th>file extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>-verilog</td>
<td>.v</td>
</tr>
<tr>
<td>-vhdl</td>
<td>.vhd, .vhdl</td>
</tr>
<tr>
<td>-constraint</td>
<td>.sdc</td>
</tr>
</tbody>
</table>

Depending on the language specified, the remaining command line arguments may or may not be available. The command line syntax for adding Verilog and VHDL files is as follows:

```
add_file -verilog fileName.v
```
add_file -vhdl [-lib libraryName] fileName.vhd

Examples

Read in the Verilog and constraint files for the design called top.

% add_file -verilog controller.v
% add_file -verilog interface.v
% add_file -verilog memory.v
% add_file -verilog top.v
% add_file -constraint top.sdc

Read in the VHDL IP files for the design called cpu.

% add_file -vhdl -lib work hard_IP1.vhd
% add_file -vhdl -lib work hard_IP2.vhd

constraint_file

Manipulates the constraint files used by the active implementation.

Syntax

constraint_file {-enable constraintFileName | -disable constraintFileName | -list | -all | -clear }

The following table describes the command options.

Table 4-1: constraint_file Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-enable</td>
<td>Selects the specified constraint file to be used by the active implementation.</td>
</tr>
<tr>
<td>-disable</td>
<td>Excludes the specified constraint file from being used for the active implementation</td>
</tr>
<tr>
<td>-list</td>
<td>Lists the constraint files used by the active implementation</td>
</tr>
<tr>
<td>-all</td>
<td>Selects (includes) all the project constraint files for the active implementation.</td>
</tr>
<tr>
<td>-clear</td>
<td>Clears (excludes) all the constraint files for the active implementation</td>
</tr>
</tbody>
</table>
Examples

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

```
% constraint_file -list
attributes.sdc clocks1.sdc clocks2.sdc eight_bit_uc.sdc

% constraint_file -disable eight_bit_uc.sdc
```

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

```
% constraint_file -clear

% constraint_file -enable clocks2.sdc
```

get_env

Reports the value of a predefined system variable.

Syntax

```
get_env system_variable
```

Once you use this command in a Tcl script, you can subsequently specify the value of the variable using the `$system_variable` syntax. This can be useful for specifying paths in your scripts. For example, if you have defined the path to your project file in a `MY_PROJECT` variable, you could include the following in a Tcl script:

```
add_file $MY_PROJECT
```

get_option

Reports the settings of predefined project and device options. The options are the same as those for `set_option`. See `set_option` on page 4-15 for details.

Syntax

```
get_option -optionName
```
Chapter 4: Tcl Commands and Scripts

Tcl Commands for Synthesis

---

project

Runs job flows, to create, load, save, and close projects, and to change and examine project status.

Syntax

```
project { -run mode | -new [projectPath] | -load projectPath | -close | -save | -result_file [resultFilePath] | -result_format [resultFormat] | -log_file [logfileName] | -active [projectName] | -dir | -file | -name | -list | -filelist }
```

The following table describes the command options.

**Table 4-2: project command options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -run mode          | Performs the indicated action (mode) on the currently active project, where mode is one of the following keywords:  
  • compile – Compiles (only) the currently active project.  
  • synthesis – Compiles (if necessary) and synthesizes the currently active project.  
  • synthesis_check – Verifies that the design is functionally correct; errors are reported in the log file.  
  • syntax_check – Verifies that the HDL is syntactically correct; errors are reported in the log file. |
| -new               | Creates a new workspace/project in the current working directory. If projectPath is specified, creates the project in the specified project directory. The workspace project name defaults to Workspace. |
| -load              | Loads the project file specified by projectPath. |
| -close             | Closes the currently active project. |
| -save              | Saves the currently active project. |
| -result_format     | Sets the synthesis result file format. If resultFormat is specified, changes the result file type to the format specified. The format varies with the vendor and the technology used. |
| -log_file          | Reports the name of the project log file. If logfileName is specified, changes the base name of the log file. |
Examples

Load the project top.prj and compile the design only, for example, so you can create a .sdc file in the SCOPE spreadsheet or see an RTL schematic representation of the design.

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

Load a project, specify a log filename other than the default (top.srr), and synthesize the design.

% project -load top.prj
% project -log_file top_Feb14.srr
% project -run synthesis

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

project_file

Used to manipulate and examine project files.

Table 4-2: project command options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-active</td>
<td>Shows the active project. If <strong>projectName</strong> is specified, makes the specified project the active project.</td>
</tr>
<tr>
<td>-result_file</td>
<td>Names the synthesis result file. If <strong>resultFilePath</strong> is specified, changes the name/location of the result file to the path specified.</td>
</tr>
<tr>
<td>-dir</td>
<td>Shows the project directory for the active project.</td>
</tr>
<tr>
<td>-file</td>
<td>Shows the active project filename.</td>
</tr>
<tr>
<td>-name</td>
<td>Returns the active project name.</td>
</tr>
<tr>
<td>-list</td>
<td>Returns a list of the loaded projects.</td>
</tr>
<tr>
<td>-insert</td>
<td>Adds selected projects to the workspace project.</td>
</tr>
</tbody>
</table>
Chapter 4: Tcl Commands and Scripts

Tcl Commands for Synthesis

Syntax

```
project_file {-lib fileName [libName] | -name fileName [newPath] |
-time fileName [format] | -date fileName |
-type fileName | -move fileName1 [fileName2] |
-remove fileName }
```

The following table describes the command options.

Table 4-3: project_file command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-lib</strong></td>
<td>Shows the project file library associated with <code>fileName</code>. If <code>libName</code> is specified, changes the project file library for the specified file to <code>libName</code>.</td>
</tr>
<tr>
<td><strong>-name</strong></td>
<td>Shows the project file path for the specified file. If <code>newPath</code> is specified, changes the location of the specified project file to the directory path specified by <code>newPath</code>.</td>
</tr>
<tr>
<td><strong>-time</strong></td>
<td>Shows the file time stamp. If a <code>format</code> is specified, changes the composition of the time stamp according to the combination of the following time formatting codes: %H (hour 00-23) %M (minute 00-59) %S (second 00-59) %d (day 01-31) %b (abbreviated month) %Y (year with century).</td>
</tr>
<tr>
<td><strong>-date</strong></td>
<td>Shows the file date.</td>
</tr>
<tr>
<td><strong>-type</strong></td>
<td>Shows the file type.</td>
</tr>
<tr>
<td><strong>-move</strong></td>
<td>Positions <code>fileName1</code> after <code>fileName2</code> in HDL file list. If <code>fileName2</code> is not specified, moves <code>fileName1</code> to the top of the list.</td>
</tr>
<tr>
<td><strong>-remove</strong></td>
<td>Removes the specified file from the project file list.</td>
</tr>
</tbody>
</table>

Examples

List the files added to a project and remove a file.

```
% project -filelist
path_name1/cpu.v path_name1/cpu_cntrl.v path_name2/cpu_cntrl.vhd
% project_file -remove path_name2/cpu_cntrl.vhd
```
**set_option**

Sets the settings of predefined project and device options.

**Syntax**

```
set_option -optionName optionValue
```

The following table lists the generic arguments for setting the device technology, part and speed grade. These options are equivalent to the options available on the Device panel of the Project -> Implementation Options dialog box.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Sets the target technology for the implementation. <em>Keyword</em> is the vendor architecture. Refer to the appropriate vendor chapter or check the Device panel of the Options for implementation dialog box (see Device Panel on page 3-32) for a list of supported families.</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Check the Device panel of the Options for implementation dialog box (see Device Panel on page 3-32) for available choices.</td>
</tr>
<tr>
<td>-speed_grade value</td>
<td>Sets the speed grade for the implementation. Check the Device panel of the Options for implementation dialog box (see Device Panel on page 3-32) for available choices.</td>
</tr>
<tr>
<td>-package value</td>
<td>Sets the package for the implementation. This option is not available for certain vendor families, because it is set in the place-and-route software. Check the Device panel of the Options for implementation dialog box (see Device Panel on page 3-32) for available choices.</td>
</tr>
<tr>
<td>-grade value</td>
<td>Same as -speed_grade. Included for backwards compatibility.</td>
</tr>
</tbody>
</table>

The following table is an alphabetical list of other arguments to the set_option and get_option commands, with brief descriptions and UI equivalents from the Project -> Implementation Options dialog box. Some options are technology-specific, and others have technology-specific defaults or limitations. The table in *Vendor-Specific Tcl Commands on page 4-19* summarizes where to go for vendor-specific details.
### Table 4-5: set_option and get_option Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>GUI Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-area_delay percent_value</code></td>
<td>Sets the percentage of paths you want optimized. This option is available only in certain device technologies.</td>
<td>Percent of design to optimize for timing dialog box zone, Device Panel. See also Area/Delay Trade-off for CPLDs on page 1-19.</td>
</tr>
<tr>
<td><code>-area_delay_percent percent_value</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`-autosm 1</td>
<td>0`</td>
<td>Enables/disables the FSM compiler.</td>
</tr>
<tr>
<td>`-symbolic_fsm_compiler 1</td>
<td>0`</td>
<td></td>
</tr>
<tr>
<td>`-block 1</td>
<td>0`</td>
<td>Enables/disables I/O insertion in some technologies. Same as <code>-disable_io_insertion</code>.</td>
</tr>
<tr>
<td>`-clique 1</td>
<td>0`</td>
<td>Enables/disables the grouping of logic functions in Altera technologies.</td>
</tr>
<tr>
<td>`-cliquing 1</td>
<td>0`</td>
<td></td>
</tr>
<tr>
<td>`-compiler_compatible 0</td>
<td>1`</td>
<td>Disables pushing of tristates across process/block boundaries.</td>
</tr>
<tr>
<td><code>-default_enum_encoding default onehot gray sequential</code></td>
<td>(VHDL only) Sets the default for enumerated types.</td>
<td>Default Enum Encoding, VHDL panel (see VHDL and Verilog Panels on page 3-36).</td>
</tr>
<tr>
<td>`-disable_io_insertion 1</td>
<td>0`</td>
<td>Enables/disables I/O insertion in some technologies. Same as <code>-block</code>.</td>
</tr>
<tr>
<td>`-domap 1</td>
<td>0`</td>
<td>Same as <code>-map_logic</code> for Xilinx macrocells. Supported for backwards compatibility.</td>
</tr>
</tbody>
</table>
### Tcl Commands for Synthesis

#### Chapter 4: Tcl Commands and Scripts

---

**Table 4-5: set_option and get_option Tcl command options (Continued)**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>GUI Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>`-force_gsr yes</td>
<td>no</td>
<td>auto`</td>
</tr>
<tr>
<td><code>-frequency value</code></td>
<td>Sets the global frequency.</td>
<td>Frequency, Options/Constraints Panel.</td>
</tr>
<tr>
<td><code>-include_path path</code></td>
<td>(Verilog only) Defines the search path used by the <code>include</code> commands in Verilog design files.</td>
<td>Include Path Order, Verilog panel (see VHDL and Verilog Panels).</td>
</tr>
<tr>
<td>`-map_logic 1</td>
<td>0`</td>
<td>Turns on direct mapping to technology-specific devices during synthesis. Applies to Altera (ATOMs and LCells), Lattice, and Xilinx (Macrocells) technologies.</td>
</tr>
<tr>
<td><code>-maxterms value</code></td>
<td>Sets the maximum number of terminals per macrocell. This option is available only if <code>-map_logic</code> is set to true. It applies to Lattice devices.</td>
<td>Maximum Terms/Macrocell, Device Panel.</td>
</tr>
<tr>
<td><code>-max_terms_per_macrocell value</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-maxfan value</code></td>
<td>Sets the fanout limit guideline for the current project. Applies to Actel, Lattice ORCA, QuickLogic, and Xilinx technologies.</td>
<td>Fanout Guide, Device Panel.</td>
</tr>
<tr>
<td><code>-fanout_guide value</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4-5: set_option and get_option Tcl command options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>GUI Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>-maxfan_hard 1</td>
<td>For Actel designs, it specifies that the -maxfan value is a hard fanout limit that the Synplify synthesis tool must not exceed.</td>
<td>Hard Limit to Fanout, Device Panel.</td>
</tr>
<tr>
<td>-maxfanin 1</td>
<td>0</td>
<td>-fanin_limit value</td>
</tr>
<tr>
<td>-no_time_gated_clock</td>
<td></td>
<td>Device Panel.</td>
</tr>
<tr>
<td>-num_startend_points value</td>
<td>Specifies the number of start and end points to include when reporting paths with the worst slack in the timing report.</td>
<td>Number of Start/End Points, Timing Report Panel.</td>
</tr>
<tr>
<td>-report_path value</td>
<td>Sets the maximum number of critical paths in a forward-annotated SDF constraint file (Actel 500K and PA designs only).</td>
<td>Max Number of Critical Paths in SDF, Device Panel.</td>
</tr>
<tr>
<td>-resource_sharing 1</td>
<td>0</td>
<td>Enables/disables resource sharing.</td>
</tr>
<tr>
<td>-result_file filename</td>
<td>Specifies the name of the results file.</td>
<td>Result File Name and Result Format, Implementation Results Panel.</td>
</tr>
<tr>
<td>-soft 1</td>
<td>0</td>
<td>-soft_buffers 1</td>
</tr>
<tr>
<td>-top_module name</td>
<td>Specifies the top-level module.</td>
<td>Top-level Entity/Module, VHDL and Verilog Panels.</td>
</tr>
</tbody>
</table>
### Table 4-5: set_option and get_option Tcl command options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>GUI Equivalenta</th>
</tr>
</thead>
<tbody>
<tr>
<td>-use_reset_pin 1</td>
<td>0</td>
<td>Use Dedicated Reset Pin, <strong>Device Panel.</strong></td>
</tr>
<tr>
<td>-userstpin 1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-use_scan_for_test</td>
<td></td>
<td>(tab)</td>
</tr>
<tr>
<td>-write_apr_constraint 1</td>
<td>0</td>
<td>Write Vendor Constraint File, <strong>Implementation Results Panel.</strong></td>
</tr>
<tr>
<td>-write_verilog 1</td>
<td>0</td>
<td>Write Mapped Verilog/VHDL Netlist, <strong>Implementation Results Panel.</strong></td>
</tr>
<tr>
<td>-write_vhdl 1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-xilinx_m1</td>
<td></td>
<td>(tab)</td>
</tr>
</tbody>
</table>

a. The panels referenced here belong to the Options for implementation dialog box.

### Vendor-Specific Tcl Commands

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

**Table 4-6: Vendor-specific Tcl commands**

<table>
<thead>
<tr>
<th>Vendor/Family</th>
<th>Tcl Commands Described in...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actel</td>
<td><strong>Actel-specific Tcl Command Options on page A-8</strong></td>
</tr>
<tr>
<td>Altera FLEX and ACEX</td>
<td><strong>Maximizing Results with FLEX and ACEX on page B-26</strong></td>
</tr>
<tr>
<td>Altera APEX, Excalibur and Mercury</td>
<td><strong>Maximizing Results with APEX, APEX II, Mercury, Excalibur on page B-14</strong></td>
</tr>
<tr>
<td>Altera MAX</td>
<td><strong>Maximizing Results with MAX on page B-32</strong></td>
</tr>
<tr>
<td>Atmel</td>
<td><strong>Atmel-specific Tcl Command Options on page C-4</strong></td>
</tr>
<tr>
<td>Cypress</td>
<td><strong>Cypress-specific Options on page D-11</strong></td>
</tr>
<tr>
<td>Lattice GAL/PAL and isp</td>
<td><strong>Lattice GAL and isp Devices on page E-11</strong></td>
</tr>
<tr>
<td>Lattice Mach</td>
<td><strong>MACH Devices on page E-12</strong></td>
</tr>
<tr>
<td>Lattice ORCA</td>
<td><strong>Lattice ORCA Devices on page E-4</strong></td>
</tr>
</tbody>
</table>
Table 4-6: Vendor-specific Tcl commands (Continued)

<table>
<thead>
<tr>
<th>Vendor/Family</th>
<th>Tcl Commands Described in...</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickLogic</td>
<td><em>QuickLogic-specific Tcl Command Options on page F-6</em></td>
</tr>
<tr>
<td>Triscend</td>
<td><em>Triscend-specific Tcl Command Options on page G-3</em></td>
</tr>
<tr>
<td>Xilinx Spartan</td>
<td><em>Using Spartan on page H-27</em></td>
</tr>
<tr>
<td>Xilinx Virtex</td>
<td><em>Using Virtex on page H-30</em></td>
</tr>
<tr>
<td>Xilinx XC4000</td>
<td><em>Using XC4000 on page H-34</em></td>
</tr>
<tr>
<td>Xilinx XC3000, XC5200, XC9500, CoolRunner, CoolRunner-II CPLDs</td>
<td><em>Using XC3000 and XC5200 on page H-37</em></td>
</tr>
</tbody>
</table>
Tcl Script Examples

This section provides examples of Tcl scripts.

- Using Several Target Technologies
- Different Clock Frequency Goals
- Setting Options and Timing Constraints
- Bottom-up Synthesis

Using Several Target Technologies

# Run synthesis multiple times without exiting while trying different
# target technologies. View their implementations in HDL Analyst.

# Open a new Project.
project -new

# Set the design speed goal to 33.3 MHz.
set_option -frequency 33.3

# Add a Verilog file to the source file list.
add_file -verilog "andorxor.v"

# Create a new Tcl variable, to be known as $try_these, that will be
# used to synthesize the design using different target technologies.

set try_these {
  ACT3
  APEX20K
  3200DX
  FLEX10K
  MAX9000
  pLSI1K
  ORCA2C
  pASIC1
  MACH
  XC4000E
  XC4000EX
  XC4000XL
  XC5200
  XC9500
  Virtex
  Virtexe}
Virtex2
}

# Loop through synthesis for each target technology.
foreach technology $try_these {

# Set the target technology from the $try_these list
  set_option -technology $technology

# Run synthesis.
  project -run

# Display the Technology View showing the implementation.
  open_file -technology_view
}

# Display one RTL View showing the RTL level schematic.
# The RTL level schematic is the same for all synthesis runs since the
# design has not changed.
  open_file -rtl_view

**Different Clock Frequency Goals**

# Run synthesis six times on the same design using different clock
# frequency goals. We want to see what the speed/area tradeoffs are for
# the different timing goals.

# Load an existing Project. This Project was created from an
# interactive session by saving the Project file after adding all the
# necessary files, and setting options in the Project->Implementation
# Options dialog box.

  project -load "design.prj"

# Create a Tcl variable, called $try_these, that will be used to
# synthesize the design with different frequencies.
  set try_these {
    20.0
    24.0
    28.0
    32.0
    36.0
    40.0
  }

# Loop through each frequency, trying each one
  foreach frequency $try_these {

# Set the frequency from the try_these list
set_option -frequency $frequency

# Since I want to keep all Log Files, save each one. Otherwise
# the default Log File name "<project_name>.srr" is used, which is
# overwritten on each run. Use the name "<$frequency>.srr" obtained from the
# $try_these Tcl variable.
project -log_file $frequency.srr

# Run synthesis.
project -run

# Display the Log File for each synthesis run
open_file -edit_file $frequency.srr
}

**Setting Options and Timing Constraints**

# Set a number of options and use timing constraints on the design.

# Open a new Project
project -new

# Set the target technology, part number, package, and speed grade options.
set_option -technology XC4000E
set_option -part XC4013E
set_option -package PC84
set_option -speed_grade -1

# Load the necessary VHDL files. Add the top-level design last.
add_file -vhdl "statemach.vhd"
add_file -vhdl "rotate.vhd"
add_file -vhdl "memory.vhd"
add_file -vhdl "top_level.vhd"

# Add a timing Constraint file and vendor-specific attributes.
add_file -constraint "design.sdc"

# The top level file ("top_level.vhd") has two different designs, of
# which the last is the default entity. Try the first entity (design1)
# for this run. In VHDL, you could also specify the top level architecture
# using <entity>.<arch>
set_option -top_module design1

# Turn on the Symbolic FSM Compiler to re-encode the state machine
# into one-hot.
set_option -symbolic_fsm_compiler true
# Set the design frequency.
set_option -frequency 30.0

# Save the existing Project to a file. The default synthesis Result File
# is named "<project name>.<ext>" so, if you wanted the synthesis Result
# File to be named something other than "design.xnf", you will change
# it with the project -result_file "<name>.xnf" command
project -save "design.prj"

# Synthesize the existing Project
project -run

# Open an RTL View
open_file -rtl_view

# Open a Technology View
open_file -technology_view

# This constraint file, "design.sdc," is read by "test3.tcl"
# with the add_file -constraint "design.sdc" command. Constraint files
# are for timing constraints and synthesis attributes.

# Timing Constraints:
# --------------------------------------------------
# The default design frequency goal is 30.0 MHz for four clocks. Except
# that clk_fast needs to run at 66.0 MHz. Override the 30.0 MHz default
# for clk_fast.
define_clock {clk_fast} -freq 66.0

# The inputs are delayed by 4 ns
define_input_delay -default 4.0

# except for the "sel" signal, which is delayed by 8 ns
define_input_delay {sel} 8.0

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

# From a previous run it was noticed in Technology View that the critical
# paths are the flip-flop to flip-flop paths going to register "inst3.q[0]"
# (in the memory). Improve the paths going to inst3.q[0] by 3.0 ns.
define_reg_input_delay {inst3.q[0]} -improve 3.0

# Xilinx-specific Attribute Constraint:
# --------------------------------------------------
# Assign a location for scalar port "sel". These VHDL object names are
# case sensitive.
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, not a clock buffer for clock "clk_slow". Save the
# clock buffers for the fast clocks
define_attribute {clk_slow} syn_noclockbuf 1

# Do not do regular buffering for "clk_slow", because its the
# slow clock and the timing can be relaxed on it.

# Set the maximum fanout to 10000.
define_attribute {clk_slow} syn_maxfan 10000

## Bottom-up Synthesis

# Bottom-up synthesis of a large design.

# The Source command reads in other Tcl scripts. Each of these scripts does
# a compile of one logic block and has its own constraint file.
source "statemach.tcl"
source "microproc.tcl"
source "handshake.tcl"
source "fifo.tcl"
source "cherstrp.tcl"

# After synthesizing the individual logic blocks, create a Project for the
# top-level design.
project -new

# Add the top level VHDL file.
add_file -vhdl top_level.vhd

# Add the top level global constraint file.
add_file -constraint top_level.sdc

# Set the top level options
set_option -technology FLEX10K
set_option -part EPF10K70
set_option -speed_grade -3
set_option -frequency 50.0
set_option -symbolic_fsm_compiler true
# Set the output file information
    project -result_file top_level.edf
    project -log_file top_level.srr
# Save the Project to file
    project -save top_level.prj
# Run the Project
    project -run
# Open the top level RTL and Technology Views
    open_file -rtl_view
    open_file -technology_view

# This file, "statemach.tcl," is read by "bottom_up.tcl," (the
# bottom up Tcl script) with the command "source statemach.tcl."
# The other .tcl scripts are similar.
# Open a new Project for "statemach"
    project -new
# Add the VHDL file for this logic block.
    add_file -vhdl statemach.vhd
# Add the constraint file for this logic block.
    add_file -constraint statemach.sdc
# Set the other options for "statemach".
    set_option -technology FLEX10K
    set_option -part EPF10K70
    set_option -speed_grade -3
    set_option -frequency 50.0
    set_option -symbolic_fsm_compiler true
# Set the Project outputs
    project -result_file statemach.edf
    project -log_file statemach.srr
# Save this Project
    project -save statemach.prj
# Run this Project
    project -run

# This file (statemach.sdc) is the constraint file read by
# "statemach.tcl," with the command add_file -constraint statemach.sdc.
# This constraint file is specific to this logic block " statemach "
# Timing Constraints:
define_input_delay -default -100
define_output_delay -default -100
define_input_delay RESET -10
define_reg_input_delay {q[8]} -improve 4.0

# Altera-Specific Attributes:
define_attribute {inst1.sqrt8} altera_implement_in_eab 1
CHAPTER 5
Using HDL Analyst

This chapter describes the HDL Analyst and its commands:

• About HDL Analyst on page 5-2
• Getting Started with HDL Analyst on page 5-4
• Traversing Hierarchy on page 5-12
• Multisheet Schematics in HDL Analyst on page 5-15
• Filtering Schematics in HDL Analyst on page 5-17
• Analyzing Critical Paths in HDL Analyst on page 5-18
About HDL Analyst

HDL Analyst is a graphical productivity tool that helps you visualize your synthesis results and analyze how you can improve the performance and area results of a device. HDL Analyst is an option to the Synplify synthesis tool.

HDL Analyst Views

HDL Analyst consists of two schematic views that graphically display the design information after compilation and mapping, respectively:

- The RTL-level schematic view
  
The RTL view displays your design as a high-level, technology-independent schematic. It consists of a Hierarchy Browser and a schematic, which represents a graphical view of the design after compilation. For a more detailed description of the view, see RTL View on page 2-4.

- The technology-primitive level schematic
  
The Technology view is a low-level, vendor-specific view of your design after technology mapping. Like the RTL view, it consists of a Hierarchy Browser and schematic, which shows vendor-specific components like look-up tables, cascade and carry chains, F and H Maps, muxes, and flip-flops. For a more detailed description of the view, see Technology View on page 2-5.

By graphically displaying the information, HDL Analyst makes it easy to analyze your design and debug it where necessary. It helps you visualize where coding changes or timing constraints might reduce the area or increase performance. You can crossprobe between the RTL and Technology views, the HDL source code, and the FSM viewer. HDL Analyst also lets you highlight and isolate parts of the design, like critical paths, so that you can analyze problem areas, add timing constraints, and resynthesize.
HDL Analyst Commands

The HDL Analyst menu choices become available after you run synthesis. You can also get to these commands from a popup menu that you access by right clicking in an RTL or Technology view. There are four places to access HDL Analyst commands:

- **HDL Analyst menubar menu commands**
  
  These are the commands on the HDL Analyst menubar menu. Not all the menu choices are available all the time; availability depends on which view is active and which objects are selected. The HDL Analyst menu commands are repeated in the popup menu in the RTL and Technology views. See *Hdl Analyst Menu Commands* on page 3-46, for more information about the commands.

- **HDL Analyst popup menu commands**
  
  Open a popup menu by right-clicking in an RTL or Technology view. The popup menu includes commonly used commands from the View and HDL Analyst menubar menus. Availability of particular commands depends on which view is active and which objects are selected. See *RTL View and Technology View Popup Menus* on page 3-85, for more information.

- **HDL Analyst toolbar icons**
  
  The HDL Analyst toolbar is the quickest way to access to open views and filter objects. See *Analyst Toolbar* on page 3-90, for more information.

- **View menubar menu Commands**
  
  When an RTL view or Technology view is the active window, the View menubar menu contains additional commands for graphical analysis. Some commands are duplicated in the popup menu. For a description of the additional View menu commands see *View Menu Commands* on page 3-22.
Chapter 5: Using HDL Analyst

Getting Started with HDL Analyst

This section contains general information to familiarize yourself with HDL Analyst and the RTL and Technology views. It discusses the following topics:

- Setting Up HDL Analyst for Synplify
- Viewing Object Information in HDL Analyst
- Finding Objects by Name
- Selecting Objects in HDL Analyst
- Crossprobing
- Setting Colors in HDL Analyst
- Changing Amount of Logic Displayed

Setting Up HDL Analyst for Synplify

HDL Analyst is included with some Synplicity products, but is sold as an option for the Synplify tool; it then requires an additional license to run. If you have an HDL Analyst license, enter it into your Synplify tool license file. If you need assistance entering this feature, ask your system administrator or contact Synplicity at this e-mail address:

support@synplicity.com

Viewing Object Information in HDL Analyst

Tooltip Information

When you put the mouse pointer over an object, you see a tooltip flag that displays information about the object. Tooltips display the names of instances, nets, ports, sheet connectors and attributes attached to them. For example, Instance: a_aux[7:0] of primitive type dff. If you are viewing a critical path in a Technology view, it also displays timing information. This following example shows tooltip information for a state machine:
To prevent tooltips from displaying, choose View -> Toolbars and disable Show Tooltips.

**Status Bar Information**

As you move the mouse pointer over objects, the status bar at the bottom of the window displays information about the object. The status bar displays the names of instances, nets, ports, sheet connectors and attributes attached to them. If you are viewing a critical path in a Technology view, it also displays timing information. The information is the same as that displayed in a tooltip, but it might contain a little additional information. This is an example of what you see for a net:

```
Net clk (local net clk) Fanout=4
```

If you prefer information not to be displayed when you move your mouse, turn off the View -> Status Bar menu item.

**Finding Objects by Name**

If you cannot locate the object you want based on the information in the tooltips or status bar, use the HDL Analyst -> Find command. The Object Properties dialog box lists objects by type (instances, symbols, nets, and ports) and lets you use wildcards to find them. For more information, see *Find Dialog Box on page 3-19.*
Finding Objects

You can use the Hierarchy Browser to browse and find objects. You can also use commands from the popup menu to find and select objects like net drivers or registers on a path.

Selecting Objects in HDL Analyst

This section outlines how to select and unselect objects in HDL Analyst.

Selecting Objects

This is the default mouse pointer mode, indicated by the crosshair pointer. You also have Push/Pop mode, indicated by one or two vertical arrows (Using Push/Pop Mode on page 5-13). To return to the default pointer mode from another mode, click the right mouse button.

- To select a single object, click it in the schematic or click its name in the Hierarchy Browser (left pane of the view).

  HDL Analyst highlights the selected object in red. Use the Hierarchy Browser to locate objects by name. You can also use the Find command to locate the objects you want.

- To select more than one object, do one of the following:
  - Drag a selection rectangle around the objects to select.
  - Add objects to the current selection by pressing and holding the Control key (Ctrl) while clicking the objects to be added.

- To select all of the objects of a certain type, right-click and choose Select All -> <object> from the popup menu.

- To select all the instances on a net or the net drivers, use the commands from the popup menu.

Unselecting Objects

To unselect highlighted objects, click the left mouse button in a blank area of the schematic or click the right mouse button to bring up the popup menu and choose Unselect All.
Setting Colors in HDL Analyst

You can customize the colors used in the RTL view and Technology view schematics by editing the `synplify.ini` file. On the Microsoft® Windows® operating system, this file is in the `WINDOWS` (or `WINNT`) directory. On the UNIX operating system, it is in the `windows` subdirectory of your home directory (`~/windows`, where `~` is your home directory, which can be set via environment variable `$HOME`).

You adjust the RGB (red, green, blue) values in the “[Schematics]” section of the file by editing the appropriate variables, as shown in the following table.

Table 5-1: Adjusting RGB values in HDL Analyst

<table>
<thead>
<tr>
<th>To Change...</th>
<th>Adjust Variable...</th>
</tr>
</thead>
<tbody>
<tr>
<td>The default color for instances, nets, and ports,</td>
<td><code>SSC_DEFAULT_LINE</code></td>
</tr>
<tr>
<td>The highlight (selected object) color</td>
<td><code>SSC_HILIT</code></td>
</tr>
<tr>
<td>The color for unselected objects</td>
<td><code>SSC_NOSELECTED</code></td>
</tr>
<tr>
<td>The window background color</td>
<td><code>SSC_BACKGROUND</code></td>
</tr>
</tbody>
</table>

Here are some example RGB values:

- `0 0 0 = black`
- `255 255 255 = white`
- `255 0 0 = red`
- `200 200 200 = gray`

Changing Amount of Logic Displayed

You can change the amount of logic displayed on a single schematic sheet in the Technology view, by editing the `PartitionSize=value` statement in the `.ini` file, as follows:

- To see your entire design on a single page, `value` should be greater than the total number of components in the design. Remember that a larger value generally means a longer time to display.
- To see a smaller area of your design on each schematic page, use a lower `value`. 
Crossprobing

Crossprobing Across Views

You can crossprobe objects and logic back and forth, from the RTL and Technology views to source code, post-place-and-route timing-report files, or a log file. Not all objects or source code crossprobe to other views, because some source code and RTL view logic is optimized away during synthesis. The following table summarizes crossprobing to and from the HDL Analyst view (RTL and Technology).

Table 5-2: Crossprobing with HDL Analyst views

<table>
<thead>
<tr>
<th>Crossprobe From</th>
<th>To...</th>
<th>Do...</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDL Analyst view</td>
<td>Other open view</td>
<td>Select an object in the RTL or Technology view. The object is automatically highlighted in all other open views.</td>
</tr>
<tr>
<td>HDL Analyst view</td>
<td>Unopened source code view</td>
<td>Double-click the object in the HDL Analyst view. For example, if you double-click a look-up table in a Technology view, the Synplify synthesis tool opens a source code window and highlights the part of the code that contains the selected look-up table.</td>
</tr>
<tr>
<td>A source code window</td>
<td>HDL Analyst view</td>
<td>Open the HDL Analyst view. To crossprobe to the Technology view, you must also have the RTL view open. The RTL view can be iconized. Select a part of the HDL code in the source code window. To jump to a specific RTL view sheet with the logic when you crossprobe, click the right mouse button over the selected text, and choose the sheet number.</td>
</tr>
</tbody>
</table>

Crossprobing to Visual Elite

Visual Elite™ is a GUI that you can use to create and edit HDL designs. Cross probing is available, in both directions, between Synplicity’s synthesis tools and Visual Elite. Visual Elite is a product of Innoveda™; for more information see http://www.innoveda.com/.
You can crossprobe to Visual Elite and trace component cause if you set up the Visual Elite software correctly. The following sections describe the setup for the Microsoft® Windows® and Unix operating systems, and describe how to crossprobe between the two.

- Setup on Microsoft® Windows® Operating System
- Setup on Unix Operating System
- Launching Synthesis from Visual Elite
- Crossprobing to Trace Component Cause

**Setup on Microsoft® Windows® Operating System**

To use Visual Elite and the Synplicity software together on the Microsoft® Windows® operating system, you need to do the following:

1. Make sure that Visual Elite tool is installed on your system, with a valid license.
2. Set variables.
   - Set the `VSH_LIB` environment variable to point to the directory:
     
     ```
     synplify_installation_dir/lib/innoveda/visual
     ```
   - Point the `VSH_VISUAL_COMMAND` environment variable to either the VHDL or Verilog versions:
     
     ```
     Vis_installation_dir/bin/vis_vhdl_version.exe
     Vis_installation_dir/bin/vis_verilog_version.exe
     ```
   - Set the `LM_LICENSE_FILE` environment variable to include the Synplicity license.
3. Set the path to include `synplify_installation_dir/bin/mbin`.
4. Set Visual Elite options:
   - Open Visual Elite, go to the Primary menu bar, and choose Options -> Synthesis -> Remote Configuration, which brings up a dialog box.
   - In the dialog box, choose No for the Remote Synthesis option.
   - In the Synthesis Directory text box, enter the path to any directory for which you have write permission. Click OK.
Setup on Unix Operating System

To use Visual Elite and the Synplicity software together on a UNIX platform, you need to do the following:

1. Make sure that the Visual Elite tool is installed on your system, and its license is in effect.

2. Set environment variables.
   - Set the VSH_LIB environment variable to point to the directory:
     
     synplify_installation_dir/lib/innoveda/visual
   
   - Point the VSH_VISUAL_COMMAND environment variable to either the Verilog or VHDL version:
     
     Vis_installation_dir/Vis_VHDL_version/platform/bin/visual_elite
     Vis_installation_dir/Vis_Verilog_version/platform/bin/visual_ver

3. Set the path.
   
   set path = (synplify_installation_dir/bin $path)

Launching Synthesis from Visual Elite

To launch synthesis from Visual Elite, do the following:

1. In Visual Elite, specify that the HDL to be generated from your design is intended for synthesis by the Synplify tool.

2. Open the editing window for the top-level design, then choose Synthesis -> Script to open the Synthesis Script editing window.


4. In the Run Synplify dialog box, enter the path to a predefined project file, and specify the required mode of work: batch or interactive.

Alternatively, you can choose the Synthesis -> Run Synplify command in the design unit’s editing window without having to first specify your synthesis tool.
Crossprobing to Trace Component Cause

You can crossprobe to trace the cause of a component from an RTL or Technology view back to a Visual Elite editor.

1. Open an RTL or Technology view.
2. Choose HDL Analyst -> External Cross Probing Engaged.
3. Click or double-click the component whose cause you want to trace.

   Clicking once opens the Visual Elite editor and highlights the cause of that component. Double-clicking highlights the cause in Visual Elite, and also indicate the cause in the Synplicity synthesis tool source file window.

Crossprobing with ModelSim

The Model Technology, Inc. ModelSim HDL simulator can be integrated into the Synplify tool, to help you to debug your design by crossprobing waveforms.

1. Make sure you have the following:
   - An installed copy of ModelSim EE or ModelSim PE, v5.2 or later.
   - A compiled project for crossprobing.

   Refer to the ModelSim User Guide for complete information on installing, starting, and running ModelSim.

2. In your Synplicity synthesis tool, do the following:
   - Compile your project.
   - Choose HDL Analyst -> External Crossprobing Engaged or type Ctrl-k to engage external crossprobing.

3. Start ModelSim, and do the following:
   - Load your design.
   - Type

     source synplify_installation_dir/lib/mti/manager.tcl
Traversing Hierarchy

Schematic views have different layers of design hierarchy. HDL Analyst provides you with two ways of traversing hierarchy, the Hierarchy Browser, and Push/Pop mode. This section discusses the following hierarchy-related topics:

- Viewing Objects Across Design Levels and Views on page 5-12
- Using the Hierarchy Browser on page 5-13
- Using Push/Pop Mode on page 5-13

Viewing Objects Across Design Levels and Views

Selecting an item in a given view or Hierarchy Browser causes that item to be crossprobed in all other open views and their Hierarchy Browsers. Items that can be crossprobed include ports, registers, and combination logic that has not been transformed.

Views track with their respective Hierarchy Browsers. Selecting an item causes the corresponding Hierarchy Browser to track or scroll to display the item. Selecting an item in a Hierarchy Browser causes the view to track to display that item. Views do not track when you crossprobe.

When a net is selected, HDL Analyst highlights it through all the hierarchical instances it traverses. As you traverse the hierarchy, you see the continuation of the selected net highlighted at different levels.
Using the Hierarchy Browser

The Hierarchy Browser is best used for an overview, or when you need to browse and find an object of which you are unsure. If you want to move between design levels of a particular object, the Push/Pop mode is more direct. Refer to Using Push/Pop Mode on page 5-13 for details.

When you start an RTL or Technology view, a Hierarchy Browser automatically starts in the left side of the view. The RTL browser displays the hierarchy specified in the RTL design description. The Technology browser displays the hierarchy of your design after technology mapping.

The Hierarchy Browsers allow you to traverse hierarchy and select ports, internal nets, components, and submodules. The browsers list the names of the objects and a symbol. A plus sign in a square icon indicates that there is hierarchy under that object. A minus sign indicates that the design hierarchy has been expanded.

Using Push/Pop Mode

Push/Pop mode is best suited for traversing a specific hierarchy. If you want a general view, use the Hierarchy Browsers described in Using the Hierarchy Browser on page 5-13.

1. To start Push/Pop mode, do one of the following:
   - Choose View -> Push/Pop Hierarchy.
   - Click the right mouse button while the mouse pointer is in an RTL or Technology view, then choose Push/Pop Hierarchy from the popup menu.
   - From the toolbar, click the Push/Pop Hierarchy button. This is the button containing an arrow pointing downward and an arrow pointing upward.
   - Press F2.

The mouse pointer changes to an arrow. An arrow pointing down means that you can push into (descend) the instance under the pointer. An arrow pointing up indicates there is a hierarchical level above this one. You see this arrow when the pointer is over blank areas of the schematic. The status bar at the bottom of the window reports information about the objects over which you move the
pointer, including when an object is a primitive. You cannot push into a primitive.

2. To push (descend) into an object, click it.

3. To pop (ascend) a level, move the mouse pointer to a blank area, then click.

4. To exit Push/Pop mode, do one of the following:
   – Right-click in an RTL or Technology view.
   – Choose Push/Pop Hierarchy from the View menu.
   – Click the Push/Pop Hierarchy icon in the toolbar.
   – Press F2.
Multisheet Schematics in HDL Analyst

Sheet Connectors

When you view a schematic that is too large to view on a single HDL Analyst schematic sheet, HDL Analyst partitions the design into multiple sheets. The Synplify synthesis tool inserts sheet connectors for nets that span multiple sheets. You can move to the sheet indicated by the connector by clicking the symbol.

Figure 5-1: Sheet connector

Viewing Other Sheets

The title bar in the RTL view or Technology view indicates how many sheets there are for the RTL-level or technology-primitive-level schematic. The schematic is initially opened to the first sheet (sheet number 1). For example:

adder: (RTL View) - sheet 1 of 1
alu: (Technology View) - sheet 1 of 13

The following steps show you common multisheet operations:

1. To view the next sheet of a multisheet schematic, do one of the following:
   - Choose View -> Next Sheet.
   - Right-click, and choose Next Sheet from the popup menu.
   - Click the Next Sheet icon ( ).
   - Type Ctrl-n.

2. To view the previous sheet, do one of the following:
   - Choose View -> Previous Sheet.
Chapter 5: Using HDL Analyst

Multisheet Schematics in HDL Analyst

- Right-click, then choose Previous Sheet from the popup menu.
- Click the Previous Sheet icon ( ),
- Type Ctrl-p.

3. To view a specific sheet number, do any of the following:
   - Choose View -> View Sheets, then specify the sheet number you want to view.
   - Right-click, then choose View Sheets from the popup menu, and specify the sheet number.
   - Type Ctrl-s to bring up the Sheet Selection dialog box, then specify the sheet number.

4. To view a schematic of selected items only, filter the schematic as described in Filtering Schematics in HDL Analyst on page 5-17.

5. To follow a net across sheets, click the right mouse button on a sheet connector to open the sheet to which the net is connected.
   Sheet connectors are symbols for nets that span multiple sheets. If the net is connected to multiple sheets, a popup menu lets you select the sheet.
Filtering Schematics in HDL Analyst

In HDL Analyst, you can put selected objects into their own schematic sheet. This is very useful for analyzing portions of your design. To do this, you use the Filter Schematic command, which takes all selected objects and groups them together in one schematic, filtering or removing all other objects.

This procedure shows you how to filter your schematic.

1. Select one or more objects that you want to filter. For example, you can select the objects on a critical path.

2. Execute the Filter Schematic command, using one of these methods:
   - Choose HDL Analyst -> Filter Schematic.
   - Right-click, then choose Filter Schematic from the popup menu.
   - Click the Filter on Selected Gates icon ( ).
   - Type the F12 key.

   You see a new schematic window with just the objects you selected.

3. To return to the previous schematic view, run the Filter Schematic command again, using one of the methods described in the previous step.
Analyzing Critical Paths in HDL Analyst

HDL Analyst can display the internal nets and instances in the critical path of your design only after you have successfully run synthesis. You can also view timing reports in the log file.

This section discusses the following topics:

- Viewing Critical Paths in HDL Analyst
- Interpreting Critical Path Information
- Handling Negative Slack
- Strategies

Viewing Critical Paths in HDL Analyst

To view critical paths in HDL Analyst:

1. Synthesize your design by clicking the Run button or choosing Run -> Synthesize.
2. Open a Technology view by choosing HDL Analyst -> Technology View.
3. Flatten your schematic using HDL Analyst -> Flatten current schematic.
4. Turn on critical path highlighting by choosing the HDL Analyst -> Show Critical Path menu item, then analyze the highlighted path. You can also click the Show Critical Path icon ( ), or the Show Critical Path command from the popup menu (right-click).

   The command highlights the instances and nets in the critical paths of your design and displays timing numbers above every instance. You can analyze your critical paths from this window, or you can isolate them into their own schematic by filtering them. For information on analyzing the path, see Interpreting Critical Path Information on page 5-19.

5. To isolate instances with the worst-case slack time in the critical path, choose HDL Analyst -> Filter Schematic. Alternatively, you can click the Filter on Selected Gates icon ( ), or choose Filter Schematic from the popup menu.
This command takes all selected objects and groups them together in one schematic, removing all other objects. All the selected instances are shown, regardless of their original sheet numbers.

6. Use the Filter Schematic command to restore your original schematic.

7. Use the Slack Margin control to view instances in your critical path that have less than the worst-case slack time.
   - Determine the slack time for your design.
   - Choose HDL Analyst -> Set Slack Margin.
   - Set a value for the slack margin, and click OK. The values are converted to negative values, and the slack margin is subtracted from the slack time for the design. The Technology view displays those instances that fall within the filtered range.

For example, if your slack time is 10 ns, and you set a slack margin of 4 ns, the critical path 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. To return to viewing just the instances with the worst-case slack time, enter a zero. For additional information on slack times, see Handling Negative Slack on page 5-21.

8. Add constraints, rerun synthesis, and check your results.

Refer to Strategies on page 5-23 for more information about strategies to use.

**Interpreting Critical Path Information**

HDL Analyst makes it simple to find and examine critical paths and the relevant source code.

1. Open a Technology view after synthesis.

2. For hierarchical designs, flatten the design by choosing HDL Analyst -> Flatten Schematic.

3. Turn on critical path highlighting by choosing HDL Analyst -> Show Critical Path, then analyze the highlighted path. You can also click the Show Critical Path icon ( ) or the Show Critical Path command from the popup menu (right-click).
The command highlights the instances and nets in the critical paths of your design and displays timing numbers above every instance. You can analyze your critical paths from this window, or you can isolate them in their own schematic to make it easier to analyze. This figure shows you how to interpret the numbers above the instance:

For combinational logic, this is the cumulative delay to the output of the instance, including the net delay of the output. For flip-flops, this 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 of the worst path that goes through the instance. A negative value indicates that timing has failed.

```
out[0] (dfm7a), Delay: 12.9 ns, Slack: -10.5 ns
```

4. To isolate instances with the worst-case slack time in the critical path, choose HDL Analyst -> Filter Schematic. Alternatively, you can click the Filter on Selected Gates icon ( ), or choose Filter Schematic from the popup menu.

This command takes all selected objects and groups them together in one schematic, removing all other objects. All the selected instances are shown, regardless of the sheet on which they were originally.

5. Use the Filter Schematic command to restore your original schematic.

6. View instances in the critical path that have less than the worst-case slack time:
   - Determine the slack time for your design.
   - Choose HDL Analyst -> Set Slack Margin.
   - Set a value for the slack margin, and click OK. The Synplify synthesis tool subtracts this number from the slack time to get a range, and the Technology view displays the instances that fall within the 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. To return to viewing just the instances with the worst-case slack time, enter a zero.
7. Use crossprobing to check the RTL view and the source code. Analyze the code and the schematic to determine how to address the problem. You can add more constraints or make code changes.

8. Rerun synthesis, and check your results.

HDL Analyst displays any remaining critical paths. Repeat this step until none or few critical paths remain.

When you are within 5 to 10 percent of your desired result, place-and-route your design to see if you have met your goal. If so, you are done. If your vendor provides timing-driven placement and routing, you might improve your results further by adding timing constraints to placement and routing.

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 failed 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 fix negative slack. Timing constraints can improve your design by 10 percent to 20 percent.

1. Isolate the critical path with Filter Schematic, and check the end points of the paths that are displayed. Look at the instances.

   The start point can be a primary input or a flip-flop. The end point can be a primary output or a flip-flop.

2. 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 attribute to an end point.

3. If there are no timing exceptions and timing is within about 20 percent of the goal, you can use the -improve option to speed up the design and meet timing:
– To speed up paths from primary inputs, add the `define_input_delay` timing constraint to the input. Use the input name displayed in HDL Analyst as the input port name argument for the timing constraint because names can vary depending on the source code: in Verilog, the name comes from the `reg register_name` declaration; in VHDL, the instance name is the signal name to which you assigned values to create the register. Specify the slack value as the `-improve` value. If the slack time is -2.4 ns, use `-improve 2.4`. The Synplify synthesis tool tries to optimize timing to meet this constraint.

– To speed up paths from flip-flops, add the `define_reg_output_delay` timing constraint to the flip-flop. Use the instance name displayed in HDL Analyst as the register name for the timing constraint. Specify the slack value as the `-improve` value.

– To speed up a path to a primary output, add the `define_output_delay` timing constraint to that output. Use the instance name displayed in HDL Analyst as the register name for the timing constraint. Specify the slack value as the `-improve` value.

– To speed up paths to flip-flops, add the `define_reg_input_delay` timing constraint on that flip-flop. Use the instance name displayed in HDL Analyst as the register name for the timing constraint. Specify the slack value as the `-improve` value.

4. If your design does not meet timing by 20 percent or more, you might need to make structural changes. You could do this by

– Using options like resource sharing.

– Changing the source code.

Rerun synthesis and check your results.
Strategies

Rerun the Synplify synthesis tool and check your results. Timing constraints can get you a 10 percent or 20 percent improvement. Rerun the tool, and then check your results with HDL Analyst. HDL Analyst displays any remaining critical paths. Add timing constraints and rerun. Follow this methodology until none or few critical paths remain.

When you are within 5 to 10 percent of your desired result, place-and-route your design to see if your vendor says that you have made your goal. If so, you are done. If your vendor provides timing-driven placement and routing, you might improve your results by adding timing constraints to placement and routing.
CHAPTER 6
Timing Constraints

This chapter contains information about timing constraints.

- About Timing Constraints on page 6-2
- SCOPE Window on page 6-3
- Tcl Constraint Files on page 6-20
- Individual Timing Constraints on page 6-24
- Black-Box Timing Models on page 6-36
- Forward Annotation on page 6-39
- Support for DCM/DLL in Xilinx on page 6-45
- Timing Report on page 6-46
Chapter 6: Timing Constraints

About Timing Constraints

This section provides an introduction to the timing constraints used in the Synplify synthesis tool. It lists the types of timing constraints and the methods you can use to specify them.

Specifying Timing Constraints

Timing constraints define the timing targets for the design that the Synplify synthesis tool must meet in order to improve synthesis results. Timing constraints define the performance goals for a design. They are passed to the synthesis environment in constraint files, called SDC (Synplicity design constraints) files. These files contain Tcl commands and have an “sdc” file extension (.sdc). Currently, you can define timing constraints using these methods:

- As attributes or constraints in the SCOPE spreadsheet, which automatically generates constraint files in Tcl format

  This is the easiest method for specifying constraints. Use it instead of the other methods wherever possible. You can use it for most constraints, except for source code directives. See SCOPE Window on page 6-3 for more information about this tool.

- As Tcl commands in an .sdc file that you create manually in a text editor.

  It is easier to use the SCOPE spreadsheet to generate the constraint syntax, as described in SCOPE Constraint Types on page 6-5. For information about Tcl constraint syntax for manually created constraint files, see Tcl Constraint Files on page 6-20.

- As source code attributes or directives

  You must enter timing directives like black-box directives in the source code, but you can use the SCOPE spreadsheet to enter attributes. See Specifying Attributes with the SCOPE Spreadsheet on page 7-9 for details. Including timing constraints in the source code makes the source code less portable and requires that you recompile the design for the constraints to take effect.
SCOPE Window

This section describes the following aspects of the SCOPE spreadsheet:

- SCOPE Spreadsheet on page 6-3
- Opening a Constraint File with the SCOPE Spreadsheet on page 6-4
- SCOPE Constraint Types on page 6-5
- Clocks Panel on page 6-6
- Inputs/Outputs Panel on page 6-10
- Registers Panel on page 6-11
- Multi-Cycle Paths Panel on page 6-12
- False Paths Panel on page 6-14
- Attributes Panel on page 6-17
- Other Panel on page 6-17
- SCOPE File Conversions on page 6-17
- SCOPE Constraint Wizards on page 6-18

SCOPE Spreadsheet

The SCOPE (Synthesis Constraints Optimization Environment™) window presents a spreadsheet interface for entering and managing timing constraints and synthesis attributes.

From the SCOPE window you can set

- Timing constraints for clocks, ports, and registers. For more information, see Individual Timing Constraints on page 6-24. You cannot set black-box constraints from the SCOPE window. The information you enter in the SCOPE window is stored in an .sdc constraint file that must be included in your project.

- Synthesis attributes. For more information, see Chapter 7, Synthesis Attributes and Directives.
Opening a Constraint File with the SCOPE Spreadsheet

1. To create a new constraint file from the SCOPE window, do one of the following in an open, compiled design:
   - Choose File -> New -> Constraint file (SCOPE) from the Project view.
   - Click the SCOPE icon in the toolbar.

If you have compiled your design and this is the first constraint file for your design, a prompt asks you if you want to initialize selected timing constraints. You choose the types of constraints you want, using the check boxes. Unless you are an advanced user, simply accept the default settings. Click Initialize to automatically initialize the new constraint file with the appropriate settings for the chosen constraints. Once you have entered your constraints using the SCOPE spreadsheet, save the file. You can optionally add it to the project file when you save it.

Figure 6-1: Opening a SCOPE constraint file
The SCOPE window opens. It has panels with tabs along the bottom edge. Each panel has different fields where you can set constraints and attributes. The window also includes a wizard for each of the first three panels (Clocks, Inputs/Outputs, Registers), to help you enter clock, input / output, and register constraints.

2. To open an existing constraint file in the SCOPE spreadsheet, do either of the following:
   - Double-click the file in the Project view.
   - Choose File -> Open in the Project view. Change the Files of type: filter to Constraint files (SCOPE), then double-click to select the file you want.

If you use a text editor to edit a constraint file, close the SCOPE spreadsheet before editing the file so the save operation does not overwrite the text editor save operation or vice versa.

**SCOPE Constraint Types**

Use the different SCOPE panels to enter specific constraint information. The following table shows the constraints you can set:

<table>
<thead>
<tr>
<th>Constraints</th>
<th>SCOPE Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Clocks Panel on page 6-6</td>
</tr>
<tr>
<td>Input/output delays</td>
<td>Inputs/Outputs Panel on page 6-10</td>
</tr>
<tr>
<td>Register constraints</td>
<td>Registers Panel on page 6-11</td>
</tr>
<tr>
<td>Multicycle path exceptions</td>
<td>Multi-Cycle Paths Panel on page 6-12</td>
</tr>
<tr>
<td>False path exceptions</td>
<td>False Paths Panel on page 6-14</td>
</tr>
<tr>
<td>Other constraints</td>
<td>Other Panel on page 6-17</td>
</tr>
</tbody>
</table>

You can also use the SCOPE spreadsheet to set attributes. See *Specifying Attributes with the SCOPE Spreadsheet on page 7-9* for details.
Chapter 6: *Timing Constraints*

**Clocks Panel**

You use the Clocks panel of the SCOPE spreadsheet to define a signal as a clock. The equivalent Tcl constraint file command is `define_clock`; its syntax is described in `define_clock` on page 6-25. For information about the Clocks wizard, see *SCOPE Constraint Wizards* on page 6-18.

Clocks fields include the following parameters:

**Table 6-2: SCOPE Clocks panel**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>Clock</td>
<td>(Required.) Specifies the name of the clock. The name can be either a top-level port in the design or the name of an internal instance used as a clock generator for the chip. In the case of virtual clocks, the field must contain a unique name not associated with any port or instance in the design. See the Virtual Clock field for more information.</td>
</tr>
<tr>
<td>Frequency (Mhz)</td>
<td>(Required.) Specifies the clock frequency in MHz. If you fill in this field, and click in the Period field, the Period value is filled in automatically. For frequencies other than that implied by the clock pin, refer to <code>syn_reference_clock</code> on page 6-35.</td>
</tr>
<tr>
<td>Period (ns)</td>
<td>(Required.) Specifies the clock period in nanoseconds. If you fill in this field, and click in the Frequency field, the frequency is filled in automatically.</td>
</tr>
<tr>
<td>Clock Group</td>
<td>Allows you to specify clock relationships. You put related (synchronized) clocks in the same clock group, and put unrelated clocks in different groups.</td>
</tr>
<tr>
<td></td>
<td>• The Synplify synthesis tool calculates the relationship between clocks in the same clock group, and analyzes all paths between them. Paths between clocks in different groups are ignored (considered to be false paths).</td>
</tr>
<tr>
<td></td>
<td>• By default, clocks are in the same clock group (default_clkgroup is the default name).</td>
</tr>
<tr>
<td></td>
<td>See <em>Clock Groups</em> on page 6-7 for more information.</td>
</tr>
</tbody>
</table>
The timing engine uses clock groups to analyze and optimize the design. It assumes that clocks in the same clock group are synchronized with each other and treats them as related clocks. Typically, clocks in a clock group are derived from the same base clock. The timing analyzer automatically calculates the relationships between the related clocks in a clock group and analyzes all paths between them. For paths between clocks of the same group, the timing engine calculates the time available based on the period of the two clocks.

**Clock Groups**

Table 6-2: SCOPE Clocks panel (Continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rise At (ns)</td>
<td>By default, the tool assumes that the clock is a 50% duty cycle clock, with rising edge at 0 and falling edge at ( \text{Period}/2 ). You can specify different rising and falling edges. See <em>Rise and Fall Constraints on page 6-9</em> for details.</td>
</tr>
<tr>
<td>Fall At (ns)</td>
<td></td>
</tr>
<tr>
<td>Duty Cycle (%)</td>
<td>Specifies the clock duty cycle as a percentage of the clock period. If you have a duty cycle that is not 50% (the default), specify the rising and falling edge values to get the duty cycle you want.</td>
</tr>
<tr>
<td>Route (ns)</td>
<td>Improves the path delays of registers controlled by the clock. The value shrinks the effective period for synthesis, without affecting the clock period that is forward-annotated to the place-and-route tool. This is an advanced user option.</td>
</tr>
<tr>
<td>Virtual Clock</td>
<td>Designates the clock as a virtual clock. A virtual clock allows you to associate arrival and required times with clocks external to the chip (or block) that you are synthesizing. The clock name field must be a unique name not associated with any port or instance in the synthesized design.</td>
</tr>
<tr>
<td>Comment</td>
<td>Lets you enter comments that are included in the constraints file.</td>
</tr>
</tbody>
</table>
Chapter 6: Timing Constraints

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

Figure 6-2: Worst case posedge-to-posedge timing

The following figure shows how the timing analyzer calculates worst case edge-to-edge timing between all possible transitions of the two related clocks. In this example, CLK1 has a period of 5 ns and CLK2 has a period of 10 ns.

Figure 6-3: Worst case edge-to-edge timing

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

By default, all clocks in a design are assigned to the same clock group, called default_clkgroup. Although setting a clock group is optional, it is recommended that you assign all clocks to explicit clock groups. If you have an unrelated clock, you must re-assign the unrelated clock to a new clock group using the Clock Group field in the SCOPE spreadsheet.
For example, if your design has three clocks (clk1, clk2 and clk3), by default they are all assigned to default_clkgroup. All paths between the three clocks are analyzed and optimized. If clk2 is unrelated to the other two clocks but is left in the same clock group, it skews the results of timing analysis and negatively affects optimization because the timing engine treats it as a related clock. Similarly, in cases where clock relationships are forward-annotated, it negatively affects place-and-route results.

For more accurate results, specify a different clock group name for clk2, such as clkgrp2. The timing analyzer now only analyzes the relationship between clk1 and clk3. Paths between clk1 and clk2, and clk3 and clk2 are considered false paths and are ignored.

**Rise and Fall Constraints**

The Synplify synthesis tool assumes an ideal clock network. By default, the constraints assume a 50% duty cycle clock with the rising edge at 0 and the falling edge at Period/2.

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

---

**Figure 6-4: Defining clocks with the SCOPEspreadsheet**

In this example,

- Clk1 is a clock with a period of 10 ns in clock group default_clkgroup.
  Since none of the other fields are specified, this is a 50% duty cycle clock rising at 0 and falling at 5.
• Clk2 is a 200 MHz (5ns) clock, also in default clkgroup. This means that the timing analyzer considers all paths from Clk1 to Clk2 and from Clk2 to Clk1.

• Clk3 is a clock with a period of 20 ns in clkgrp2. This means all paths between Clk3 and either Clk1 or Clk2 are automatically treated as false paths. In addition, Clk3 has a Rise At value of 0 and a Fall At value of 12, which means it has a 60% duty cycle.

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

Inputs/Outputs Panel

This panel models the interface of the FPGA with the outside environment. You use it to specify delays outside the device. The default delay outside the FPGA is 0.0 ns. The equivalent Tcl constraint file commands are define_input_delay and define_output_delay. See define_input_delay on page 6-29 and define_output_delay on page 6-31 for details For information about the Inputs/Outputs wizard, see SCOPE Constraint Wizards on page 6-18.
The SCOPE Inputs/Outputs fields are shown in the following table:

Table 6-3: SCOPE Inputs/Outputs panel

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(Required.) Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>Port</td>
<td>(Required.) Specifies the name of the port. If you have initialized a compiled design, you can select a port name from the pulldown list. The first two entries let you specify global input and output delays.</td>
</tr>
<tr>
<td>Type</td>
<td>(Required.) Specifies whether the delay is an input or output delay. See define_input_delay on page 6-29 and define_output_delay on page 6-31 for details</td>
</tr>
<tr>
<td>Clock Edge</td>
<td>The rising or falling edge that controls the event. The syntax for this field is the clock name, followed by a colon and the edge: r for rising, and f for falling. For example, CLK1:r.</td>
</tr>
<tr>
<td>Value</td>
<td>(Required.) Specifies the delay value. You do not need this value if you supply a Route value.</td>
</tr>
<tr>
<td>Route</td>
<td>Improves the delay of the paths to and from the port. The value shrinks the effective synthesis constraint without affecting the constraint that is forward-annotated to the place-and-route tool. This is an advanced user option.</td>
</tr>
<tr>
<td>Comment</td>
<td>Lets you enter comments that are included in the constraints file.</td>
</tr>
</tbody>
</table>

Registers Panel

Speeds up paths feeding into or out of a register by a given number of nanoseconds. See define_reg_input_delay on page 6-27, and define_reg_output_delay on page 6-28, for details. For information about the Registers wizard, see SCOPE Constraint Wizards on page 6-18.
The Registers SCOPE panel includes the following fields:

Table 6-4: SCOPE Registers panel

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(Required.) Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>Register</td>
<td>(Required.) Specifies the name of the register. If you have initialized a compiled design, you can choose from the pulldown list.</td>
</tr>
<tr>
<td>Type</td>
<td>(Required.) Specifies whether the delay is an input or output delay. See <code>define_reg_input_delay</code> on page 6-27, and <code>define_reg_output_delay</code> on page 6-28, for details</td>
</tr>
<tr>
<td>Route</td>
<td>Improves the speed of the paths to or from the register. The value shrinks the effective period for the constrained registers without affecting the clock period that is forward-annotated to the place-and-route tool. This is an advanced user option.</td>
</tr>
<tr>
<td>Comment</td>
<td>Lets you enter comments that are included in the constraints file.</td>
</tr>
</tbody>
</table>

Multi-Cycle Paths Panel

This panel lets you specify paths with multiple clock cycles. The following table defines the parameters for this constraint.

Table 6-5: SCOPE Multi-Cycle Paths panel

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(Required.) Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>From</td>
<td>Specifies the starting point for the timing exception. From points can be registers, top-level input or bidirectional ports. For more information, see <code>Defining Paths/Points for Timing Exceptions</code> on page 6-15.</td>
</tr>
<tr>
<td>To</td>
<td>Specifies the ending point for the timing exception. To points can be registers, top-level output or bidirectional ports. See <code>Defining Paths/Points for Timing Exceptions</code> on page 6-15 for more information.</td>
</tr>
</tbody>
</table>
Any combination of from, to, and through points is allowed: from–through, from, from–through–to, and so on. The following shows an example:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Through</td>
<td>Specifies the intermediate points for the timing exception. You can use any nets in the design as intermediate points. The intermediate points can be specified as a space-separated list, which is treated as an OR list. The exception is applied to all paths that cross any instance in the list. See Defining Paths/Points for Timing Exceptions on page 6-15.</td>
</tr>
<tr>
<td>Cycles</td>
<td>Number of cycles for the required time calculated for the path.</td>
</tr>
<tr>
<td>Comment</td>
<td>Lets you enter comments that are included in the constraints file.</td>
</tr>
</tbody>
</table>

In the following figure, CLK1 has a period of 10 ns. The data in this path has only 1 clock cycle before it must reach D2. To allow more time for the signal to complete this path, add a multicycle constraint that specifies 2 clock cycles (or 20 ns) for the data to reach D2.
Chapter 6: Timing Constraints

SCOPE Window

Figure 6-6: Multicycle constraint example

**False Paths Panel**

Use the False Paths constraint to specify clock paths that you want the Synplify synthesis tool to ignore during timing analysis and assign low (or no) priority during optimization.

Table 6-6: SCOPE False Paths panel

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(Required.) Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>From</td>
<td>Specifies the starting point for the timing exception. From points can be registers, top-level input or bidirectional ports, or black-box outputs. For more information, see <em>Defining Paths/Points for Timing Exceptions on page 6-15.</em></td>
</tr>
</tbody>
</table>
Table 6-6: SCOPE False Paths panel (Continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To</td>
<td>Specifies the ending point for the timing exception. To points can be registers, top-level output or bidirectional ports, or black-box inputs. For more information, see <em>Defining Paths/Points for Timing Exceptions</em> on page 6-15.</td>
</tr>
<tr>
<td>Through</td>
<td>Specifies the intermediate points for the timing exception. You can use any nets in the design as intermediate points. The intermediate points can be specified as a space-separated list, which is treated as an OR list. The exception is applied to all paths that cross any instance in the list. See <em>Defining Paths/Points for Timing Exceptions</em> on page 6-15.</td>
</tr>
<tr>
<td>Comment</td>
<td>Lets you enter comments that are included in the constraints file.</td>
</tr>
</tbody>
</table>

Any combination of from, to, and through points is allowed: from–through, from, from–through–to, and so on. The following is an example:

![Image of SCOPE False Paths panel](image)

**Figure 6-7: Specifying false paths**

**Defining Paths/Points for Timing Exceptions**

Multicycle paths and false paths are timing exceptions that you must identify. You specify points for multicycle and false paths from the appropriate SCOPE panels. For both types of paths, you can specify the exceptions as constraints on starting (From), ending (To), or intermediate (Through) points.

The following table lists the that objects you can constrain as starting and ending points. If you have compiled your design, you can choose the object you want from the pulldown list.
You can specify multiple From points in a single constraint, such as all the bits of a bus: From A[0:15]. Similarly, you can specify multiple To points in a single constraint. You can combine starting and ending points to specify a constraint From A[0:15] to B, for example. In this case, the constraint starts at any of the bits of A and ends at B. If you specify multiple starting points and multiple ending 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 \(16 \times 16 = 256\) combinations of start/end points.

Intermediate points are nets in the design, that you specify in the Through field. You can specify a list of intermediate points as a space-separated list (A B C). This list is treated as an OR list, and the constraint is applied to any path being analyzed that crosses any of the specified Through points. The order of intermediate points is not important. If you specify a bus as a Through point, it is treated as any other list: the path must go through at least one of the bits of the bus.

If you have multiple constraints on the same object, the timing analyzer uses the following rules to determine how the constraints are applied:

- Bit constraints override bus constraints. For example, if you have a constraint From A[0:15] to B, and a second constraint 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 Synplify synthesis tool uses the tightest constraint. If you have 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) are constrained to 3 cycles.
Attributes Panel

You use this panel to specify design attributes. For details, see *Specifying Attributes with the SCOPE Spreadsheet on page 7-9.*

Other Panel

The Other panel is intended for advanced users to enter newly-supported constraint file commands. This panel contains the following fields.

Table 6-8: Other panel

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(Required.) Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>Command</td>
<td>(Required.) Specifies the command that you want to pass to the place-and-route tool.</td>
</tr>
<tr>
<td>Arguments</td>
<td>(Required.) Specifies arguments to the command.</td>
</tr>
<tr>
<td>Comment</td>
<td>Lets you enter a comment about the commands that you are passing to the place-and-route tool.</td>
</tr>
</tbody>
</table>

SCOPE File Conversions

- If you use version 7.0 or later to open a SCOPE constraint file generated with an older version, check the following:
  - Previously entered values are displayed, but not modified. Only commands entered in the new SCOPE window are saved. When you save the constraints, the constraints are saved in the new format.
  - Check warnings for clock_to_clock delays, which are no longer supported.
  - Check designs with multiple clocks. By default, all clocks are assigned to the default group.
- If you use an older version of the Synplify synthesis tool to open a SCOPE file generated with a version of the tool after 7.0, you see warnings for unsupported constraints, like path constraints.
If you open a SCOPE file that is generated by another Synplicity product, you might see warnings for unsupported constraints.

**SCOPE Constraint Wizards**

Some SCOPE panels have a wizard that you can use to help you enter constraints. To start its wizard, display a SCOPE panel, right-click, and choose *Insert Wizard* from the popup menu. The wizard is best used for entering a number of constraints, or for setting defaults.

Each constraint wizard guides you through two dialog boxes similar to these:

![Figure 6-8: Insert Wizard dialog box (Inputs/Outputs panel)](image)
The following table shows you what to do for each SCOPE panel:

**Table 6-9: SCOPE panel wizards**

<table>
<thead>
<tr>
<th>Wizard (SCOPE Panel)</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clocks</td>
<td>For each clock, select the clock from the list, then click the right arrow to move it to the selected list. If you do not see a list, disable Exclude Duplicates. Click Next.</td>
<td>Set the <strong>Period</strong> and turn on the <strong>Enabled</strong> check box. The other fields are optional. Click <strong>Finish</strong>.</td>
</tr>
<tr>
<td>Inputs/Outputs</td>
<td>For each constraint select the port from the list, then click the right arrow to move it to the selected list. If you do not see a list, disable Exclude Duplicates. Click Next.</td>
<td>Set the values you want, and turn on the <strong>Enabled</strong> check box. Click <strong>Finish</strong>.</td>
</tr>
<tr>
<td>Registers</td>
<td>For each constraint select the register from the list, then click the right arrow to move it to the selected list. If you do not see a list, disable Exclude Duplicates. Click Next.</td>
<td>Set the values you want and turn on the <strong>Enabled</strong> check box. Click <strong>Finish</strong>.</td>
</tr>
</tbody>
</table>
Tcl Constraint Files

You can manually create constraint files in a text editor using Tcl commands, but it is recommended that you instead use the SCOPE window to generate the file automatically. Manually generated constraint files must have an “sdc” file extension (.sdc). They can include timing constraints, general attributes, and vendor-specific attributes.

After you create the constraint file, include it in the project. Add constraint files to the source files list of a project by clicking Add in the Project view and then selecting the file from the list. You might need to change the List Files of Type option to constraint (.sdc) files to view the available constraint files. Alternatively, you can read in a constraint file during a synthesis run from the project file with the add_file -constraint command.

Guidelines for Creating Constraint Files

The following are general guidelines for writing constraint files:

- Constraint files are written in Tcl, which is case sensitive.
- Make sure object names (signals, ports, instances) match the names found in the RTL view.
- Enclose all objects with curly braces { }. For example: {foo[2]}.
- Use a period (.) as the separator character between levels of hierarchy to reference instances in lower levels of the design hierarchy.
- Distinguish between instances and ports with the same name by using the correct prefix to identify the object. For example, a prefix of p: indicates that the constraint is attached to a port. See Object Naming Syntax on page 6-21 for details about the prefixes.
- Use a prefix of n: for internal nets, as described in Object Naming Syntax on page 6-21.
Object Naming Syntax

This section describes the prefix syntax for identifying objects when different design objects share the same name. Objects like Verilog modules, VHDL design units, component instances, ports, and internal nets can have shared names. The prefix identifiers are used in constraint files to assign timing constraints and synthesis attributes to the correct object. The syntax varies slightly, depending on the language used to define the module or component.

Verilog

This section provides syntax for object names in Verilog. No spaces are allowed in names. You can use the * and ? characters as wildcards in names. These characters do not match the dot (.) used as a hierarchy separator.

Verilog uses the following syntax for module names:

\[
\text{v: cell}
\]

\[
\text{v:}
\quad \text{Identifies a view object}
\]

\[
\text{cell}
\quad \text{The name of the Verilog module}
\]

Instance, ports and net names have the following syntax:

\[
\text{cell typespec object\_path}
\]

\[
\text{cell}
\quad \text{The name of the Verilog module}
\]

\[
\text{typespec}
\quad \text{A single letter followed by a colon. The letter can be one of the following, and is used to determine the kind of object when objects have the same names:}
\]

\[
i: \text{identifies the name of an instance.}
\]

\[
p: \text{identifies the name of an entire port (the port itself).}
\]

\[
b: \text{identifies the name of a slice of a port (bit-slice of the port).}
\]

\[
n: \text{identifies the name of an internal net. This designation is required for all internal nets.}
\]

\[
\text{object\_path}
\quad \text{An instance path with a dot (.) used as a hierarchy separator in the name. The object path ends with the name of the desired object.}
\]
See *Object Naming Examples* on page 6-23.

**VHDL**

This section provides syntax for object names in VHDL. No spaces are allowed in names. You can use the * and ? characters as wildcards in names. These characters do not match the dot (.) used as a hierarchy separator.

VHDL has the following syntax for design unit names:

```
v:[lib.]cell [view]
```

- **v**: Identifies a view object
- **lib**: The name of a library that contains the design unit. The default is the library containing the top-level design unit.
- **cell**: The name of the design unit’s entity.
- **view**: The name of the design unit’s architecture. The use of *view* with VHDL designs is optional, and is required only when the design unit has more than one architecture.

Instance, port and net names have the following syntax:

```
[[lib.]cell [view]] | [typespec] objpath
```

- **lib**: The name of a library that contains the design unit. The default is the library containing the top-level design unit.
- **cell**: The name of the VHDL entity.
### Naming Objects in the SCOPE Window

If you choose an object from a SCOPE pulldown menu, it has the appropriate prefix appended automatically. If you drag and drop an object from an RTL view, for example, make sure to add the prefix appropriate to the language used for the module.

#### Object Naming Examples

To identify all bits of instance statereg in module statemod:

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

To identify instances one level of hierarchy from the top with names that begin with state:

```
i:*state*
```

To identify port mybus[19:0] instead of a driving register that also has the name mybus[19:0]:

```
p:mybus[19:0]
```
To identify top-level port mybus bit 1 instead of a driving register that also has the name mybus[1]:

\[ b::\text{mybus}[1] \]

The following shows the constraint file (.sdc) entries that correspond to the previous source code examples:

\[
\text{define_attribute \{statemod|i:staterg[*]\} syn_encoding sequential}
\]
\[
\text{define_multicycle_path -to \{*.slowreg[*]\} 2}
\]

**Individual Timing Constraints**

This section discusses the following timing constraints:

- define_clock on page 6-25
- define_reg_input_delay on page 6-27
- define_reg_output_delay on page 6-28
- define_input_delay on page 6-29
- define_output_delay on page 6-31
- define_multicycle_path on page 6-32
- define_false_path on page 6-34
- syn_reference_clock on page 6-35

For black-box timing models, see *Black-Box Timing Models* on page 6-36.

The synthesis tool global frequency constraint is set in the Frequency (MHz) field of the Project view. This constraint can also be set in the project file using the `set_option -frequency` Tcl command.
define_clock

Defines a clock with a specific frequency or clock period goal, and duty cycle. For the equivalent SCOPE interface, see Clocks Panel on page 6-6.

You can have multiple clocks with different clock frequencies. Set the default frequency for all clocks with the set_option -frequency Tcl command. Then use the define_clock timing constraint to override the default, and to specify unique clock frequency goals for specific clock signals. Additionally, you can use the define_clock timing constraint to set the clock frequency for a clock signal output of clock divider logic. The clock name is the output signal name for the register instance.

This is the syntax:

```
define_clock [-disable] [-name {clock_name}] [[-freq MHz | -period ns]] [-clockgroup domain] [[-rise value] [-fall value] | [-duty_pct percent]] [-route ns] -virtual -comments
```

- **-disable** Disables a previous clock definition.
- **-name** (Required). Specifies the clock name. The name can be either a top-level port in the design or the name of an internal instance used as a clock generator for the chip. In the case of virtual clocks, the field can contain a unique name not associated with any port or instance in the design.
- **-freq** Defines the frequency of the clock in MHz. You can specify either this or -period, but not both.
- **-period** Specifies the period of the clock in ns. You can specify either this or -freq, but not both.
- **-clockgroup** Allows you to specify clock relationships. You put related (synchronized) clocks in the same clock group, and put unrelated clocks in different groups.
  - The Synplify synthesis tool calculates the relationship between clocks in the same clock group, and analyzes all paths between them. Paths between clocks in different groups are ignored (considered to be false paths).
  - By default, clocks are in the same clock group (default_clkgroup is the default name).

See Clock Groups on page 6-7 for more information.
-rise / -fall  
Specifies a non-default duty cycle. By default, the Synplify synthesis tool assumes that the clock is a 50% duty cycle clock, with the rising edge at 0 and the falling edge at period/2. If you have another duty clock cycle, specify the appropriate Rise At and Fall At values. For more information, see Rise and Fall Constraints on page 6-9.

-duty_pct  
Specifies the duty cycle of the clock as a percentage of the period. When the file is saved, the Synplify synthesis tool translates this to -rise and -fall.

-route  
An advanced user option that improves the path delays of registers controlled by this clock. The value used is the difference between the synthesis timing report value of the path delays and the value in the timing report generated from the place-and-route tool. The -route constraint applies globally to the clock domain, and can overconstrain registers where constraints are not needed.

Before you use this option, evaluate the path delays on individual registers in the optimization timing report, and try to improve the delays with the define_reg_input_delay and define_reg_output_delay constraints only on the registers that need it. See define_reg_input_delay on page 6-27 and define_reg_output_delay on page 6-28 for more information.

-virtual  
Specifies arrival and required times on top level ports that are enabled by clocks external to the chip (or block) that you are synthesizing. When specifying -name for the virtual clock, the field can contain a unique name not associated with any port or instance in the design.

Here are some examples:

```plaintext
define_clock -name {clock} -period 10.0 -clockgroup d1
define_clock -virtual -name {CLK5} -period 20.0 -clockgroup d2
define_clock -name {CLK4} -period 20.000 -clockgroup d2
    -rise 0.000 -fall 10.000
```
**define_reg_input_delay**

Speeds up paths feeding a register by a given number of nanoseconds. The Synplify synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with `define_clock`). Use this constraint to speed up the paths feeding a register. For information about the equivalent SCOPE interface, see *Registers Panel on page 6-11*.

This is the syntax:

```
define_reg_input_delay register_name [-route ns] [-disable] [-comments]
```

- `register_name`: A single bit register. For multiple-bit registers (buses), use multiple `define_reg_input_delay` constraints.
- `route`: Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register. See *Example: -route option on page 6-27* for an example of its use.

The slack time reported on a register is the amount of time by which the clock frequency goal is exceeded (positive slack time), or not met (negative slack time). You can view slack times through HDL Analyst or in the timing report section of the log file.

**Example: -route option**

Use this option if you do not meet timing goals because the routing delay after placement and routing exceeds the delay predicted by the Synplify synthesis tool.

Rerun synthesis using this option, including the actual route delay (from place-and-route results) so that the tool can meet the required clock frequency. Assume the frequency goal is 50 MHz (clock period goal of 20.0 ns). If the synthesis timing reports show the frequency goal is met, but the detailed timing report from placement and routing shows that the actual clock period is 21.8 ns because of paths to reg_a, you can either tighten the constraints in the place-and-route tool, or you can rerun optimization using `-route 1.8` to the `define_reg_input_delay` constraint.

```
define_reg_input_delay reg_a -route 1.8
```
Using this option adds 1.8 ns to the route calculations for paths to `reg_a` so that timing is improved by 1.8 ns.

**define_reg_output_delay**

Speeds up paths coming from a register by a given number of nanoseconds. The Synplify synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with `define_clock`). Use this constraint to speed up the paths coming from a register. For information about the equivalent SCOPE interface, see *Registers Panel on page 6-11*.

This is the syntax:

```
define_reg_output_delay register_name [-route ns] -comments
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-disable</code></td>
<td>Disables a previous clock definition.</td>
</tr>
<tr>
<td><code>register_name</code></td>
<td>A single bit register. For multiple register bits (buses), use multiple <code>define_reg_output_delay</code> constraints.</td>
</tr>
<tr>
<td><code>-route</code></td>
<td>Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths from the register. For an example of its use (on an input register), see <em>Example: -route option on page 6-27</em>.</td>
</tr>
</tbody>
</table>

The slack time reported on a register is the amount of time by which the clock frequency goal is exceeded (positive slack time), or under achieved (negative slack time). You can see these slack times through HDL Analyst or in the timing report section of the log file.
**define_input_delay**

Specifies the external input delays on top-level ports in the design. It is the delay outside the chip before the signal arrives at the input pin. This constraint is used to model the interface of the inputs of the FPGA with the outside environment. The default delay outside the FPGA is 0.0 ns. Signals typically do not arrive with 0.0 ns delay, but the tool has no way of knowing what the input delay is unless you specify it in a timing constraint. For information about the equivalent SCOPE interface, see *Inputs/Outputs Panel* on page 6-10.

Here is the syntax:

```
define_input_delay [-disable] [{input_port_name} | -default] ns [-route ns] -ref clock [r | f] -comments
```

**-disable** Disables a previous delay specification on the named port.

**input_port_name** The name of the input port.

**-default** Sets a default input delay for all inputs. Use this option to set an input delay for all inputs. You can then set `define_input_delay` on individual inputs to override the default constraint. This example sets a default input delay of 3.0 ns:

```
define_input_delay -default 3.0
```

This constraint overrides the default and sets the delay on `input_a` to 10.0 ns:

```
define_input_delay {input_a} 10.0
```

**-ref** Clock name and edge that triggers the event. Value must be either 
- `r` - rising edge 
- `f` - falling edge 
For example: `define_input_delay {portb[7:0]} 10.00 -ref clock2:f`

**-route** An advanced option, which includes route delay when the Synplify synthesis tool tries to meet the clock frequency goal. Use the `-route` option on an input port when the place-and-route timing report shows that the timing goal is not met because of long paths through the input port. See *Example: -route option on page 6-30* for an example of its use.
Here are some examples of the `define_input_delay` constraint:

```plaintext
define_input_delay {porta[7:0]} -ref clock:r
define_input_delay {A} -ref CLK1:f
define_input_delay -disable {resetn}
```

The slack time reported on an instance along a path from an input port is the amount of time by which the clock frequency goal is exceeded (positive slack), or not met (negative slack) because of the input delay. You can view slack times through HDL Analyst or in the timing report section of the log file.

**Example: -route option**

Use this option if you do not meet timing goals because the routing delay after placement and routing exceeds the delay predicted by the Synplify synthesis tool.

Rerun synthesis using this option, including the actual route delay (from place-and-route results) so that the tool can meet the required clock frequency. Assume the frequency goal is 50 MHz (clock period goal of 20.0 ns). If the synthesis timing reports show the frequency goal is met, but the detailed timing report from placement and routing shows that the actual clock period is 21.8 ns because of paths through `input_a`, you can either tighten the constraints in the place-and-route tool, or you can rerun optimization using `-route 1.8` to the `define_input_delay` constraint.

```plaintext
# In this example, input_a has a 10.0 ns input delay.
# Rerun synthesis with an added 1.8 ns constraint, to
# improve accuracy and match post-place-and-route results.

define_input_delay {input_a} 10.0 -route 1.8
```

Using this option adds 1.8 ns to the route calculations for paths to `input_a` and improves timing by 1.8 ns.
**define_output_delay**

Specifies the delay of the logic outside the FPGA driven by the top-level outputs. This constraint is used to model the interface of the outputs of the FPGA with the outside environment. The default delay outside the FPGA is 0.0 ns. Output signals typically drive logic that exists outside the FPGA, but the tool has no way of knowing the delay for that logic unless you specify it using a timing constraint. For information about the equivalent SCOPE interface, see *Inputs/Outputs Panel on page 6-10.*

Here is the syntax:

```
```

- **-disable** Disables a previous delay specification on the named port.

- **output_port_name** The name of the output port.

- **-default** Sets a default input delay for all outputs. Use this option to set a delay for all outputs. You can then set `define_output_delay` on individual outputs to override the default constraint. This example sets a default output delay of 8.0 ns. The delay is outside the FPGA.

  ```
  define_output_delay -default 8.0
  ```

  The following constraint overrides the default and sets the output delay on `output_a` to 10.0 ns. This means that `output_a` drives 10 ns of combinational logic before the relevant clock edge.

  ```
  define_output_delay {output_a} 10.0
  ```

- **-ref** Defines the clock name and edge that controls the event. Value must be one of the following:

  - `r` - rising edge
  - `f` - falling edge

  For example: `define_output_delay {portb[7:0]} 10.00 -ref clock2:f`

- **-route** An advanced option, which includes route delay when the Synplify synthesis tool tries to meet the clock frequency goal. See *Example: -route option on page 6-30* for an example of its use with an input register.
The slack time reported on an instance driving an output port is the amount of time by which the clock frequency goal is exceeded (positive slack), or not met (negative slack) because of the output delay. You can view slack times with HDL Analyst or in the timing report section of the log file.

**define_multicycle_path**

Specifies a path that is a timing exception because it uses multiple clock cycles. This constraint provides extra clock cycles to the designated paths for timing analysis and optimization. For information about the equivalent SCOPE interface, see *Multi-Cycle Paths Panel on page 6-12.*

Here is the syntax:

```plaintext
define_multicycle_path {-from reg | input} {-to reg | output} {-through net} clock_cycles -comments
```

- **-disable** Disables a previous delay specification on the named port.
- **-from** Specifies the starting point for the multicycle timing exception. From points can be registers, top-level input, or bidirectional ports. For more information, see *Defining Paths/Points for Timing Exceptions on page 6-15.* You can combine this option with -to or -through to get a specific path.
- **-to** Specifies the ending point for the timing exception. To points can be registers, top-level output, or bidirectional ports. For more information, see *Defining Paths/Points for Timing Exceptions on page 6-15.* You can combine this option with -from or -through to get a specific path.
- **-through** Specifies the intermediate points for the timing exception. You can use any nets in the design as intermediate points. The intermediate points can be specified as a space-separated list, which is treated as an OR list. The exception is applied to all paths that cross any instance in the list. See *Defining Paths/Points for Timing Exceptions on page 6-15.* You can combine this option with -to or -from to get a specific path. To keep the signal name intact throughout synthesis when you this option, set the **syn_keep** directive (Verilog or VHDL) on the signal.
- **clock_cycles** The number of clock cycles of the start clock to be used for this path.
Here are some examples of the `define_multicycle_path` constraint syntax:

```plaintext
define_multicycle_path -from{i:regs.addr[4:0]} -to{i:special_regs.w[7:0]} 2

define_multicycle_path -to {i:special_regs.inst[11:0]} 2

define_multicycle_path -from {p:porta[7:0]} -through {n:prgmcntr.pc_sel44[0]} -to {p:portc[7:0]} 2
```

The following case illustrates where you might use a multicycle timing exception. Your design has a multiplier that multiplies `signal_a` with `signal_b` and puts the result into `signal_c`. Assume `signal_a` and `signal_b` are outputs of registers `register_a` and `register_b`, respectively. On clock cycle 1, a state machine enables an input enable signal to load `signal_a` into `register_a` and `signal_b` into `register_b`. At the beginning of clock cycle 2, the multiply begins. After two clock cycles, the state machine enables an output enable signal on clock cycle 3 to load the result of the multiplication (`signal_c`) into an output register (`register_c`).

The design frequency goal is 50 MHz (20 ns) and the multiply function takes 35 ns, but it is given 2 clock cycles. After optimization, this 35 ns path would normally be reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To avoid reporting the paths as timing violations or impacting optimization, you would include the following in the timing constraint file:

```plaintext
# Paths from register_a use 2 clock cycles
define_multicycle_path -from register_a 2

# Paths from register_b use 2 clock cycles
define_multicycle_path -from register_b 2
```

Alternatively, you can specify that the paths to the output register use multiple clock cycles:

```plaintext
# Paths to register_c use 2 clock cycles
define_multicycle_path -to register_c 2
```
Define false path

Defines paths to ignore (remove) during timing analysis and assign lower (or no) priority during optimization. The false paths are also passed on to supported place-and-route tools. For information about the equivalent SCOPE interface, see False Paths Panel on page 6-14.

Here is the syntax:

```
define_false_path {-from reg | input | to reg | output | through reg} -comments
```

- **-disable** Disables a previous delay specification on the named port.
- **-from** Specifies the starting point for the false path. From points can be: registers, top-level input, or bidirectional ports. For more information, see Defining Paths/Points for Timing Exceptions on page 6-15. You can combine this option with -to or -through to get a specific path.
- **-to** Specifies the ending point for the false path. To points can be: registers, top-level output or bidirectional ports. For more information, see Defining Paths/Points for Timing Exceptions on page 6-15. You can combine this option with -from or -through to get a specific path.
- **-through** Specifies the intermediate points for the timing exception. You can use any nets in the design as intermediate points. The intermediate points can be specified as a space-separated list, which is treated as an OR list. The exception is applied to all paths that cross any instance in the list. See Defining Paths/Points for Timing Exceptions on page 6-15. You can combine this option with -to or -from to get a specific path. To keep the signal name intact throughout synthesis when you this option, set the syn_keep directive (Verilog or VHDL) on the signal.

The following case uses a define_false_path constraint. Your design frequency goal is 50 MHz (20ns), and the path from register_a to register_c is a false path with a large delay of 35 ns. After optimization, this 35 ns path would normally be reported as a timing violation because it is more than the 20 ns clock-cycle timing goal.

To lower the priority of this path during optimization, define it as a false path. If all paths from register_a to any register or output pins are not timing-critical, then add the following line to the timing constraint file:

```
#Paths from register_a are ignored
define_false_path -from register_a
```
If all paths to register_c are not timing-critical, then add the following line in the timing constraint file:

```plaintext
#Paths to register_c are ignored
define_false_path -to register_c
```

**syn_reference_clock**

This is an attribute that specifies a clock frequency other than that implied by the signal on the clock pin of the register. For example, when flip-flops have an enable with a regular pattern, such as every second clock cycle, use `syn_reference_clock` to have the timing analyzer treat the flip-flops as if they were connected to a clock at half the frequency. You can also set this attribute from the SCOPE spreadsheet Attributes panel, as described in *Specifying Attributes with the SCOPE Spreadsheet on page 7-9*, or in the source code.

Here is the syntax:

```plaintext
define_attribute register syn_reference_clock clock
```

To use `syn_reference_clock`, define a new clock using the `define_clock` constraint. Name the clock, then apply the name of the new clock to the registers you want to change.

This example show how to use `syn_reference_clock`:

```plaintext
define_clock dummy -period 40.0 -virtual
define_attribute myreg[31:0] syn_reference_clock dummy
```

This next example applies `syn_reference_clock` to all registers enabled by signal `en40`.

```plaintext
define_attribute {find -reg -enable en40} syn_reference_clock clk2
```
Black-Box Timing Models

Black-box Source Code Directives

Black-box timing models are specific to components that you have instantiated (or declared) as black boxes using `syn_black_box`. You specify the models with source code directives.

You must put the directives in the source code because the models are specific to individual instances. There are no corresponding Tcl directives available for use in a constraint file. The following are the attributes you use with `syn_black_box` to characterize black-box timing:

- `syn_tpd`\(n\) – timing propagation for combinational delay through the black box. See `syn_tpd<n>` on page 7-180 for details.
- `syn_tsu`\(n\) – timing setup delay required for input pins relative to the clock. See `syn_tsu<n>` on page 7-184 for details.
- `syn_tco`\(n\) – timing clock to output delay through the black box, where \(n\) is an integer from 1 through 10, inclusive. See `syn_tco<n>` on page 7-177 for details.

There are 10 instances of each of these attributes available, for example: `syn_tpd1`, `syn_tpd2`, `syn_tpd3`, ... `syn_tpd10`. If you need more than 10, you can declare the desired amount (start with an integer greater than 10), for example:

```
attribute syn_tpd11 : string;
attribute syn_tpd12 : string;
```

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are in ns.

```plaintext
For syn_tpd: "bundle -> bundle = value "
For syn_tsu: "bundle -> [!]clock = value "
For syn_tco: [!]clock -> bundle = value "
```

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is `A,B,C` which lists three signals.
**VHDL Example**

In addition to the syntax used in the code below, you can also use the following Verilog-style syntax to specify the black-box attributes:

```vhdl
    attribute syn_tco1 of inputfifo_coregen : component is 
        "rd_clk->dout[48:0]=3.0";
```

The following is an example of a VHDL black box:

```vhdl
-- A USE clause for the synthesis tool Attributes
-- package was included earlier to make the attribute
definitions visible here.
architecture top of top is
component rcf16x4z port ( 
    ado, ad1, ad2, ad3 : in std_logic;
    dio, 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_tco1 of rcf16x4z : component is 
    "clk -> do0,do1 = 4.0";
attribute syn_tpd1 of rcf16x4z : component is 
    "ado,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 
    "ado,ad1,ad2,ad3 -> ck = 1.2";
attribute syn_tsu2 of rcf16x4z : component is 
    "wren,wpe -> ck = 0.0";

-- Other code
```
Verilog Example

module ram32x4(z, d, addr, we, clk);
/* synthesis black_box
   syn_tcol="clk->z[3:0]=4.0"
   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
Forward Annotation

The Synplify synthesis tool generates vendor-specific constraint files that can be forward-annotated (read in and used) with the place-and-route tools. The constraint files are generated by default. To disable this feature, go to the Implementation Results panel of the Options for implementation dialog box (Project -> Implementation Options), and disable the Write Vendor Constraint File option. Constraint files for these vendors are forward-annotated: Lattice, Altera and Xilinx.

Altera Max+Plus II

The constraint file generated for Altera Max+Plus II place-and-route tools has a “acf” file extension (.acf). The following constraints are forward-annotated to Altera Max+Plus II in this file:

• define_clock
• define_input_delay
• define_output_delay

Altera Quartus II

The constraint file generated for Quartus II place-and-route tools has a “tcl” file extension (.tcl). The following constraints are forward-annotated to Quartus II in this file for APEX:

• define_clock
• define_false_path
• define_input_delay
• define_multicycle_path
• define_output_delay

In addition to these constraints, the Synplify synthesis tool forward-annotates relationships between different clocks.
In designs to be sent to Quartus II, the synthesis tool forward-annotates the `define_input_delay` and `define_output_delay` timing constraints to Quartus II if you use the `syn_forward_io_constraints` attribute. A value of 1 enables forward annotation, while a value of 0 (default behavior) disables forward annotation. Use this attribute on the top level of a VHDL or Verilog file, or place it on the global object in the Attributes panel in the SCOPE spreadsheet.

```plaintext
syn_forward_io_constraints = { 1 | 0 }
```

**Verilog Example**

```verilog
module top (a,b,c) /*synthesis syn_forward_io_constraints = 1 */;
    input a, b;
    output c;
    assign c = a | b;
endmodule
```

**VHDL Example**

```vhdl
library Ieee;
use ieee.std_logic_1164.all;
entity top is port (
    a : in std_logic;
    b : in std_logic;
    o1 : out std_logic);
end top;

architecture archtop of top is
attribute syn_forward_io_constraints : boolean;
attribute syn_forward_io_constraints of archtop :
    architecture is true;
begin
    o1 <= a and b;
end archtop;
```

**SDC Example**

```vhdl
define_global_attribute syn_forward_io_constraints { 1 }
```
Lattice ORCA

The constraint file generated for Lattice's ORCA Foundry place-and-route tools has a "lp" file extension (.lp). The Lattice ORCA flow requires that you copy the .lp file containing the constraints from the Synplify synthesis tool or a text editor into Lattice ORCA's .prf file.

Follow these steps to forward-annotate your synthesis constraint file to ORCA Foundry:

1. Set your constraints in the SCOPE spreadsheet (see SCOPE Spreadsheet on page 6-3).
2. Run your design. The synthesis tool creates an .lp file in the same directory as your result files.
3. Open Lattice's ORCA Foundry place-and-route tool. Run the Map stage, which creates the .prf file.
4. From any text editor, copy the .lp file and place it at the bottom of the .prf file. Do not copy over the .prf file with the .lp file.
5. Run the PAR and BIT stages in Foundry.

The following constraints are forward-annotated to ORCA Foundry in this file:

- define_clock
- define_input_delay
- define_multicycle_path
- define_output_delay
Xilinx

The constraint file generated for Xilinx place-and-route tools has an “ncf” file extension (.ncf). The following constraints are forward-annotated to Xilinx in this file:

- define_clock
- define_false_path
- define_input_delay
- define_multicycle_path
- define_output_delay

In addition to these constraints, the Synplify synthesis tool forward-annotates relationships between different clocks.

Forward-annotating I/O Timing Constraints

The Synplify synthesis tool forward-annotates the define_input_delay and define_output_delay constraints to the Xilinx .ncf file. The syn_forward_io_constraints attribute controls forward annotation. An attribute value of 1 (default behavior) enables forward annotation, while a value of 0 disables forward annotation. Use this attribute at the top level of a VHDL or Verilog file, or place it on the global object in the Attributes panel of the SCOPE spreadsheet.

\[
syn\textunderscore forward\textunderscore io\textunderscore constraints = \{ 1 \mid 0 \}
\]

Verilog Example

```
module top (a,b,c) /* synthesis syn_forward_io_constraints = 1 */;
    input a, b;
    output c;
    assign c = a | b;
endmodule
```
VHDL Example

```vhdl
library Ieee;
use ieee.std_logic_1164.all;
entity top is port (
    a : in std_logic;
    b : in std_logic;
    o1 : out std_logic);
end top;

architecture archtop of top is
attribute syn_forward_io_constraints : boolean;
attribute syn_forward_io_constraints of archtop :
    architecture is true;
begin
    o1 <= a and b;
end archtop;
```

SDC Example

```vhdl
define_global_attribute syn_forward_io_constraints {1}
```

Relaxing Forward-annotated Constraints

You use the `xc_ncf_auto_relax` attribute to control the automatic relaxation of constraints that are forward-annotated to the `.ncf` file. A value of 0 disables this feature; the default value is 1.

The Synplify synthesis tool constrains input-to-register, register-to-register and register-to-output paths with the frequency constraint. However, if the period constraint is too tight for the `input-to-register` or `register-to-output` paths, the tool will try to relax the constraints to these paths. You can disable this relaxation by setting the `xc_ncf_auto_relax` attribute to 0 if your design requires it:

```vhdl
define_global_attribute xc_ncf_auto_relax {0}
```

Constraint File

```vhdl
define_global_attribute xc_ncf_auto_relax {0}
```
Verilog Example

module test_relax (clk, rst, a, b, out1)
    /* synthesis xc_ncf_auto_relax = 0 */;

    // Other code
endmodule

VHDL Example

library ieee;
use ieee.std_logic_1164.all;
entity test_relax is
    port(
        clk, rst, a, b : in std_logic;
        out1 : out std_logic
    );
end test_relax;

architecture behave of test_relax is
    attribute xc_ncf_auto_relax : boolean;
    attribute xc_ncf_auto_relax of behave : architecture is false;

    -- Other code
end behave;
Support for DCM/DLL in Xilinx

The Synplify synthesis tool is able to take advantage of the Frequency Synthesis and Phase Shifting features of Digital Clock Manager (DCM) and Delay Locked Loop (DLL) for Xilinx devices.

If you are using a DLL or DCM for on-chip clock generation, you need only define the clock at the primary inputs. The synthesis tool will propagate clocks through any number of DLLs or DCMs. It will automatically generate clocks at the outputs of a DLL or DCM, as needed, taking into account any phase shift or frequency change.

You can specify the phase shift and frequency multiplication parameters using Xilinx standard properties like duty_cycle_correction and clkdv_divide, as well as properties like clkfx_multiply and clkfx_divide for DCMs if needed.

The synthesis tool also takes into account the fact that these clocks are related (synchronized) with each other, by putting them in the same clock-group.

However only the clock at the input of a DLL/DCM is forward-annotated in the .ncf file, because the backend tools understand the DLL and DCMs, and do their own clock propagation across them.
Chapter 6: Timing Constraints

Timing Report

The timing report is part of the log file (project_name.srr) in the results directory. You can find it by opening the log file (View -> Log File) after a synthesis run, and searching for “START TIMING REPORT”.

You can control the size of the timing report by choosing Project -> Implementation Options and clicking the Timing Report panel. This panel lets you specify the number of start/end points, and the number of critical paths to be reported.

The timing report consists of the following sections:

- Timing Report Header on page 6-46
- Performance Summary on page 6-47
- Clock Relationships on page 6-48
- Interface Information on page 6-48
- Detailed Clock Report on page 6-49

Timing Report Header

The timing report header lists the date and time, the name of the top-level module, the number of paths requested for the timing report, and the constraint files used.

```
00089  START TIMING REPORT
00090  # Timing Report written on Fri Jan 11 13:16:49 2002
00091  
00092  
00093  
00094  
00095  Top view: eight_bit_uc
00096  Slew propagation mode: worst
00097  Paths requested: 5
00098  Constraint File(s): D:\Designs\my_project.sdc
00099  
01000 | This timing report estimates place and route data. Please look
```

Figure 6-9: Timing Report header
Performance Summary

The performance summary section of the timing report reports clock information, with the clocks sorted by negative slack. You can set the number of critical paths reported by choosing Project -> Implementation Options and setting the number of paths you want on the Timing Report panel. The timing report has another section for detailed clock information (see *Detailed Clock Report on page 6-49*). The Performance Summary lists the following information for each clock in the design:

Table 6-10: Performance summary

<table>
<thead>
<tr>
<th>Performance Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting clock</td>
<td>The name of the clock. If the clock name is “none,” you</td>
</tr>
<tr>
<td>Clock group</td>
<td>The name of the clock group to which the clock belongs.</td>
</tr>
<tr>
<td>Requested Frequency</td>
<td>Target frequency goal.</td>
</tr>
<tr>
<td>Estimated Frequency</td>
<td>Estimated value after synthesis.</td>
</tr>
<tr>
<td>Requested Period</td>
<td>Target clock period.</td>
</tr>
<tr>
<td>Estimated Period</td>
<td>Estimated value after synthesis.</td>
</tr>
<tr>
<td>Slack</td>
<td>Difference between estimated and requested period.</td>
</tr>
</tbody>
</table>

Slack for starting clock listed as “none” is the worst slack for all paths in the none category. The none category is a collection of clocks with an undefined clock event. This can include clocks when you specify a global frequency.

| Clock type | Describes the type of clock. A clock can be an inferred, declared, derived or system clock. |

The Synplify synthesis tool does not report inferred clocks with an unreasonable slack time. You might also have a real clock with a negative period, for example, a clock going to only one flip-flop, which has only one path going to an output. If you specify a –1000 output delay on this output, the synthesis tool cannot calculate the clock frequency. It reports a negative period and no clock.
Clock Relationships

For each pair of clocks in the design, the Clock Relationships section of the timing report lists both the required time (constraint) and the worst slack time for each of the intervals rise to rise, fall to fall, rise to fall and fall to rise.

This information is provided for the paths between related clocks (that is, clocks in the same clock group). If there is no path at all between two clocks, then that pair is not reported. If there is no path for a given pair of edges between two clocks, then an entry of No paths appears.

For information about how these relationships are calculated, see Clock Groups on page 6-7.

Figure 6-10: Clock Relationships report

Interface Information

The interface section of the timing report contains information on arrival times, required times and slack for the top-level ports. It is divided into two subsections, one each for inputs and outputs. Bidirectional ports are listed under both. For each port, the interface report contains the following information.

Table 6-11: Interface report

<table>
<thead>
<tr>
<th>Port Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port Name</td>
<td>Port name.</td>
</tr>
<tr>
<td>Reference Clock</td>
<td>The reference clock.</td>
</tr>
<tr>
<td>User Constraint</td>
<td>The input/output delay. If a port has multiple delay records, the report contains the values for the record with the worst slack. The reference clock corresponds to the worst slack delay record.</td>
</tr>
</tbody>
</table>
Detailed Clock Report

Each clock reported in the performance summary also has a detailed clock report section in the timing report. The clocks are listed by negative slack.

Table 6-12: Detailed clock report

<table>
<thead>
<tr>
<th>Clock Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ most critical start points</td>
<td>Starting points can be input ports or registers. If the starting point is a register, you see the starting pin in the report. To change the number of start points reported, choose <strong>Project -&gt; Implementation Options</strong>, and set the number on the <strong>Timing Report</strong> panel.</td>
</tr>
<tr>
<td>$N$ most critical end points</td>
<td>End points can be output ports or registers. If the end point is a register, you see the ending pin in the report. To change the number of start points reported, select <strong>Project -&gt; Implementation Options</strong>, and set the number on the <strong>Timing Report</strong> panel.</td>
</tr>
<tr>
<td>$N$ critical path tables (see the next table for details)</td>
<td>You can change the number of critical paths on the <strong>Timing Report</strong> panel of the <strong>Implementation Options</strong> dialog box. Paths from clock A to clock B are reported as critical paths in the section for clock A.</td>
</tr>
</tbody>
</table>

For each critical path, the Synplify synthesis tool uses this report format: Output pin—Input pin—Output pin—Input pin, without including the intervening nets. The following table describes the critical path information reported:
Table 6-13: Critical path reports

<table>
<thead>
<tr>
<th>Critical Path Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance/Net Name</td>
<td>Technology view names for the instances and nets in the critical path</td>
</tr>
<tr>
<td>Type</td>
<td>Type of cell</td>
</tr>
<tr>
<td>Pin Name</td>
<td>Name of the pin</td>
</tr>
<tr>
<td>Pin Dir</td>
<td>Pin direction</td>
</tr>
<tr>
<td>Arrival Time</td>
<td>Clock delay at the source + the propagation delay through the path</td>
</tr>
<tr>
<td>Delay</td>
<td>The delay value</td>
</tr>
<tr>
<td>Arrival Time</td>
<td>Arrival time of the signal</td>
</tr>
<tr>
<td>Fanout</td>
<td>Number of fanouts for the point in the path</td>
</tr>
</tbody>
</table>
CHAPTER 7

Synthesis Attributes and Directives

This chapter shows you how to specify attributes and directives, and contains individual attribute and directive descriptions, syntax definitions, and examples.

These are the chapter topics:

- Attributes and Directives: Summary on page 7-2
- Specifying Attributes in a Constraint File on page 7-9
- Specifying Directives (and Attributes) in HDL on page 7-14
- Attributes on page 7-18
- Directives on page 7-139
Attributes and Directives: Summary

The following sections summarize the synthesis attributes and directives:

- Attributes and Directives: Differences, Specifying on page 7-2
- Attribute and Directive Summary by Vendor on page 7-3
- Attribute Summary (Alphabetical) on page 7-3
- Directive Summary (Alphabetical) on page 7-7

For detailed descriptions of individual attributes and directives, see Attributes on page 7-18 and Directives on page 7-139, respectively.

Attributes and Directives: Differences, Specifying

Synthesis attributes and directives let you direct the way a design is analyzed, optimized, and mapped during synthesis. For example, you can specify how you want the Synplify synthesis tool to manage hierarchy during optimization or which objects to treat as black boxes.

Attributes control mapping optimizations.
Directives control compiler optimizations.

Because of this difference, directives must be entered in the HDL source code, but attributes can be entered either in an .sdc constraint file (recommended) or in HDL source code.

If you specify attributes in source code, whenever you change them you must recompile the design. Recompilation can be very time consuming for large designs. If you use a constraint file instead, you can easily change attributes at any time, without compiling. The preferred way to specify attributes is using the SCOPE spreadsheet (see SCOPE Window on page 6-3), which automatically generates and edits an .sdc constraint file.

For information on specifying attributes in constraint files, in particular via the SCOPE spreadsheet, see Specifying Attributes in a Constraint File on page 7-9.

For information on specifying directives (and attributes) in the HDL source code, see Specifying Directives (and Attributes) in HDL on page 7-14. Remember, though, that it is generally best to specify attributes through the SCOPE spreadsheet.
Attribute and Directive Summary by Vendor

For a list of vendor-specific attributes and directives, see the appropriate vendor chapters, as shown in the following table. For alphabetical summaries of attributes and directives, see Attribute Summary (Alphabetical) on page 7-3 and Directive Summary (Alphabetical) on page 7-7, respectively.

Table 7-1: Vendor-specific attributes and directives

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Attributes and Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actel</td>
<td>Actel Attribute and Directive Summary on page A-10</td>
</tr>
<tr>
<td>Altera</td>
<td>Altera Attribute and Directive Summary on page B-36</td>
</tr>
<tr>
<td>Atmel</td>
<td>Atmel Attribute and Directive Summary on page C-5</td>
</tr>
<tr>
<td>Cypress</td>
<td>Cypress Attribute and Directive Summary on page D-12</td>
</tr>
<tr>
<td>Lattice</td>
<td>Lattice Attribute and Directive Summary on page E-16</td>
</tr>
<tr>
<td>Quicklogic</td>
<td>QuickLogic Attribute and Directive Summary on page F-7</td>
</tr>
<tr>
<td>Triscend</td>
<td>Triscend Attribute and Directive Summary on page G-4</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Xilinx Attribute and Directive Summary on page H-43</td>
</tr>
</tbody>
</table>

Attribute Summary (Alphabetical)

The following table summarizes the synthesis attributes. For details on individual attributes, see the alphabetized list in section Attributes on page 7-18. For a summary of the directives, see Directive Summary (Alphabetical) on page 7-7.

Table 7-2: Attributes, alphabetical summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alsloc</td>
<td>Preserves relative placement in Actel designs.</td>
</tr>
<tr>
<td>alspin</td>
<td>Assigns scalar or bus ports to I/O pin numbers in Actel designs.</td>
</tr>
<tr>
<td>alspreserve</td>
<td>Specifies nets that must be preserved by the Actel place-and-route tool.</td>
</tr>
</tbody>
</table>
### Table 7-2: Attributes, alphabetical summary (Continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altera_auto_use_eab</td>
<td>Automatically implements large RAMs and ROMs as extended array blocks (EABs) for FLEX10K and ACEX, and MAX designs.</td>
</tr>
<tr>
<td>altera_auto_use_esb</td>
<td>Automatically implements large RAMs and ROMs as PTERMs in embedded system blocks (ESBs) for APEX20K/20KC/20KE, APEX II and Mercury designs.</td>
</tr>
<tr>
<td>altera_chip_pin_lc</td>
<td>Specifies pin locations for Altera I/Os and forward-annotates information to place-and-route tool.</td>
</tr>
<tr>
<td>altera_implement_in_eab</td>
<td>Implements a logic design unit as extended array blocks (EABs) for FLEX10K and ACEX designs.</td>
</tr>
<tr>
<td>altera_implement_in_esb</td>
<td>Implements a logic design unit as a PTERM in an extended system block (ESB) in APEX, APEX II, Excalibur and Mercury designs.</td>
</tr>
<tr>
<td>altera_io_opendrain</td>
<td>Specifies open drain mode for Altera output and bidirectional ports.</td>
</tr>
<tr>
<td>altera_io_powerup</td>
<td>Controls the power up mode for Altera I/O registers.</td>
</tr>
<tr>
<td>din</td>
<td></td>
</tr>
<tr>
<td>dout</td>
<td></td>
</tr>
<tr>
<td>loc</td>
<td>Specifies I/O pin locations for Lattice ORCA designs.</td>
</tr>
<tr>
<td>lock</td>
<td>Specifies I/O pin locations for Lattice designs.</td>
</tr>
<tr>
<td>orca_padtype</td>
<td>Specifies I/O pin locations for Lattice ORCA designs.</td>
</tr>
<tr>
<td>orca_props</td>
<td>Specifies Lattice ORCA I/O properties for forward annotation.</td>
</tr>
<tr>
<td>ql_padtype</td>
<td>Specifies I/O pin locations for Quicklogic designs.</td>
</tr>
<tr>
<td>ql_placement</td>
<td>Marks I/O pad locations for forward annotation (Quicklogic).</td>
</tr>
</tbody>
</table>
Table 7-2: Attributes, alphabetical summary (Continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_direct_enable</td>
<td>Assigns clock enable nets to dedicated flip-flop enable pins. It can also be used as a compiler directive that marks flip-flops with clock enables for inference.</td>
</tr>
<tr>
<td>syn_edif_bit_format</td>
<td>Formats bus names and delimiters in a Xilinx output EDIF file.</td>
</tr>
<tr>
<td>syn_edif_name_length</td>
<td>Determines the length of port names in an Altera output EDIF file.</td>
</tr>
<tr>
<td>syn_edif_scalar_format</td>
<td>Formats scalar names in a Xilinx output EDIF file.</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_forward_io_constraints</td>
<td>Controls forward annotation of I/O timing constraints.</td>
</tr>
<tr>
<td>syn_hier</td>
<td>Determines hierarchical control across module or component boundaries.</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>Sets maximum fanout for individual nets or registers, overriding the default.</td>
</tr>
<tr>
<td>syn_multstyle</td>
<td>Determines whether multipliers are implemented in logic or hardware blocks.</td>
</tr>
<tr>
<td>syn_netlist_hierarchy</td>
<td>Controls hierarchy generation in EDIF output files</td>
</tr>
<tr>
<td>syn_noarrayports</td>
<td>Specifies ports as individual signals or bus arrays.</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Disables automatic clock buffer insertion.</td>
</tr>
<tr>
<td>syn_preserve_sr_priority</td>
<td>Forces set/resets to observe priority (Actel).</td>
</tr>
<tr>
<td>syn_props</td>
<td>Specifies Triscend attributes for forward annotation</td>
</tr>
<tr>
<td>syn_radhardlevel</td>
<td>Specifies radiation-resistant design technique (Actel).</td>
</tr>
<tr>
<td>syn_ramstyle</td>
<td>Determines how RAMs are implemented.</td>
</tr>
<tr>
<td>syn_reference_clock</td>
<td>Specifies a clock frequency other than that implied by the signal on the clock pin of the register.</td>
</tr>
<tr>
<td>syn_replicate</td>
<td>Controls replication in Xilinx devices.</td>
</tr>
</tbody>
</table>
### Table 7-2: Attributes, alphabetical summary (Continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>syn_resources</strong></td>
<td>Specifies resources for black boxes.</td>
</tr>
<tr>
<td><strong>syn_romstyle (Altera)</strong>, <strong>syn_romstyle (Xilinx)</strong></td>
<td>Specifies how inferred ROMs should be implemented.</td>
</tr>
<tr>
<td><strong>syn_srlstyle</strong></td>
<td>Determines how sequential shift components are implemented (Xilinx).</td>
</tr>
<tr>
<td><strong>syn_tristatetomux</strong></td>
<td>Controls conversion of tristate buses to muxes.</td>
</tr>
<tr>
<td><strong>syn_useenable</strong></td>
<td>Generates clock enable pins for registers.</td>
</tr>
<tr>
<td><strong>syn_useioff (Altera)</strong>, <strong>syn_useioff (Lattice ORCA)</strong>, <strong>syn_useioff (QuickLogic)</strong>, <strong>syn_useioff (Xilinx)</strong></td>
<td>Controls register packing in I/Os.</td>
</tr>
<tr>
<td><strong>xc_alias</strong></td>
<td>Aliases names for XNF Writer (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_clockbuftype</strong></td>
<td>Specifies clock buffer type for clock ports.</td>
</tr>
<tr>
<td><strong>xc_fast</strong></td>
<td>Speeds up transition times for output drivers (Xilinx 4000).</td>
</tr>
<tr>
<td><strong>xc_fast_auto</strong></td>
<td>Determines whether fast or slow output buffers are inferred (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_global_buffers</strong></td>
<td>Determines the number of global buffers available (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_loc</strong></td>
<td>Specifies port locations (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_map</strong></td>
<td>Specifies primitive to use for relative location (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_ncf_auto_relax</strong></td>
<td>Relaxes I/O timing constraints (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_nodelay</strong></td>
<td>Controls insertion of default input delay (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_padtype</strong></td>
<td>Specifies I/O buffer standard (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_props</strong></td>
<td>Specifies Xilinx attributes for forward annotation (Xilinx).</td>
</tr>
<tr>
<td><strong>xc_pullup / xc_pulldown</strong></td>
<td>Defines pullup / pulldown ports (Xilinx).</td>
</tr>
</tbody>
</table>
Table 7-2: Attributes, alphabetical summary (Continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xc_rloc</td>
<td>Specifies the relative location of instances (Xilinx).</td>
</tr>
<tr>
<td>xc_slow</td>
<td>Slows down transition times for output drivers (Xilinx).</td>
</tr>
<tr>
<td>xc_uset</td>
<td>Specifies a group of instances for relative placement (Xilinx).</td>
</tr>
</tbody>
</table>

Table 7-3: Directives, alphabetical summary

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>black_box_pad_pin</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>parallel_case</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>syn_black_box</td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td>syn_direct_enable</td>
<td>When used as a compiler directive, this attribute marks the flip-flops with clock enables for the compiler to infer.</td>
</tr>
<tr>
<td>syn_enum_encoding</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
</tbody>
</table>
### Table 7-3: Directives, alphabetical summary (Continued)

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_isclock</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td>syn_keep</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td>syn_macro</td>
<td></td>
</tr>
<tr>
<td>syn_noprune</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td>syn_preserve (D)</td>
<td>Enables or prevents sequential optimizations, which eliminate redundant registers and registers with constant drivers.</td>
</tr>
<tr>
<td>syn_sharing</td>
<td>Enables/disables resource sharing of operators inside a module.</td>
</tr>
<tr>
<td>syn_state_machine</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td>syn_tco&lt;n&gt;</td>
<td>Defines timing clock to output delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_lpd&lt;n&gt;</td>
<td>Specifies timing propagation for combinational delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_tristate</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td>syn_tsu&lt;n&gt;</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>translate_off/translate_on</td>
<td>Specifies sections of code to exclude from synthesis, such as simulation-specific code.</td>
</tr>
<tr>
<td>xc_isgsr</td>
<td>Specifies black-box ports that are connected to internal startup blocks (Xilinx).</td>
</tr>
</tbody>
</table>
# Specifying Attributes in a Constraint File

Attributes direct the way a design is optimized during the mapping phase of synthesis. You can specify attributes in either HDL source code (see *Specifying Directives (and Attributes) in HDL* on page 7-14) or a constraint file (.sdc). A constraint file can be either generated automatically by the SCOPE spreadsheet or typed in manually using a text editor.

**Note:** If you specify attributes in source code, whenever you change them you must recompile the design. Recompilation can be very time consuming for large designs. If you use a constraint file, you can easily change attributes at any time, without compiling.

---

## Specifying Attributes with the SCOPE Spreadsheet

Perhaps the best way to specify attributes is with the SCOPE spreadsheet (see *SCOPE Window* on page 6-3). This automatically generates and updates a constraint file (.sdc) that specifies the attributes.

To use the SCOPE spreadsheet to enter attributes, start with a compiled design. This allows the tool to access the objects and values from the design. Then, click the SCOPE toolbar icon ( ). When the SCOPE spreadsheet opens, click the Attributes tab. Use this tab's panel to specify attributes either directly in the fields or through the wizard, described in *SCOPE Attributes Wizard* on page 7-12. The Attributes panel fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>(Required.) Turn this on to enable the constraint.</td>
</tr>
<tr>
<td>Object Type</td>
<td>Specifies the type of object to which the attribute is assigned. Choose from the pulldown list, to filter the available choices in the Object field.</td>
</tr>
<tr>
<td>Object</td>
<td>(Required.) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field.</td>
</tr>
</tbody>
</table>
Enter the appropriate attributes and corresponding values, by setting the attributes manually, clicking in the field and choosing from the pulldown menus, dragging and dropping objects from the RTL or Technology views into the Object field, or by using the wizard. For more information about the Attributes wizard, see SCOPE Attributes Wizard on page 7-12. When you have entered the attributes, save the file and add it to your project. This creates a constraint file with an “sdc” file extension (.sdc).

Table 7-4: SCOPE Attribute fields (Continued)

| Attribute | (Required.) Specifies the attribute name. You can choose from a pulldown list that includes all available attributes for the specified technology. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the Synplify synthesis tool filters the available choices in the Object field. You must select an attribute before entering a value. |
| Value     | (Required.) Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate. |
| Val Type  | Specifies the kind of value for the attribute. For example, string or boolean. |
| Description | Contains a one-line description of the attribute. |
| Comment | Lets you enter notes comments about the attributes. |

Enter the appropriate attributes and corresponding values, by setting the attributes manually, clicking in the field and choosing from the pulldown menus, dragging and dropping objects from the RTL or Technology views into the Object field, or by using the wizard. For more information about the Attributes wizard, see SCOPE Attributes Wizard on page 7-12. When you have entered the attributes, save the file and add it to your project. This creates a constraint file with an “sdc” file extension (.sdc).
Add the constraint file to the source file list. Choose Project -> Implementation Options, and check that the file is selected on the Options/Constraints panel before you run synthesis. If you have more than one constraint file, select all those that apply to the implementation.

Figure 7-2: Specifying constraint files

**.sdc File Syntax**

When you use the SCOPE spreadsheet to create and modify the timing constraint files, the proper syntax is automatically generated. This is the syntax used to define attributes in the .sdc file.

```
define_attribute {object} attribute_name {value}
```

where:

- `object` is the appropriate design object, such as module, signal, input, instance, port, or wire name. The object naming syntax varies, depending on whether your source code is in Verilog or VHDL format. See *Object Naming Syntax on page 6-21* for details about the syntax conventions.
• **attribute_name** is the name of the synthesis attribute. This must be an attribute, not a directive, as directives are not supported in constraint files.

• **value** is either a string or Boolean (0 | 1).

**SCOPE Attributes Wizard**

The SCOPE Attributes wizard provides an easy interface to add attributes to design objects. The wizard consists of two dialog boxes.

To add an attribute using the SCOPE wizard, right-click in the SCOPE spreadsheet Attributes panel, then choose Insert Wizard from the popup menu.

![Insert Wizard Step 1 dialog box](image)

Figure 7-3: Insert Wizard Step 1 dialog box
In the first dialog box displayed, select the object to which you want to attach the attribute, using either the browser pane on the left or wildcards to filter your choices. Move your selection to the Selected column on the right by clicking the right arrow. Click Next to display the next dialog box.

![Figure 7-4: Insert Wizard Step 2 dialog box](image)

This next dialog box shows you the attributes for the object. You Enable or Disable the attribute, specify the Value you want for it, then click Finish. The SCOPE spreadsheet reflects your choices.

The following table shows you what to do for each SCOPE panel:
Specifying Directives (and Attributes) in HDL

Directives control the design analysis during the compilation phase of synthesis. Because they are required at the time a design is compiled, directives can only be specified in the HDL source code.

Attributes are read after a design is compiled, during the mapping phase of synthesis, and they can therefore be specified in either HDL source code or an .sdc constraint file. Specifying attributes in a constraint file provides the flexibility of changing them at any time without having to recompile the design and is the recommended method. See Specifying Attributes in a Constraint File on page 7-9 for more information.

The following sections describe how to specify directives and attributes in Verilog or VHDL source code.
Verilog Attribute and Directive Syntax

To define the directives (or attributes) that specify your synthesis preferences, attach them to the appropriate objects in the source code as regular Verilog or C-style comments. Each specification begins with the keyword `synthesis`. The directive or attribute value is either a string, placed within double quotes, or a Boolean integer (0 or 1). Directives, attributes, and their values, are case sensitive and are usually in lower case.

Here is the syntax using a regular Verilog comment:

```
// synthesis directive | attribute = "value"
```

This example shows how you would use the `syn_hier` attribute:

```
// synthesis syn_hier = "firm"
```

This example shows the `parallel_case` directive:

```
// synthesis parallel_case
```

This directive forces a multiplexed structure in Verilog designs and is implied to be true whenever you use it, which is why there is no associated value. Here is the syntax for the C-style comment:

```
/* synthesis directive | attribute = "value" */
```

This example shows how you use the `syn_hier` attribute in a C-style comment:

```
/* synthesis syn_hier = "firm" */
```

To specify more than one directive or attribute for a given design object, place them within the same comment, separated by whitespace. Do not use commas. Here is an example where the `syn_preserve` and `syn_implement` directives are specified in a single comment:

```
module radhard_dffrs(q,d,c,s,r)
    /* synthesis syn_preserve=1 syn_state_machine=0 */;
```

If you specify a directive or attribute using a `/* . . . */` C-style comment, you must place the comment before the statement’s semicolon. For example:

```
module bl_box(out, in) /* synthesis syn_black_box */ ;
```
VHDL Attribute and Directive Syntax

All of the Synplify synthesis tool directives and attributes that you can use to specify your synthesis preferences are predefined in the attributes package in the synthesis tool library.

You can either use the attributes package (recommended) or redeclare the types of directives and attributes each time you use them.

Using the attributes Package

Here is the syntax for using directives and attributes from the attributes package in your code:

```vhdl
library synplify;
use synplify.attributes.all;
-- design_unit_declarations
attribute synplify_attribute of object : object_type is value;
```

The attributes package can be found in the following location:

```
synplify_installation_dir/lib/vhd/synattr.vhd
```

The following is an example using syn_noclockbuf from the attributes package:

```vhdl
library synplify;
use synplify.attributes.all;
entity simpledff is
  port (q : out bit_vector(7 downto 0);
       d : in bit_vector(7 downto 0);
       clk : in bit);

// Note that no explicit type declaration is necessary
attribute syn_noclockbuf of clk : signal is true;

-- Other code
```
Without Using the attributes Package

Here is the syntax for explicitly defining directives and attributes in your code, without referencing the attributes package:

```vhdl
-- design_unit_declarations
attribute attribute_name: data_type;
attribute attribute_name of object: object_type is value;
```

Here is an example using the syn_noclockbuf attribute:

```vhdl
entity simpledff is
  port (q : out bit_vector(7 downto 0);
        d : in bit_vector(7 downto 0);
        clk : in bit);

  // Note the explicit type declaration
  attribute syn_noclockbuf : boolean;
  attribute syn_noclockbuf of clk : signal is true;

-- Other code
```
Attributes

The individual synthesis attributes are detailed in this section, in alphabetical order. Each attribute description includes the following:

- Technology support
- Detailed attribute description
- File syntax and examples for the following:
  - .sdc constraint file (the recommended method for defining attributes)
  - Verilog source code
  - VHDL source code

For an alphabetical summary of the attributes, see Attribute Summary (Alphabetical) on page 7-3. For a summary of the attributes by vendor, refer to the appropriate vendor chapter or see Attribute and Directive Summary by Vendor on page 7-3.

For general information about specifying attributes, see Specifying Attributes in a Constraint File on page 7-9.
**alsloc**

*Attribute; Actel (except 500K and PA).* Preserves relative placements of macros and IP blocks in the Actel Designer place-and-route tool. The alsloc attribute has no effect on synthesis, but is passed directly to Actel Designer.

**.sdc File Syntax and Example**

```verbatim
define_attribute {object} alsloc {location}
```

In the attribute syntax, `object` is the name of a macro or IP block and `location` is the row-column address of the macro or IP block.

Following is an example of setting alsloc on a macro (u1).

```verbatim
define_attribute {u1} alsloc {R15C6}
```

**Verilog Syntax and Example**

```verbatim
object /* synthesis alsloc = "location" */;
```

Where `object` is a macro or IP block and `location` is the row-column string. For example:

```verbatim
module test(in1, in2, in3, clk, q);
  input in1, in2, in3, clk;
  output q;
  wire out1 /* synthesis syn_keep = 1 */ , out2;
  and2a u1 (.A (in1), .B (in2), .Y (out1)) /* synthesis alsloc="R15C6" */;
  assign out2 = out1 & in3;
  df1 u2 (.D (out2), .CLK (clk), .Q (q)) /* synthesis alsloc="R35C6" */;
endmodule
```

```verbatim
module and2a(A, B, Y); // synthesis syn_black_box
  input A, B;
  output Y;
endmodule
```

module df1(D, CLK, Q); // synthesis syn_black_box
input D, CLK;
output Q;
endmodule

**VHDL Syntax and Example**

```vhdl
attribute alsloc of object : object_type is "location";
```

Where `object` is a macro or IP block, `object_type` is label, and `location` is the row-column string.

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

entity test is
  port(in1, in2, in3, clk : in std_logic;
       q : out std_logic);
end test;

architecture rtl of test is
  signal out1, out2 : std_logic;
  component AND2A
    port( A, B : in std_logic;
          Y : out std_logic);
  end component;
  component df1
    port(D, CLK : in std_logic;
         Q : out std_logic);
  end component;
  attribute syn_keep : boolean;
  attribute syn_keep of out1 : signal is true;
  attribute alsloc: string;
  attribute alsloc of U1: label is "R15C6";
  attribute alsloc of U2: label is "R35C6";
  attribute syn_black_box : boolean;
  attribute syn_black_box of AND2A, df1 : component is true;
begin
  U1: AND2A port map (A => in1, B => in2, Y => out1);
  out2 <= in3 and out1;
  U2: df1 port map (D => out2, CLK => clk, Q => q);
end rtl;
```


### alspin

*Attribute: Actel (except 500K and PA).* The *alspin* attribute assigns the scalar or bus ports of the design to Actel I/O pin numbers (pad locations). Refer to the Actel databook for valid pin numbers. If you want to use *alspin* for bus ports or for slices of bus ports, you must also use the *syn_noarrayports* attribute. See **.sdc File Syntax and Example** below for information on assigning pin numbers to buses and slices.

#### .sdc File Syntax and Example

```vhdl
define_attribute {port_name} alspin {pin_number}
```

In the attribute syntax, *port_name* is the name of the port and *pin_number* is the Actel I/O pin.

```vhdl
define_attribute {DATAOUT} alspin {48}
```

If you want to assign pin numbers to a bus port, or to a single or multiple bit slice of a bus port, edit the constraint file as follows:

1. Specify the *syn_noarrayports* attribute to your design to “bit blast” all bus ports.
2. Specify the *alspin* attribute for the individual bits of the bus.

The following example illustrates these steps.

```vhdl
define_attribute {ADDRESS0} syn_noarrayports {1};
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}
```

#### Verilog Syntax and Example

```verilog
object/* synthesis alspin = "pin_number" */;
```

Where *object* is the port and *pin_number* is the Actel I/O pin. For example:

```verilog
object/* synthesis alspin = "pin_number" */;
```
VHDL Syntax and Example

```
attribute alspin of object : object_type is "pin_number";
```

Where `object` is the port, `object_type` is signal, and `pin_number` is the Actel I/O pin. For example:

```
entity comparator is
  port (datain : in bit_vector(7 downto 0);
         clk : in bit;
         dataout : out bit);
attribute alspin : string;
attribute alspin of dataout : signal is "48";
-- Other code
```
alspreserve

Attribute; Actel (except 500K and PA). Specifies a net that should not be removed (optimized away) by the Actel Designer place-and-route tool. The alspreserve attribute has no effect on synthesis, but is passed directly to the Actel Designer place-and-route software. However, to prevent the net from being removed during the synthesis process, you must also use the syn_keep directive.

.sdc File Syntax and Example

```
define_attribute {n:net_name} alspreserve {1}
```

In the attribute syntax, `net_name` is the name of the net to be preserved.

```
define_attribute {n:and_out3} alspreserve {1};
define_attribute {n:or_out1} alspreserve {1};
```

Verilog Syntax and Example

```
oBJECT /* synthesis alspreserve = 1 */;
```

Where `object` is the name of the net to preserve. For example:

```
module complex (in1, out1);
input [6:1] in1;oh
output out1;
wire out1;
wire or_oosut1 /* synthesis syn_keep=1 alspreserve=1 */;
wire and_out1;
wire and_out2;
wire and_out3 /* synthesis syn_keep=1 alspreserve=1 */;
assign and_out1 = in1[1] & in1[2];
assign and_out2 = in1[3] & in1[4];
assign and_out3 = in1[5] & in1[6];
assign or_out1 = and_out1 | and_out2;
assign out1 = or_out1 & and_out3;
endmodule
```

VHDL Syntax and Example

```
attribute alspreserve of object : object_type is true;
```

Where `object` is the name of the net to be preserved and `object_type` is signal.
For example:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;
entity complex is
    port (input : in std_logic_vector (6 downto 1);
           output : out std_logic);
end complex;

architecture RTL of complex is
    signal and_out1 : std_logic;
    signal and_out2 : std_logic;
    signal and_out3 : std_logic;
    signal or_out1 : std_logic;
    attribute syn_keep of and_out3 : signal is true;
    attribute syn_keep of or_out1 : signal is true;
    attribute alspreserve of and_out3 : signal is true;
    attribute alspreserve of or_out1 : signal is true;
begin
    and_out1 <= input(1) and input(2);
    and_out2 <= input(3) and input(4);
    and_out3 <= input(5) and input(6);
    or_out1 <= and_out1 or and_out2;
    output <= or_out1 and and_out3;
end;
```
**altera_auto_use_eab**

*Attribute. Altera (FLEX and ACEX families).* Automatically implements large inferred RAMs and ROMs as embedded block arrays (EABs) for FLEX10K and ACEX, and MAX designs. Altera mappers automatically use EAB resources for inferred RAMs and ROMs. However, a potential for the mapper to overuse these resources is possible since the mapper is not aware of resources used whenever you specify the `altera_implement_in_eab` attribute or resources used through black boxes or instantiated LPMs. When you set `altera_auto_use_eab` to 0 or false, inferred RAMs and ROMs are not implemented in ESBs.

You can override this attribute for specific RAMs and/or ROMs. To do this for RAMs, set the `syn_ramstyle` attribute to `block_ram` and apply it to a specific RAM. Likewise, for ROMs, set the `syn_romstyle` to `block_rom` and apply it to a specific ROM.

The usage summary in the `.srr` file shows the number of EAB resources used, the number available, and percent utilization.

**.sdc File Syntax and Example**

```
define_global_attribute altera_auto_use_eab {0 | 1}
```

For example:

```
define_global_attribute altera_auto_use_eab {0}
```

**Verilog Syntax and Example**

```
object /* synthesis altera_auto_use_eab = 0 | 1 */ ;
```

For example:

```
module rom2(z, a);
output [3:0] z;
input [4:0] a;
reg [3:0] z;
always @(a)
begina
    case (a)
        5'b00000: z = 4'b1011;
        5'b00001: z = 4'b0001;
        5'b00100: z = 4'b0011;
        5'b00110: z = 4'b0010;
    endcase
end
```

---

Synplify Reference Manual, April 2002 7-25
5'b00111: z = 4'b1110;
5'b01001: z = 4'b0111;
5'b01010: z = 4'b0101;
5'b01101: z = 4'b0100;
5'b10000: z = 4'b1100;
5'b10001: z = 4'b1101;
5'b10010: z = 4'b1111;
5'b10011: z = 4'b1110;
5'b11000: z = 4'b1010;
5'b11010: z = 4'b1011;
5'b11110: z = 4'b1001;
5'b11111: z = 4'b1000;
default: z = 4'b0000;
endcase
end
endmodule

module top (z, a) /* synthesis altera_auto_use_eab = 0 */;
output [3:0] z;
input [4:0] a;
rom2 myrom (.z(z), .a(a));
endmodule

VHDL Syntax and Example

attribute altera_auto_use_eab of object : object_type is false | true ;

For example:

library IEEE;
use IEEE.std_logic_1164.all;
entity rom6 is
port ( a: in std_logic_vector(3 downto 0);
       z: out std_logic_vector(3 downto 0) );
end rom6;
architecture rtl of rom6 is
begin
process(a)
begin
case a is
when "0000" => z <= "1010";
when "0001" => z <= "0100";
when "0010" => z <= "0111";
when "0011" => z <= "0010";
when "0100" => z <= "0101";
library IEEE;
use IEEE.std_logic_1164.all;
entity top is port (
a: in std_logic_vector(3 downto 0);
z: out std_logic_vector(3 downto 0)
);
end top;
architecture archtop of top is
component rom6 is port (a: in std_logic_vector(3 downto 0);
z: out std_logic_vector(3 downto 0)
);
end component;
attribute altera_auto_use_eab : boolean;
attribute altera_auto_use_eab of archtop : architecture is false;
begin
U1: rom6 port map (a => a,
z => z);
end archtop;
altera_auto_use_esb

Attribute. Altera (APEX families). Automatically implements large, inferred RAMs and ROMs as PTERMs in embedded system blocks (ESBs) for APEX20K/20KC/20KE, APEX II and Mercury designs. Altera mappers automatically use ESB resources for inferred RAMs and ROMs. However, a potential for the mapper to overuse these resources is possible since the mapper is not aware of resources used whenever you specify the altera_implement_in_esb attribute or resources used through black boxes or instantiated LPMs. When you set altera_auto_use_esb to 0 or false, inferred RAMs and ROMs are not implemented in ESBs.

You can override this attribute for specific RAMs and/or ROMs. To do this for RAMs, set the syn_ramstyle attribute to block_ram and apply it to a specific RAM. Likewise, for ROMs, set the syn_romstyle to block_rom and apply it to a specific ROM.

The usage summary in the .srr file shows the number of ESB resources used, the number available, and percent utilization.

.sdc File Syntax and Example

```
define_global_attribute altera_auto_use_esb {0 | 1}
```

For example:

```
define_global_attribute altera_auto_use_esb {0}
```

Verilog Syntax and Example

```
object /* synthesis altera_auto_use_esb = 0 | 1 */;
```

For example:

```
module rom2(z, a);
output [3:0] z;
input [4:0] a;
reg [3:0] z;
always @(a)
begin
  case (a)
    5'b00000: z = 4'b1011;
    5'b00001: z = 4'b0001;
    5'b00100: z = 4'b0011;
    5'b00110: z = 4'b0010;
  endcase
end
```

Altera Auto Use ESB Attributes

7-28 Synplify Reference Manual, April 2002
5'b00111: z = 4'b1110;
5'b01001: z = 4'b0111;
5'b01010: z = 4'b0101;
5'b01101: z = 4'b0100;
5'b10000: z = 4'b1100;
5'b10001: z = 4'b1101;
5'b10010: z = 4'b1111;
5'b10011: z = 4'b1110;
5'b11000: z = 4'b1010;
5'b11010: z = 4'b1011;
5'b11100: z = 4'b1001;
5'b11101: z = 4'b1000;
default: z = 4'b0000;
endcase
endmodule

module top (z, a) /* synthesis altera_auto_use_esb = 0 */;
output [3:0] z;
inout [4:0] a;
rom2 myrom (.z(z), .a(a));
endmodule

VHDL Syntax and Example

attribute altera_auto_use_esb of object: object_type is false | true ;

For example:

library IEEE;
use IEEE.std_logic_1164.all;
entity rom6 is
port ( a: in std_logic_vector(3 downto 0);
       z: out std_logic_vector(3 downto 0) );
end rom6;
architecture rtl of rom6 is
begin
process(a)
begin
case a is
when "0000" => z <= "1010";
when "0001" => z <= "0100";
when "0010" => z <= "0111";
when "0011" => z <= "0010";
when "0100" => z <= "0101";
when "0101" => z <= "1001";
when "0110" => z <= "1011";
when "0111" => z <= "0011";
when "1000" => z <= "1101";
when "1001" => z <= "0001";
when "1010" => z <= "1111";
when "1011" => z <= "0110";
when "1100" => z <= "1000";
when "1101" => z <= "1110";
when "1110" => z <= "1100";
when others => z <= "0000";
end case;
end process;
end rtl;

library IEEE;
use IEEE.std_logic_1164.all;
entity top is port (  
a: in std_logic_vector(3 downto 0);
  z: out std_logic_vector(3 downto 0)
);
end top;
architecture archtop of top is  
component rom6 is port (  
a: in std_logic_vector(3 downto 0);
  z: out std_logic_vector(3 downto 0)
);
end component;
attribute altera_auto_use_esb : boolean;
attribute altera_auto_use_esb of archtop : architecture is false;
beginn
U1: rom6 port map (  
a =>a,
  z =>z
);
end archtop;
altera_chip_pin_lc

Attribute; Altera. Specifies pin locations for Altera I/Os and forward-annotates information to placement and routing tool. Refer to the Altera databook for valid pin numbers.

.sdc File Syntax and Examples

```verbatim
define_attribute {port_name} altera_chip_pin_lc {@pin_number}
```

The following examples show bus pin assignments for the APEX, ACEX, and FLEX technologies. Notice the APEX technology syntax does not use the @ sign prefix.

APEX Example

```verbatim
define_attribute {DATA0[7:0]} altera_chip_pin_lc {14,12,11,5,21,18,16,15}
```

ACEX and FLEX Example

```verbatim
define_attribute {DATA0[7:0]} altera_chip_pin_lc {@14,@12,@11,@5,@21,@18,@16,@15}
```

Verilog Syntax and Examples

```verbatim
object /* synthesis altera_chip_pin_lc = "@pin_number" */;
```

The following Verilog examples show bus pin assignment for the APEX, ACEX, and FLEX technologies. Notice the APEX technology syntax does not use the @ sign prefix.

APEX Example

```verbatim
output [7:0] DATA0
   /* synthesis altera_chip_pin_lc = "14,12,11,5,21,18,16,15" */;
```

ACEX, FLEX Example

```verbatim
output [7:0] DATA0
   /* synthesis altera_chip_pin_lc="@14,@12,@11,@5,@21,@18,@16,@15" */;
```
VHDL Syntax and Examples

attribute altera_chip_pin_lc of object : object_type is "@pin_number";

The following VHDL examples show bus pin assignment for both the APEX and the ACEX and FLEX technologies. Notice the APEX technology syntax does not use the @ sign prefix.

APEX Example

entity prep2_2 is
    port(data0: out std_logic_vector(7 downto 0);
    .
    .
    .
    qout: out std_logic_vector(7 downto 0));
attribute altera_chip_pin_lc of data0: signal is
    "14,12,11,5,21,18,16,15"
end prep2_2;

ACEX and FLEX Example

entity prep2_2 is
    port(data0: out std_logic_vector(7 downto 0);
    .
    .
    .
    qout: out std_logic_vector(7 downto 0));
attribute altera_chip_pin_lc of data0: signal is
    "@14,@12,@11,@5,@21,@18,@16,@15"
end prep2_2;
**altera_implement_in_eab**

Attribute; Altera (FLEX10K and ACEX). Implements a logic design unit as extended array blocks (EABs) for FLEX10K and ACEX designs.

**.sdc File Syntax and Example**

```plaintext
define_attribute {v:module_or_architecture_name} altera_implement_in_eab {1}
```

The following example implements the instance sqrt8 as an Altera EAB.

```plaintext
define_attribute {v:mymux} altera_implement_in_eab {1}
```

**Verilog Syntax and Example**

```plaintext
object /* synthesis altera_implement_in_eab = 1 */;
```

The following Verilog example implements an instance of a 2-to-1 multiplexer as an Altera EAB:

```plaintext
module mymux (in1 , sel, dout);
    input [9:0] in1;
    input sel;
    output [9:0] dout;
    assign dout = sel ? in1 : ~in1;
endmodule

define_attribute {v:mymux} altera_implement_in_eab = 1 */;
```

module esb_test (a, s, o);
    input [9:0] a;
    input s;
    output [9:0] o;
    mymux my (.in1(a), .sel(s), .dout(o)) /* synthesis altera_implement_in_eab = 1 */;
endmodule
VHDL Syntax and Example

attribute altera_implement_in_eab of object : object_type is 1 */ ;

The following example implements an instance of a 2-to-1 multiplexor as an Altera EAB:

```vhdl
module mymux (in1, sel, dout);
input [9:0] in1;
input sel;
output [9:0] dout;
assign dout = sel ? in1 : ~in1;
endmodule
module esb_test (a, s, o);
input [9:0] a;
input s;
output [9:0] o;
mymux my (.in1(a), .sel(s), .dout(o)) /* synthesis
altera_implement_in_eab = 1 */;
endmodule
```
altera_implement_in_esb

**Attribute:** Altera. Implements a logic design unit as a PTERM in an extended system block (ESB) in APEX, APEX II, Excalibur, and Mercury designs. You can apply this attribute to either a module or an instance. To use altera_implement_in_esb, make your design hierarchical. Then, instantiate the module or instance in the ESB at its top level.

**.sdc File Syntax and Example**

```verbatim
define_attribute {v:module_or_architecture_name} altera_implement_in_esb {1}
```

Examples:

```verbatim
// Applied to a module
define_attribute {v:mymux} altera_implement_in_esb {1}

// Applied to an instance
define_attribute {i:my} altera_implement_in_esb {1}
```

**Verilog Syntax and Example**

```verbatim
object /* synthesis altera_implement_in_esb = 1 */;
```

The following Verilog example shows how to implement a 2:1 multiplexer as a PTERM in an extended system block (ESB).

```verbatim
module mymux (in1, sel, dout);
  input [9:0] in1;
  input sel;
  output [9:0] dout;

  assign dout = sel ? in1 : ~in1;
endmodule

module esb_test (a, s, o);
  input [9:0] a;
  input s;
  output [9:0] o;

  // Apply attribute to instance
  mymux my (.in1(a), .sel(s), .dout(o))
    /* synthesis altera_implement_in_esb = 1 */;
endmodule
```

Synplify Reference Manual, April 2002

7-35
VHDL Syntax and Example

The following VHDL example shows how to implement a 2:1 multiplexer as a PTERM in an extended system block (ESB).

```
library ieee;
use ieee.std_logic_1164.all;
entity mymux is
  port (in1: in std_logic_vector(9 downto 0);
        sel : in std_logic;
        dout : out std_logic_vector(9 downto 0));
end mymux;

architecture behave of mymux is
begin
  dout <= in1 when sel = '1' else
         (NOT in1) when sel = '0' else
         (others => 'X');
end behave;

library ieee;
use ieee.std_logic_1164.all;

entity esb_test is
  port (a: in std_logic_vector(9 downto 0);
        s: in std_logic;
        o: out std_logic_vector(9 downto 0));
end esb_test;

architecture arch1 of esb_test is
component mymux is
  port (in1: in std_logic_vector (9 downto 0);
        sel : in std_logic;
        dout: out std_logic_vector (9 downto 0));
end component mymux;
attribute altera_implement_in_esb : boolean;
attribute altera_implement_in_esb of U1: label is true;

begin
  U1: mymux port map (
       in1 => a,
       sel => s,
       dout => o);
end arch1;
```
altera_io_opendrain

Attribute: Altera, APEX20KE. Supports an open-drain mode for output and bidirectional ports for I/Os. The value is Boolean and the default is 0, which disables open-drain mode.

.sdc File Syntax and Example

```
define_attribute {output | bidir} altera_io_opendrain {1 | 0}
```

For example:

```
define_attribute {aluout} altera_io_opendrain {1}
```

You can apply the `altera_io_opendrain` attribute to a port, bus, or slice of a bus using the SCOPE spreadsheet, whereas in the source code, this attribute can be applied only to the port definition.

1. On the Attributes panel of the SCOPE spreadsheet, select the port in the Object column.
2. Click the pull-down menu in the Attribute column and choose `altera_io_opendrain`.
3. Assign the appropriate Boolean value in the Value field.

Verilog Syntax and Example

```
object /* synthesis altera_io_opendrain = 1 | 0 */;
```

You can only apply the `altera_io_opendrain` attribute to the port definition. For example:

```
module alu (..., aluop, alua, alub, alu_cout, aluz, alout);
input [3:0] aluop;

// Apply to a bi-directional bus port
inout [7:0] alua /* synthesis altera_io_opendrain=0 */ , alub;

// Apply to a bit port
output alu_cout /* synthesis altera_io_opendrain=1 */ , aluz;
```
VHDL Syntax and Example

attribute altera_io_opendrain of object : object_type is true | false;

You can only apply the altera_io_opendrain attribute to the port definition. For example:

```vhdl
entity ALU is port (  
    A :  inout std_logic_vector(7 downto 0);  
    ALUOUT : out std_logic_vector(7 downto 0);  
    ALUCOUT : out std_logic;  
);  
attribute altera_io_opendrain : boolean;  
-- Other code
end ALU;
```

```vhdl
-- Apply to a bi-directional bus port  
attribute altera_io_opendrain of A : signal is false;

-- Apply to a bus port  
attribute altera_io_opendrain of ALUOUT : signal is true;

-- Apply to a bit port  
attribute altera_io_opendrain of ALUCOUT : signal is true;
end ALU;
```
**altera_io_powerup**

*Attribute; Altera APEX20KE.* Controls the power-up mode of registers in the I/O port (bit or bus). However, you can only specify this attribute if the I/O register does not have a defined preset or clear condition.

If the I/O register has a preset, powerup is set to high. If the I/O register has a clear, powerup is set to low. You can only specify the altera_io_powerup attribute to control the powerup mode if the I/O register is not defined.

By default, Quartus II sets the power-up mode of the I/O to low.

If register cannot be packed into an I/O, the Synplify synthesis tool issues a warning saying it has ignored the attribute.

**.sdc File Syntax and Examples**

```plaintext
define_attribute {port} altera_io_powerup {high | low}
```

You can apply the altera_io_powerup attribute to a port, bus, or slice of a bus in the SCOPE spreadsheet, whereas in the source code this attribute can only be applied to a port definition.

1. In the SCOPE spreadsheet, click the port in the Object column.
2. Click the pull-down menu in the Attribute column and choose altera_io_powerup.
3. Enter the appropriate string in the Value column: high or low.

Following is an example:

```plaintext
define_attribute {seg [31:0]} altera_io_powerup {high}
```

**Verilog Syntax and Example**

```plaintext
object /*synthesis altera_io_powerup = "high | low" */ ;
```

The following example demonstrates how to apply the altera_io_powerup attribute to a single-bit port in Verilog.
module dff_or(q, a, b, clk);
output q /* synthesis altera_io_powerup="low" */;
input a, b, clk;
reg q;

// Other code

**VHDL Syntax and Example**

```vhdl
attribute altera_io_powerup of object : object_type is "high | low";
```

The following demonstrates how to apply the `altera_io_powerup` attribute to a 32-bit bus port in VHDL.

```vhdl
entity eight_bit_uc is
port (    
clock : in std_logic;
resetn : in std_logic;
com : out std_logic_vector(3 downto 0);
seg : out std_logic_vector(31 downto 0);
porta : inout std_logic_vector(7 downto 0);
portb : inout std_logic_vector(7 downto 0);
portc : inout std_logic_vector(7 downto 0));

attribute altera_io_powerup : string;
attribute altera_io_powerup of seg : signal is "high";
end EIGHT_BIT_UC;
```
**Attributes**

**Attributes**

**.sdc File Syntax and Example**

```bash
define_attribute {input_port_name} din ""
```

For example:

```bash
define_attribute {load} din ""
```

**Verilog Syntax and Example**

```verilog
object * synthesis din = "" */;
```

This is an example of the `din` attribute in a Verilog source file:

```verilog
module counter4 (cout, out, in, ce, load, clk, rst);

// Place the flip-flops near the load input
input load /* synthesis din="" */;
endmodule
```

**VHDL Syntax and Example**

```vhdl
attribute din of object : object_type is "" ;
```

This is an example of the `din` attribute in a VHDL source file:

```vhdl
entity counter4 is
  port (cout: out bit;
       output_vector: out bit_vector (3 downto 0);
       input_vector: in bit_vector (3 downto 0);
       ce, load, clk, rst: in bit);

-- Place the flip-flops near "input_vector"
attribute din : string;
attribute din of input_vector : signal is "";
end counter4;
```
dout

Attribute; Lattice ORCA. Specifies that an output register goes in the location next to the I/O pad. You can attach dout to top-level ports or manually-instantiated input cells. The dout attribute is takes an empty string (""") as its value.

.sdc File Syntax and Example

```
define_attribute {output_port_name} dout ""
```

For example:
```
define_attribute {cout} dout ""
```

Verilog Syntax and Example

```
object /* synthesis dout = "" */ ;
```

This is an example of the dout attribute in a Verilog source file:
```
module counter4 (cout, out, in, ce, load, clk, rst);

    // Place the flip-flops near the cout output
    output cout /* synthesis dout="" */ ;

endmodule
```

VHDL Syntax and Example

```
attribute dout of object : object_type signal is "" ;
```

This is an example of the dout attribute in a VHDL source file:
```
entity counter4 is
    port (cout: out bit;
          output_vector: out bit_vector (3 downto 0);
          input_vector: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit);

    -- Place the flip-flops near "output_vector"
    attribute dout : string;
    attribute dout of output_vector: signal is "" ;

end counter4;
```
**loc**

*Attribute; Lattice.* The loc attribute specifies pin locations for Lattice MACH and ORCA I/Os. See also *lock on page 7-44*. Refer to the Lattice databook for valid pin location values.

**.sdc File Syntax and Example**

```verbatim
define_attribute {port_name} loc {pin_location}
```

The following example assigns a pad location to all bits of a bus:

```verbatim
define_attribute {DATA0[3:0]} loc {P14,P12,P11,P5}
```

**Verilog Syntax and Example**

```verbatim
object /* synthesis loc = "pin_location" */ ;
```

For example:

```verbatim
input [3:0]DATA0 /* synthesis loc = "p10,p12,p11,p15" */;
```

**VHDL Syntax and Example**

```verbatim
attribute loc of object : object_type is "pin_location" ;
```

For example:

```verbatim
category mycomp is port(DATA0 : in std_logic_vector (3 downto 0);
    .
    .
    .);
attribute loc : string;
attribute loc of DATA0 : signal is "p10,p12,p11,p15";
```
lock

Attribute; Lattice. The lock attribute specifies pin locations for Lattice ISP I/Os. See also loc on page 7-43. Refer to the appropriate Lattice databook for valid pin numbers.

.sdc File Syntax and Example

define_attribute {port_name} lock {pin_number}

For example:

define_attribute {ADDR[3:0]} lock {P6, P8, P10, P14}
**orca_padtype**

*Attribute; Lattice ORCA.* The orca_padtype attribute specifies the pad type for a Lattice ORCA I/O.

**.sdc File Syntax and Example**

```plaintext
define_attribute {port_name} orca_padtype {pad_type}
```

Values for `pad_type` are defined in the Lattice ORCA databook. The default for inputs is IBM, which is a CMOS pad type. The default for outputs is OB6.

This example defines bit 3 of AIN to be an ORCA TTL input pad.

```plaintext
define_attribute {AIN[3]} orca_padtype {IBT}
```

**Verilog Syntax and Example**

```plaintext
object/* synthesis orca_padtype = "pad_type" */;
```

This is an example of the orca_padtype attribute in a Verilog source file:

```plaintext
module counter4 (cout, out, in, ce, load, clk, rst);

// Choose IBT padtype to select a TTL input pad for "load"
input load /* synthesis orca_padtype="IBT" */;
endmodule
```
orca_padtype

**VHDL Syntax and Example**

```vhdl
attribute orca_padtype of object : object_type is "pad_type";
```

This is an example of the `orca_padtype` attribute in a VHDL source file:

```vhdl
entity counter4 is
  port (cout: out bit;
       output_vector: out bit_vector (3 downto 0);
       input_vector: in bit_vector (3 downto 0);
       ce, load, clk, rst: in bit);
-- Choose OB12 padtype for "output_vector".
-- The OB12 padtype is being set for all four
-- bits in "output_vector".
attribute orca_padtype : string;
attribute orca_padtype of output_vector: signal is "OB12";
end counter4;
```
orca_props

*Attribute; Lattice ORCA.* Specifies Lattice ORCA I/O properties on selected ports that are forward-annotated.

.sdc File Syntax and Example

```define_attribute {p:port} orca_props {property=value}```

For more information about the properties and their values, see *I/O Properties and Values* on page 7-48. The following is an example of the syntax:

```define_attribute {p:data_in} orca_props {LEVELMODE=LVDS}```

Verilog Syntax and Example

```object /* synthesis orca_props = "property=value" */;```

where `object` can be an input, output, bidirectional, or tristate ports. For more information about the properties and their values, see *I/O Properties and Values* on page 7-48. The following is an example of the syntax:

```output data_out /* synthesis orca_props = "LEVELMODE=LVDS" */;```

If multiple properties are required, separate them with commas. This is an example of an object with more than one property:

```output data_out /* synthesis orca_props = "BUFMODE=FAST,LEVELMODE=LVTTL" */;```

VHDL Syntax and Example

```attribute orca_props : string;
attribute orca_props of object : object_type is "property=value" ;```

For more information about the properties and their values, see *I/O Properties and Values* on page 7-48. The following is an example of the VHDL syntax:

```Entity test is port (a : in std_logic; b : out std_logic);
attribute orca_props : string;
attribute orca_props of a :signal is "LEVELMODE=LVDS";```
This is an example of the syntax when you have multiple properties:

```vhdl
Entity test is port (a : in std_logic; b : out std_logic);
attribute orca_props : string;
attribute orca_props of a :signal is "LEVELMODE=LVDS, AMPSMODE=6";
```

### I/O Properties and Values

The following table lists the properties and values that are currently supported.

<table>
<thead>
<tr>
<th>Property</th>
<th>Valid Values</th>
<th>Attached to</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMPSMODE</td>
<td>6</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>BUFMODE</td>
<td>SLOW</td>
<td>FAST</td>
<td>Bidirectional or output ports</td>
</tr>
<tr>
<td>DELAYMODE</td>
<td>true</td>
<td>false</td>
<td>Input ports</td>
</tr>
<tr>
<td>KEEPERMODE</td>
<td>off</td>
<td>on</td>
<td>Bidirectional or tristate ports</td>
</tr>
<tr>
<td>LEVELMODE</td>
<td>LVTTL</td>
<td>LVCMOS2</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>RESISTOR</td>
<td>off</td>
<td>on</td>
<td>Input or output ports</td>
</tr>
</tbody>
</table>
ql_padtype

Attribute; QuickLogic. Overrides the default pad types that the Synplify synthesis tool chooses for inputs, outputs, and inout (bidirectional) ports. By default, the tool infers pad types, using the available high-drive and clock pads where appropriate.

.sdc File Syntax and Example

```plaintext
define_attribute {port_name} ql_padtype {pad_type}
```

For example:

```plaintext
define_attribute {clk} ql_padtype {CLOCK}
```

Values for `pad_type` are:

- **INPUT** – A high-drive input. QuickLogic parts have high-drive inputs that vary in quantity depending upon the device. The number of high-drives varies with the part, refer to the QuickLogic data book for more information.

- **BIDIR** – An input, output, or bidirectional pad.

- **CLOCK** – A clock pad. QuickLogic clock pads are high-drive inputs that are routed on special clock wires (clock trees). CLOCK pad types can also be used for sets and resets, a feature the Synplify synthesis tool normally infers.

- **CLOCKB** – A special clock pad used only for QuickPCI QL5064 devices.

Verilog Syntax and Example

```plaintext
object /* synthesis ql_padtype = "pad_type" */ ;
```

For example:

```plaintext
module ql_padtype_example (d, q, clk, rst);
  input d  /* synthesis ql_padtype = "INPUT" */ ;
  input clk /* synthesis ql_padtype = "CLOCK" */ ;
  input rst /* synthesis ql_padtype = "INPUT" */ ;
  output q  /* synthesis ql_padtype = "BIDIR" */ ;
  reg q;
```
always@(posedge clk or posedge rst)
if (rst)
  q = 1'b0;
else
  q = d;
endmodule

**VHDL Syntax and Example**

attribute ql_padtype of object: object_type is "pad_type";

For example:

library ieee;
use ieee.std_logic_1164.all;

entity ql_padtype_example is
  port (
    d   : in  std_logic;
    q   : out std_logic;
    clk : in  std_logic;
    rst : in  std_logic
  );

attribute ql_padtype: string;
attribute ql_padtype of  d: signal is "INPUT";
attribute ql_padtype of clk: signal is "CLOCK";
attribute ql_padtype of rst: signal is "INPUT";
attribute ql_padtype of q: signal is "BIDIR";
end ql_padtype_example;

architecture rtl of ql_padtype_example is
begin
  process(clk, rst)
  begin
    if (rst='1')then
      q <= '0';
    elsif (clk'event and clk='1')then
      q <= d;
    end if;
  end process;
end rtl;
ql_placement

Attribute; QuickLogic. Used for forward annotation of pad locations for use by the QuickLogic QuickWorks place-and-route tool. ql_placement overrides the default pad locations chosen by QuickWorks.

Values for ql_placement depend on the device package documented in the QuickLogic data book. All placement names must match the pin names as they appear in the QuickLogic data book (for example, A21, B23).

.sdc File Syntax and Example

```text
define_attribute {port_or_instance_name} ql_placement {placement}
```

Examples:

```text
define_attribute {d} ql_placement {A21}
define_attribute {q} ql_placement {B40}
define_attribute {clk} ql_placement {B23}
```

Verilog Syntax and Example

```text
object /* synthesis ql_placement = "placement" */ ;
```

For example:

```text
module ql_placement_example (d, q, clk, rst);
   input d   /* synthesis ql_placement = "A21" */;
   input clk /* synthesis ql_placement = "B23" */;
   input rst /* synthesis ql_placement = "A31" */;
   output q  /* synthesis ql_placement = "B40" */;
   reg q;
   always@(posedge clk or posedge rst)
      if (rst)
         q = 1'b0;
      else
         q = d;
endmodule
```

VHDL Syntax and Example

```text
attribute ql_placement of object : object_type is "placement" ;
```
library ieee;
use ieee.std_logic_1164.all;

entity ql_placement_example is
  port ( 
    d : in  std_logic;
    q : out std_logic;
    clk : in  std_logic;
    rst : in  std_logic
  );

attribute ql_placement: string;
attribute ql_placement of   d: signal is "A21";
attribute ql_placement of clk: signal is "B23";
attribute ql_placement of rst: signal is "A31";
attribute ql_placement of   q: signal is "B40";
end ql_placement_example;

architecture rtl of ql_placement_example is
begin
  process(clk, rst)
  begin
    if (rst='1') then
      q <= '0';
    elsif (clk'event and clk='1') then
      q <= d;
    end if;
  end process;
end rtl;
**syn_direct_enable**

Attribute; The `syn_direct_enable` attribute controls the assignment of a clock enable net to the dedicated enable pin of a storage element (flip-flop). Using this attribute, you can direct the mapper to use a particular net as the only clock enable when the design has multiple clock-enable candidates.

You can also use this attribute as a compiler directive to infer flip-flops with clock enables. To do so, enter `syn_direct_enable` as a directive in source code, not the SCOPE spreadsheet.

This directive is supported in the following technologies:

<table>
<thead>
<tr>
<th><strong>Altera</strong></th>
<th><strong>Xilinx</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACEX</td>
<td>Virtex</td>
</tr>
<tr>
<td>APEX20K</td>
<td>Virtex-E</td>
</tr>
<tr>
<td></td>
<td>Virtex2</td>
</tr>
<tr>
<td></td>
<td>Virtex2p</td>
</tr>
<tr>
<td>FLEX10K</td>
<td>Spartan2</td>
</tr>
<tr>
<td></td>
<td>XC4000</td>
</tr>
<tr>
<td>Stratix</td>
<td>XC5200</td>
</tr>
</tbody>
</table>

The `syn_direct_enable` data type is Boolean. A value of 1 or true enables net assignment to the clock enable pin.

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_direct_enable = 1 */;
```

For example:
```
`timescale 1 ns/ 100 ps
module dff2(q1, d1, clk, e1, e2, e3);
input [4:0] d1;
input clk;
output [4:0] q1;
reg [4:0] q1;
input e1, e3;
input e2 /* synthesis syn_direct_enable = 1 */;
always @(posedge clk)
begin
  if (e1 & e2 & e3 ) begin
    q1 = d1;
  end
end
endmodule

VHDL Syntax and Examples

attribute syn_direct_enable of object : object_type is true;

The first example shows signals:

architecture dff2_arch of dff2 is
signal enable : std_logic;
attribute syn_direct_enable : boolean;
attribute syn_direct_enable of enable : signal is true;

The second example shows ports in an entity.

entity dff2 is
port (q1 : out std_logic_vector (4 downto 0);
  d1: in std_logic_vector (4 downto 0);
  clk , e1, e2 , e3 : in std_logic);
attribute syn_direct_enable : boolean;
attribute syn_direct_enable of e2 : signal is true;
end dff2;
```
**syn_edif_bit_format**

*Attribute; Xilinx.* Global character formatting attribute to use with vector (bus) port names to control the naming style in the EDIF output file. The first argument (%n, %d, or %u) controls the case of the names, while the second argument (%i) controls the appearance of the vector range. The default preserves the character case of the names as they appear in source code. To change the character formatting of scalar ports, see the *syn_edif_scalar_format* attribute.

**.sdc File Syntax and Examples**

The constraint file syntax for the global attribute is:

```
define_global_attribute syn_edif_bit_format { %n | %u | %d [char] %i [char] }
```

Use this attribute to control the following:

1. Up-shift (%u) or down-shift (%d) the character case of entire names; %n means leave case as is.
2. Append the vector range to the end of names.
3. Append additional characters to the end of names.
4. Convert the default bus delimiters to another representation (parentheses, square brackets or angle brackets). By default, Verilog designs produce square bracket delimiters in EDIF and VHDL designs produce parenthesis delimiters in EDIF.

**Example: Converting Bus Delimiters**

The port definition as it would appear in Verilog source code:

```
input [31:0] AbCd;
```

Specifying the attribute in a constraint file to convert the bus delimiters to angle brackets(<%i>) in the EDIF file:

```
define_global_attribute syn_edif_bit_format { %n<%i> }
```

The port definition as it would appear in the EDIF output file; note the use of angle brackets as bus delimiters:

```
(port (array (rename abcd "AbCd<31:0>") 32) (direction INPUT))
```
Example: Appending the Vector Range
The port definition as it appears in Verilog source code:

    input[31:0] AbCd;

Specifying the attribute in a constraint file to append the vector range (_%i) in the EDIF file:

    define_global_attribute syn_edif_bit_format {%n_%i}

The port definition as it appears in the EDIF output file; note the appended vector ranges:

    (port (array (rename abcd "AbCd_31") 32) (direction INPUT))
    (port (array (rename abcd "AbCd_30") 32) (direction INPUT))
    (port (array (rename abcd "AbCd_29") 32) (direction INPUT))
    .
    .
    (port (array (rename abcd "AbCd_0") 32) (direction INPUT))

Example: Changing All Characters to Uppercase
The port definition as it appears in Verilog source code:

    input[31:0] AbCd;

Specifying the attribute in a constraint file to shift the names to uppercase (%u) and use parentheses as the bus delimiters (%i):

    define_global_attribute syn_edif_bit_format {%u(%i)}

The port definition as it appears in the EDIF output file; note the uppercase name and the use of parentheses as bus delimiters:

    (port (array (rename abcd "ABCD(31:0)") 32) (direction INPUT))

Example: Appending Additional Characters
The port definition as it appears in Verilog source code:

    input[31:0] AbCd;

Specifying the attribute in a constraint file to append additional characters (_mydesign) and use parentheses as the bus delimiters (%i):

    define_global_attribute syn_edif_bit_format {%n_mydesign(%i)}
The port definition as it appears in the EDIF output file; note the appended characters and the use of parentheses as bus delimiters

```
(port (array (rename abcd "AbCd_mydesign(31:0)") 32)
(direction INPUT)))
```

**Verilog Syntax and Example**

```
object /* synthesis syn_edif_bit_format = "\%n \%u \%d [char] \%i [char]" */ ;
```

The following example shows how to set the bus format for a Verilog module. In the EDIF output file the case of the name will be unchanged (%n) and the bus delimiters will be angle brackets (<%i>):

```
module pci_lc_i (...)
/* synthesis syn_black_box syn_edif_bit_format = "%n<%i>" */;
```

**VHDL Syntax and Example**

```
attribute syn_edif_bit_format of object : object_type is "%n | %u | %d [char] %i [char]" ;
```

The following example shows how to specify angle brackets as the bus delimiters for the VHDL black-box component pci_lc_i:

```
component pci_lc_i
  port (.
  .
  .);
end component;
attribute syn_edif_bit_format of pci_lc_i : component is "%n<%i";
```

The following example shows how to shift the name to uppercase and specify square brackets as the bus delimiters for the VHDL black-box architecture rtl:

```
architecture rtl of pci_top is
attribute syn_edif_bit_format of rtl : architecture is "%u[%i]";
```
syn_edif_name_length

*Attribute; Altera.* Global truncation attribute that determines the length of port names written to the EDIF output file for Altera designs. If your target place-and-route tool is Quartus II, the length of the port names does not matter. However, if Max+PlusII is the target tool, port names cannot be more than 30 characters in length.

You specify this attribute globally on the top-level module or architecture. The value of the attribute can be either restricted or unrestricted. The restricted setting (the default) limits the port names to 30 characters. Use this setting with Max+PlusII. If you set the attribute to unrestricted, the port names are not truncated. Use this value if you are targeting Quartus II as the place-and-route tool.

**.sdc File Syntax**

The constraint file syntax for the global attribute is as follows:

```syntactically_conscious_language
define_global_attribute syn_edif_name_length {restricted | unrestricted}
```

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_edif_name_length = "restricted | unrestricted" */ ;
```

Attach the attribute to the top-level module. The following is an example:

```verilog
module test(in1, in2 , out )
    /* synthesis syn_edif_name_length = "restricted" */;
    input in1, in2;
    output out;
    textbox inst1(.in1(in1), .in2(in2), .abcdefghijklmnopqrstuvwxyz_abcdedfghijklmnopqrstuvwxyz (out));
endmodule
```
VHDL Syntax and Example

attribute syn_edif_name_length of object:object_type is "restricted | unrestricted";

Attach the attribute to the top-level entity, as shown in this example:

architecture beh of test is
    component testbox is port(
        in1, in2 : in std_logic;
        abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz : out
        std_logic);
    end component;
attribute syn_black_box : boolean;
attribute syn_black_box of testbox : component is true;
attribute syn_edif_name_length : string;
attribute syn_edif_name_length of beh : architecture is
    "Unrestricted";

begin
    u0: testbox port map (in1 => in1, in2 => in2,
        abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz
        => out1);
end beh;
syn_edif_scalar_format

**syn_edif_scalar_format**

*Attribute; Xilinx.* Global character formatting attribute to use with scalar port names to up-shift (%u) or down-shift (%d) the character case of the name in the EDIF output file. The default (syn_edif_scalar_format (%n)) preserves the character case of the names as they appear in source code. To change the character formatting and style of vector (bus) signals and ports, see the syn_edif_bit_format attribute.

**.sdc File Syntax and Examples**

The constraint file syntax for the global attribute is:

```
define_global_attribute syn_edif_scalar_format {%u | %d | %n}
```

For example, the port definition as it would appear in source code:

```verbatim
input AbCd;
```

Specifying the attribute in a constraint file to up-shift the base names:

```verbatim
define_global_attribute syn_edif_scalar_format {%u}
```

The port definition as it would appear in the EDIF output file.

```verbatim
(port (rename AbCd "ABCD") (direction INPUT))
```

**Verilog Syntax and Example**

```verbatim
object /* synthesis syn_black_box syn_edif_scalar_format = "%u | %d | %n" */;
```

where *object* is a black-box module.

The following example shows how to shift all scalar port names of the Verilog black-box module pci_lc_i to uppercase:

```verbatim
module pci_lc_i (...) /* synthesis syn_black_box syn_edif_scalar_format = "%u" */;
```
VHDL Syntax and Examples

```
attribute syn_edif_scalar_format of object : object_type is "%u | %d | %n" ;
```

where `object` is a VHDL black-box component or architecture.

The following example shows how to shift all scalar port names of the VHDL black-box component `pci_lc_i` to uppercase:

```
component pci_lc_i
  port ( ...
  ...
    ...
  );
end component;

attribute syn_black_box of pci_lc_i : component is true;
attribute syn_edif_scalar_format of pci_lc_i : component is "%u";
```

The following example shows how to shift all scalar port names of the VHDL black-box architecture `rtl` to lowercase:

```
architecture rtl of pci_top is
attribute syn_black_box of rtl : architecture is true;
attribute syn_edif_scalar_format of rtl : architecture is "%d";
```
**syn_encoding**

*Attribute.* Overrides the default FSM Compiler encoding for a state machine. This attribute takes effect only when FSM Compiler is enabled. See *FSM Compiler on page 1-21*, for more information on FSM Compiler.

The default encoding style automatically assigns encoding based on the number of states in the state machine as follows:

- **sequential** for 0-4 states
- **onehot** for 5-24 states
- **gray** for >24 states

Use the `syn_encoding` attribute when you want to override these defaults. You can also use `syn_encoding` when you want to disable the FSM Compiler globally but there are a select number of state registers in your design that you want extracted. In this case, you would use this attribute with the `syn_state_machine` directive on just those specific registers.

`syn_encoding` can have the following values:

- **default** – synthesis selects encoding style based on the number of states. See default values above.
- **onehot** – Only two bits of the state register change (one goes to ‘0’, one goes to ‘1’) and only one of the state registers is hot (driven by ‘1’) at a time. For example:
  
  \[
  0001, 0010, 0100, 1000
  \]

- **gray** – More than one of the state registers can be hot. The Synplify synthesis tool *attempts* to have only one bit of the state registers change at a time, but it might allow more than one bit to change, depending upon certain conditions for optimization. For example:
  
  \[
  000, 001, 011, 010, 110
  \]

- **sequential** – More than one bit of the state register can be hot. The synthesis tool makes no attempt at limiting the number of bits that can change at a time. For example:
  
  \[
  000, 001, 010, 011, 100
  \]

- **safe** – This implements the state machine in the default encoding, and adds reset logic to force the state machine to a known state if it
reaches an invalid state. This option can be used in combination with any of the other encoding styles described above. For example, if the default encoding is onehot, and the state machine reaches a state where all the bits are 0, which is an invalid state, the safe value ensures that the state machine is reset to a valid state.

The encoding style is implemented during the mapping phase. A message is displayed when the synthesis tool extracts a state machine, for example:

```
@N: "c:\design\..."|Trying to extract state machine for register current_state
```

The log file reports the encoding styles used for the state machines in your design.

See also the following:

- For information on enabling state machine optimization for individual modules, see `syn_state_machine` on page 7-173.
- For VHDL designs, see `syn_encoding Versus syn_enum_encoding` on page 7-154 for comparative usage information.

**Verilog Syntax and Examples**

```verilog
object /* synthesis syn_encoding = "value" */ ;
```

where `object` can be `reg` (register definition signals that hold the state values of state machines) and `value` can be: onehot, gray, sequential, safe, default.

In this example, `syn_encoding` overrides the default encoding style for `current_state` using the gray encoding style.

```verilog
module prep3 (CLK, RST, IN, OUT);
input CLK, RST;
input [7:0] IN;
output [7:0] OUT;
reg [7:0] OUT;
reg [7:0] current_state /* synthesis syn_encoding="gray" */;

// Other code
```
5. Here is an example using safe, gray.

```vhdl
module prep3 (CLK, RST, IN, OUT);
    input CLK, RST;
    input [7:0] IN;
    output [7:0] OUT;
    reg [7:0] OUT;
    reg [7:0] current_state /* synthesis syn_encoding="safe,gray" */;

    // Other code
```

In this example, the encoding style for register `OUT` is gray and, by specifying `safe`, if the state machine reaches an invalid state, the Synplify synthesis tool will reset the values to a valid state.

### VHDL Syntax and Example

```vhdl
attribute syn_encoding of object : object_type is "string";
```

where `object` is a signal that holds the state values of the state machines. Here is an example of using `syn_encoding` to define the gray encoding style for the signal `s1`.

```vhdl
library synplify;
package my_states is
type state is (Xstate, st0, st1, st2, st3, st4, st5, st6, st7,
    st8, st9, st10, st11, st12, st13, st14, st15);
signal s1 : state;
attribute syn_encoding : string;
attribute syn_encoding of s1 : signal is "gray";
end my_states;
```
Attributes | syn_forward_io_constraints
---|---

**syn_forward_io_constraints**

*Attribute; Altera, Xilinx.* Enables/disables forward annotation of the define_input_delay and define_output_delay timing constraints in the .tcl or .ncf file for the place-and-route tool. The default is 1 for Xilinx (I/O constraints are forward-annotated) and 0 for Altera (I/O constraints are not forward-annotated).

Use this attribute on the top level of a VHDL or Verilog file, or you can place syn_forward_io_constraints on a global object in the .sdc file.

**.sdc File Syntax and Example**

```
define_global_attribute syn_forward_io_constraints = {1 | 0}
```

For example:

```
define_global_attribute syn_forward_io_constraints {1}
```

**Verilog Syntax and Example**

```
module test (a,b,c,d,clk,rst)
object /* synthesis syn_forward_io_constraints = 1 | 0 */;
input a,b,c;
input clk, rst;
output d;
wire temp;
reg d;
assign temp = a && b || c;
always@(posedge clk or posedge rst)
  if (rst)
    d = 1 'b0;
  else
    d = temp;
endmodule
```
VHDL Syntax and Example

\texttt{attribute syn_forward_io_constraints of object : object_type is true | false ;}

For example:

\begin{verbatim}
library IEEE;
use ieee.std_logic_1164.all;
entity test is port ( 
    a : in std_logic;
    b : in std_logic;
    clk :in std_logic;
    rst : in std_logic;
    d: out std_logic);
end test;

architecture archtest of test is
attribute syn_forward_io_constraints : boolean;
attribute syn_forward_io_constraints of all :
arithmetic is true;
signal temp : std_logic;
begin
    temp <= a and b;
    process (clk, rst)begin
        if (rst = '1')then 
            d <= '0';
        elsif (clk'event and clk = '1')then 
            d <= temp;
        end if;
    end process;
end archtest;
\end{verbatim}
syn_hier

Attribute. Allows you to control the amount of hierarchical transformation that occurs across boundaries on module or component instances during optimization.

During synthesis, the Synplify tool dissolves as much hierarchy as possible to allow efficient optimization of logic across hierarchical boundaries while maintaining optimal run times. The tool then rebuilds the hierarchy to be as close as possible to the original source code in an effort to preserve the topology of the design. However, you can use the syn_hier attribute to address any specific needs you might have to maintain design hierarchy during optimization. This attribute allows you manual control over flattening for instances, modules or architectures in the design.

Here are the values you can use for syn_hier:

- **soft** (default) – when you use soft, the synthesis tool determines the best optimization across hierarchical boundaries. This attribute affects only the design unit in which it is specified.
- **firm** – preserves the interface of the design unit and allows for cell packing across the boundary. This attribute affects only the design unit in which it is specified.
- **hard** – (Actel, except 500K and PA; Altera and Xilinx; other vendors: same as firm) preserves the interface of the design unit with no exceptions. This attribute affects only the specified design units.
- **remove** – removes the level of hierarchy for the design unit in which it is specified. The hierarchy at lower levels is unaffected.
- **macro** – (Actel except 500K and PA, Altera, Lattice ORCA and QuickLogic only) preserves the interface and contents of the design with no exceptions.
- **flatten** – flattens the hierarchy of all levels below, but not the one in which it is specified. By default, the hierarchy of the design is restored in the output netlist and in the HDL Analyst schematic view.

You can optionally use flatten in combination with other syn_hier options as shown below:

- flatten,soft – same as flatten.
- `flatten.firm` – flattens all lower levels of the design but preserves the interface of the design unit in which it is specified. This option also allows optimization of cell packing across the boundary.

- `flatten.remove` – flattens all lower levels of the design including the one in which it is specified.

If you use `flatten` in combination with another option, the tool flattens as directed until encountering another `syn_hier` attribute at a lower level. The lower level `syn_hier` attribute then takes precedence over the higher level one.

Here is an example of two versions of a design: one with `syn_hier` set on modules `inc` and `reg4`; the other shows what happens to those modules with the automatic flattening that occurs during synthesis.

With `syn_hier="hard"`

```
.define_attribute {v:fifo} syn_hier {hard}
```

### .sdc File Syntax and Example

```
define_attribute {object} syn_hier {value}
```

where `object` can be module declarations or architecture names. Check the attribute values to determine where you should attach the attribute. The syntax is

```
define_attribute {v:fifo} syn_hier {hard}
```
**Verilog Syntax and Examples**

```verilog
object/* synthesis syn_hier = "value" */ ;
```

where `object` can be a module declaration. Check the attribute values to determine where you should attach the attribute. The Verilog syntax is

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

// Other code
```

This next example demonstrates the `flatten,remove` value which states to flatten the current level of the hierarchy and all levels below it (unless another `syn_hier` attribute is found at a lower level).

```verilog
module top1 (Q, CLK, RST, LD, CE, D)
    /* synthesis syn_hier = "flatten,remove" */;

// Other code
```

**VHDL Syntax and Examples**

```vhdl
attribute syn_hier of object : object_type is "value" ;
```

where `object` can be architecture names. Check the attribute values to determine the level at which you should attach the attribute.

```vhdl
architecture struct of cpu is

attribute syn_hier : string;
attribute syn_hier of struct: architecture is "firm";

-- Other code
```

This next example demonstrates the `flatten,remove` value for this attribute. This value flattens the current level of the hierarchy and all levels below it (unless another `syn_hier` attribute is found at a lower level).

```vhdl
architecture struct of cpu is

attribute syn_hier : string;
attribute syn_hier of struct: architecture is "flatten,remove";

-- Other code
```
**syn_maxfan**

*Attribute; No Cypress or Lattice support.* Overrides the default (global) fanout guide for an individual input port, net or register output. You set the default Fanout Guide for a design through the Device panel on the Options for Implementation dialog box (see *Options for implementation Dialog Box on page 3-31*) or using the set_option -fanout_limit command or -fanout_guide (for Virtex) in the project file. Use the *syn_maxfan* attribute to specify a different (local) value for individual I/Os.

Both *syn_maxfan* and the default fanout guide are suggested *guidelines* only; they do not function as hard limits. This means that the Synplify synthesis tool takes them into account but does not always respect them absolutely. If they impose constraints that interfere with optimization they are not respected.

You can apply the *syn_maxfan* attribute to an input port, register output, or a net. If you apply it to a net, the Synplify synthesis tool creates a KEEPBUF component, and attaches the attribute to it, because the net itself might be optimized away during synthesis. If you attach the attribute to a lower level module or entity that is subsequently optimized during synthesis, the synthesis tool moves the *syn_maxfan* attribute up to the next higher level. If you do not want *syn_maxfan* moved up during optimization, set the *syn_hier* attribute for the entity or module to hard. This prevents the module or entity from being flattened when the design is optimized.

The *syn_maxfan* attribute is often used along with the *syn_noclockbuf* attribute on an input port that you do not want buffered. There are a limited number of clock buffers in a design, so if you want to save these special clock buffer resources for other clock inputs, put the *syn_noclockbuf* attribute on the clock signal. If timing for that clock signal is not critical, you can turn off buffering completely, to save area. To turn off buffering, set the maximum fanout to a very high number; for example, 1000.

**.sdc File Syntax and Example**

```
define_attribute {object} syn_maxfan {integer}
```

The following example limits the fanout for the signal clk to 200:

```
define_attribute {clk} syn_maxfan {200}
```
Verilog Syntax and Example

```verilog
object /* synthesis syn_maxfan = "value" */ ;
```

For example:

```verilog
module test (registered_data_out, clock, data_in);
output [31:0] registered_data_out;
input clock;
input [31:0] data_in /* synthesis syn_maxfan=1000 */;
reg [31:0] registered_data_out /* synthesis syn_maxfan=1000 */;

// Other code
```

VHDL Syntax and Example

```vhdl
attribute syn_maxfan of object : object_type is "value" ;
```

For example:

```vhdl
entity test is
  port(clock : in bit;
       data_in : in bit_vector(31 downto 0);
       registered_data_out: out bit_vector(31 downto 0)
  )

attribute syn_maxfan : integer;
attribute syn_maxfan of data_in : signal is 1000;

-- Other code
```
**syn_multstyle**

*Attribute; Altera Stratix, Xilinx Virtex2/Virtex2p.*

This attribute determines how multipliers are implemented: as dedicated hardware multiplier blocks or as logic. You can apply it to multiplier instances of Altera Stratix or Xilinx Virtex2/Virtex2p devices.

For Altera Stratix, the default implementation uses DSP MAC blocks. You can override this behavior by specifying value `lpm_mult` or `logic`. The implementation uses LPMs when the value is `lpm_mult`; it uses logic when the value is `logic`.

For Xilinx Virtex2/Virtex2p, the default implementation uses block multipliers (this corresponds to the attribute value `block_mult`). You can override this behavior by specifying a value of `logic`; the implementation then uses logic.

### .sdc File Syntax and Example

```
define_attribute (object) syn_multstyle (block_mult | logic | lpm_mult)
```

This example specifies that multipliers are to be implemented as logic.

```
define_attribute {temp[15:0]} syn_multstyle {logic}
```

### Verilog Syntax and Example

```
object /* synthesis syn_multstyle = "block_mult | logic | lpm_mult" */;

module mult(a,b,c,r,en);
  input [7:0] a,b;
  output [15:0] r;
  input [15:0] c;
  input en;

  wire [15:0] temp /* synthesis syn_multstyle="logic" */;
  assign temp = a*b;
  assign r = en ? temp : c;
endmodule
```
 VHDL Syntax and Example

```vhdl
attribute syn_multstyle of object : object_type is "block_mult | logic | lpm_mult" ;

library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity onereg is port ( 
  r : out std_logic_vector(15 downto 0);
  en : in std_logic;
  a: in std_logic_vector(7 downto 0);
  b: in std_logic_vector(7 downto 0);
  c : in std_logic_vector(15 downto 0) 
);

end onereg;

architecture beh of onereg is
signal temp : std_logic_vector(15 downto 0);
attribute syn_multstyle : string;
attribute syn_multstyle of temp : signal is "logic";

begin
  temp <= a * b;
  r <= temp when en='1' else c;
end beh;
```
**syn_netlist_hierarchy**

*Attribute; Actel, Altera, Lattice ORCA, Xilinx.* A global attribute that controls the generation of hierarchy in the EDIF output (result file) when you assign it to the top-level module of your design. The default (true) is to allow hierarchy generation.

**.sdc File Syntax and Example**

```
define_global_attribute syn_netlist_hierarchy {0 | 1}
```

For example:

```
define_global_attribute syn_netlist_hierarchy {0}
```

**Verilog Syntax and Example**

```
object /* synthesis syn_netlist_hierarchy = 0 | 1 */ ;
```

where *object* can be a top-level module declaration. For example:

```
module top (clk, qout, a, b)
    /*synthesis syn_netlist_hierarchy=0 */;
```

```
// Other code
```

**VHDL Syntax and Example**

```
attribute syn_netlist_hierarchy of object : object_type is true | false ;
```

where *object* can be a top-level architecture name. The data type is Boolean. For example:

```
architecture top of top is

attribute syn_netlist_hierarchy : boolean;
attribute syn_netlist_hierarchy of top : architecture is false;
```

```
-- Other code
```
**syn_noarrayports**

*Attribute.* Specifies that the ports of a design unit should be treated as individual signals (scalars) and not as a bus (array) in the EDIF file.

### .sdc File Syntax and Example

```
define_global_attribute syn_noarrayports {0 | 1}
```

For example:

```
define_global_attribute syn_noarrayports {1}
```

### Verilog Syntax and Example

```
object /* synthesis syn_noarrayports = 0 | 1 */;
```

Where `object` can be a module declarations. For example:

```
module adder8(cout, sum, a, b, cin)
    /* synthesis syn_noarrayports = 1 */;

// Other code
```

### VHDL Syntax and Example

```
attribute syn_noarrayports of object : object_type is true | false;
```

where `object` is an architecture name. Data type is Boolean.

In this example, the ports of `adder8` are treated as scalars during synthesis.

```
architecture adder8 of adder8 is

attribute syn_noarrayports : boolean;
attribute syn_noarrayports of adder8 : architecture is true;

-- Other code
```
**syn_noclockbuf**

*Attribute. Actel, Atmel, QuickLogic, Xilinx.* The Synplify synthesis tool uses clock buffer resources, if they exist in the target module, and puts them on the highest fanout clock nets. You can turn off automatic clock buffer usage by using the `syn_noclockbuf` attribute. For example, you might want to put a clock buffer on a lower fanout clock that has a higher frequency and tighter timing constraint.

You can turn off automatic clock buffering for entire modules or nets, or just for specific input ports. The value is Boolean with 1 or true turning off automatic clock buffering.

**.sdc File Syntax and Example**

```sh
define_attribute {clock_port} syn_noclockbuf {1 | 0}

define_global_attribute syn_noclockbuf {1 | 0}
```

For example:

```sh
define_attribute {clk} syn_noclockbuf {1}

define_global_attribute syn_noclockbuf {1}
```

**Verilog Syntax and Examples**

```verilog
object /* synthesis syn_noclockbuf = 1 | 0 */;
```

Here is an example of turning off automatic clock buffering for an entire module.

```verilog
module my_design(out, in, clk_in)
    /* synthesis syn_noclockbuf = 1 */;
```

This next example demonstrates turning off automatic clock buffering for a specific input port:

```verilog
module my_design(out, in, clk_in);
output out;
input in;
input clk_in /* synthesis syn_noclockbuf = 1 */;
```

```
// Other code
```
VHDL Syntax and Examples

attribute syn_noclockbuf of object : object_type is true | false ;

Where object can be an architecture or an input (clock) port.

This example turns off automatic clock buffering for an entire architecture using the syn_noclockbuf attribute from the Synplicity attributes package.

```vhdl
library ieee, synplify;
use ieee.std_logic_1164.all;
entity simpledff is
  port (q : out std_logic_vector(7 downto 0);
        d : in std_logic_vector(7 downto 0);
        clk : in std_logic);
end simpledff;
architecture behavior of simpledff is
  -- Turn off automatic clock buffers for this architecture, by
  -- setting the attribute on the architecture itself.
  attribute syn_noclockbuf : boolean;
  attribute syn_noclockbuf of behavior : architecture is true;
  -- Coding for the behavior of this architecture
end simpledff;
```

The next example demonstrates turning off automatic clock buffering for and individual signal:

```vhdl
library ieee, synplify;
use ieee.std_logic_1164.all;
entity simpledff is
  port (q : out std_logic_vector(7 downto 0);
        d : in std_logic_vector(7 downto 0);
        clk : in std_logic);
  -- Turn off automatic clock buffering on clk
  attribute syn_noclockbuf : boolean;
  attribute syn_noclockbuf of clk : signal is true;
end simpledff;
architecture behavior of simpledff is
  -- Coding for the behavior of this architecture
```

```vhdl
```
**syn_pipeline**

*Attribute; Altera (APEX, FLEX10K, Mercury, Excalibur, ACEX, and ACEX II); Xilinx (XC4000 and Virtex families).* Specifies that registers be moved into multipliers to improve frequency. For Xilinx, you can also pipeline ROMs for the XC4000 and Virtex architectures if registers exist in the RTL code.

### .sdc File Syntax and Example

```
define_attribute {register} syn_pipeline {1 | 0}
```

For example:

```
define_attribute {res[15:0]} syn_pipeline {1}
```

### Verilog Syntax and Example

```
object /* synthesis syn_pipeline = 1 | 0 */;
```

This example shows you how to pipeline a multiplier with `syn_pipeline`.

```verilog
`define lefta 7
`define leftb 7
module mult(r, a, b, preset, clk);
output [`lefta+`leftb +1:0] r;
input [`lefta:0] a;
input [`leftb:0] b;
input clk;
input preset;
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;
wire [`lefta+`leftb+1:0] res_out1 = a_aux * b_aux;
wire [`lefta+`leftb+1:0] res_out2 = res;
always @(posedge clk or posedge preset)
begin
  if (preset)
    res = 16'hFFF;
  else
    res = res_out1;
end
```
always @(posedge clk or posedge preset)
begin
  if (preset)
    res1 = 16'hFFFF;
  else
    res1 = res_out2;
end

always @(posedge clk)
begin
  a_aux = a;
  b_aux = b;
end

assign r = res1;
endmodule

**VHDL Syntax and Example**

```vhdl
attribute syn_pipeline of object : object_type is true | false;
```

For example:

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

entity onereg is port (  
  r : out std_logic_vector(31 downto 0);
  a : in std_logic_vector(15 downto 0);
  b : in std_logic_vector(15 downto 0);
  clk, rst : in std_logic );
end onereg;

architecture beh of onereg is
  signal temp1, temp2, temp3, wire : 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;
```
begin
  process(clk, rst)
  begin
    if rst='1' then
      temp1 <= (others => '0');
      temp2 <= (others => '0');
      temp3 <= (others => '0');
    elsif
      clk'event and clk='1' then
        temp1 <= a * b;
        temp2 <= temp1;
        temp3 <= temp2;
    end if;
  end process;

  wire <= temp3;
  r <= wire;
end beh;
**syn_preserve_sr_priority**

*Attribute; Actel.* Globally implements hardware that forces set/reset flip-flops to honor the priority for the set or reset, as coded in the design. This attribute can increase the area of your design.

Actel’s sequential components do not have a well-defined behavior when both the set and reset signals are simultaneously active, but you can have the Synplify synthesis tool automatically add hardware to force the priority that you have coded in the source code of your design.

**.sdc File Syntax and Example**

```sdc
define_global_attribute syn_preserve_sr_priority {1}
```

For example:

```sdc
define_global_attribute syn_preserve_sr_priority {1}
```

Most language models for a set/reset level-sensitive latch or D flip-flop define the behavior for this condition. Using this attribute sets the priority of set/reset as it is specified in your HDL source code. The following Verilog code implies that reset has priority over set if both are active:

```verilog
if (reset)
    q=0;
else if (set)
    q=1;
else
    q=d;
end if;
```

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_preserve_sr_priority = 1 */;
```

The following example sets the `syn_preserve_sr_priority` attribute on the latch3 module. The value 1 indicates that the attribute is on.

```verilog
module latch3(q, data, set, reset, clk)
    /* synthesis syn_preserve_sr_priority = 1 */;
        output q;
        input data, clk, set, reset ;
        reg q;
```
always @(clk or data or set or reset)
begin
  if (reset)
    q = 0;
  else if (set)
    q = 1;
  else if (clk)
    q = data;
end
endmodule

VHDL Syntax and Example

attribute syn_preserve_sr_priority of object : object_type is true;

Where object is the entity and object_type is architecture.

Here is an example:

library ieee;
use ieee.std_logic_1164.all;
entity dff1 is
  port (data, clk, reset, set : in std_logic;
        qrs: out std_logic);
end dff1;
architecture async_set_reset of dff1 is
  -- Set the attribute to "true" for async_set_reset architecture
  attribute syn_preserve_sr_priority : boolean;
  attribute syn_preserve_sr_priority of async_set_reset : architecture is true;
begin
  setreset: process (clk, reset, set)
  begin
    if reset = '1' then
      qrs <= '0';
    elsif set = '1' then
      qrs <= '1';
    elsif rising_edge(clk) then
      qrs <= data;
    end if;
  end process setreset;
end async_set_reset;
**syn_props**

*Attribute; Triscend only.* Specifies Triscend attributes to forward-annotate to the gate-level netlist.

**.sdc File Syntax and Example**

```sdc
define_attribute {i:instance} syn_props {property=value}
```

For example:

```sdc
define_attribute {U0} syn_props {INITV=0x00ff}
```

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_props = "property=value" */;
```

where *object* is an instance.

```verilog
module romtest(A, out1, out2);
    input [4:0] A;
    output out1;
    output out2;


endmodule
```
**VHDL Syntax and Example**

```vhdl
attribute syn Props of object : object_type is "property=value";
```

where `object` is an instance and `object_type` is an instance.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library triscend;
use triscend.components.all;

entity vhdrom is port (A : in std_logic_vector(4 downto 0);
out1, out2 : out std_logic);
end vhdrom;

architecture beh of vhdrom is
attribute syn Props : string;
attribute syn Props of U0 : label is "INITV=0x00ff";
attribute syn Props of U1 : label is "INITV=0xFFFFFFFF";

begin
U0 : ROM16X1 port map (A3 => A(3), A2 => A(2), A1 => A(1), A0 => A(0), O => out1);
U1 : ROM32X1 port map (A4 => A(4), A3 => A(3), A2 => A(2), A1 => A(1), A0 => A(0), O => out2);
end beh;
```
**syn_radhardlevel**

*Attribute; Actel (RT, RH and RD radhard devices).* Specifies the radiation-resistant design technique to use on an object. This attribute can be applied to a module or architecture, or a register output signal (inferred register in VHDL), and must be one of the techniques available in the project. You can apply **syn_radhardlevel** globally to the top-level module or architecture of your design and then selectively override it on different portions of the design. You can also control the design technique to apply on individual registers.

Values for **syn_radhardlevel** are:

- **none** – standard design techniques are used.
- **cc** – combinatorial cells with feedback are used to implement storage rather than flip-flop or latch primitives.
- **tmr** – triple module redundancy or triple voting is used to implement registers. Each register is implemented by three flip-flops or latches that “vote” to determine the state of the register.
- **tmr_cc** – triple module redundancy is used where each voting register is composed of combinatorial cells with feedback rather than flip-flop or latch primitives.

Some techniques are not available or appropriate for all Actel families. Contact Actel technical support for more information.

**syn_radhardlevel** is only effective when the corresponding Actel Verilog (.v) or VHDL (.vhd) file for the specified design technique is included in the source files list of your Project. If you include more than one file, the first Actel file specified in the list determines the default design technique. You can add **syn_radhardlevel** to individual registers to override the default in the file.

For example, to use both **cc** and **tmr** techniques in a Verilog design, include the following files in the source files list of your Project.

- `synplify_installation_dir/lib/actel/cc.v`
- `synplify_installation_dir/lib/actel/tmr.v`

Then use **syn_radhardlevel** to choose between **cc**, **tmr**, and **none**. Where there is no attribute specified, the **cc** design technique applies, because it is the first Actel file in the list.
**.sdc File Syntax and Example**

```
define_attribute {object} syn_radhardlevel {none | cc | tmr | tmr_cc}
```

where `object` is a register. For example:

```
define_attribute {dataout[3:0]} syn_radhardlevel {cc}
```

**Verilog Syntax and Example**

```
object /* synthesis syn_radhardlevel ="none | cc | tmr | tmr_cc" */;
```

Where `object` is a module or a register output signal. For example:

```verilog
module top (clk, dataout, a, b);
input clk;
input a;
input b;
output dataout [3:0];
reg [3:0] dataout /* synthesis syn_radhardlevel="cc" */;

// Other code
```

**VHDL Syntax and Example**

```vhdl
attribute syn_radhardlevel of object : object_type is "none | cc | tmr | tmr_cc";
```

Where `object` is a module or a register output signal, `object_type` is architecture or signal. For example:

```vhdl
library synplify;
architecture top of top is
attribute syn_radhardlevel : string;
attribute syn_radhardlevel of top: architecture is "cc";

-- Other code
```
**syn_ramstyle**

*Attribute.* The `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. To turn off RAM inference, set the attribute to `registers`. You can apply `syn_ramstyle` to the instance name of a RAM.

The attribute is only available for certain technologies. The following table summarizes information about the values and implementations in the supported technologies.

Table 7-8: syn_ramstyle support

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Technology</th>
<th>Values</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera</td>
<td>Flex10K, ACEX</td>
<td>registers</td>
<td>Default: Small RAMs implemented as registers. Large RAMs implemented as EABs, if altera_auto_use_eab=1 or true.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block_ram</td>
<td>Registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EABs.</td>
</tr>
<tr>
<td>APEX</td>
<td>families</td>
<td>registers</td>
<td>Default: Small RAMs implemented as registers. Large RAMs implemented as ESBs, if altera_auto_use_esb=1 or true.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block_ram</td>
<td>Registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ESBs.</td>
</tr>
<tr>
<td>Stratix</td>
<td></td>
<td>registers</td>
<td>Default: EABs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block_ram</td>
<td>Registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ESBs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_rw_check</td>
<td>EABs without read/write address conflict (glue) logic.</td>
</tr>
<tr>
<td>Atmel</td>
<td>40K</td>
<td>registers</td>
<td>Default: RAMs implemented as LPMs (which use dedicated RAM cells)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RAMs implemented as registers.</td>
</tr>
</tbody>
</table>
The value for *syn_ramstyle* depends upon the vendor. These are the *syn_ramstyle* values:

- **registers** – specifies that an inferred RAM is to be mapped to registers (flip-flops and logic) and not technology-specific RAM resources. If your RAM resources are limited, for whatever reason, you can map additional RAMs to registers instead of RAM resources using this attribute.

- **block_ram** – specifies that the inferred RAM be mapped to the appropriate vendor-specific memory. It uses the dedicated memory resources in the FPGA. For Xilinx Virtex designs, it implements Block SelectRAM+ with additional glue logic to resolve read/write address conflicts.

- **no_rw_check** (Altera Stratix and Xilinx only). For Xilinx Virtex, this implements the Virtex Block SelectRAM+, without any glue logic. For Altera Stratix, it implements EABs without any glue logic. When you

---

Table 7-8: *syn_ramstyle* support (Continued)

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Technology</th>
<th>Values</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cypress</td>
<td>38K, 39K</td>
<td>registers</td>
<td>Default: RAMs implemented as LPMs (which use dedicated RAM cells) RAMs implemented as registers.</td>
</tr>
<tr>
<td>Lattice</td>
<td>Orca2, Orca3</td>
<td>registers</td>
<td>Default: RAMs implemented as synchronous dual-port memory cells. RAMs implemented as registers.</td>
</tr>
<tr>
<td>Xilinx</td>
<td>XC4000 families</td>
<td>registers</td>
<td>Registers.</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Virtex families</td>
<td>registers</td>
<td>Registers and combinatorial logic.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block_ram</td>
<td>Block SelectRAM plus read/write address conflict logic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_rw_check</td>
<td>Block SelectRAM without read/write address conflict (glue) logic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_ram</td>
<td>(Default) Distributed RAM</td>
</tr>
</tbody>
</table>
read and write to the same Block SelectRAM+ address, the value of the output is indeterminate. An indeterminate output can create a simulation mismatch between RTL and post-synthesis simulations. Without no_rw_check, the Synplify synthesis tool inserts bypass logic around the RAM to prevent the mismatch. If you know your design does not read and write to the same address simultaneously, use no_rw_check to eliminate bypass logic. Use this value only when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic.

• select_ram – (Xilinx only; default value). Implements RAMs using the distributed RAM resources in the CLBs. For distributed RAMs, the write operation must be synchronous and the read operation asynchronous. For dual port RAMs, the synthesis tool adds glue logic to direct the RAM input to go to the output when the read address is the same as the write address.

**.sdc File Syntax and Example**

```plaintext
define_attribute {signal_name[bit_range]} syn_ramstyle {string}
```

If you edit a constraint file to apply syn_ramstyle, be sure to include the range of the signal with the signal name. For example:

```plaintext
define_attribute {mem[7:0]} syn_ramstyle {registers};
```

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_ramstyle = "string" */;
```

where object can be register definition (reg) signals. The data type is string.

Here is an example:

```verilog
module ram4 (datain,dataout,clk);
output dataout[31:0];
input clk;
input datain[31:0];
reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="registers" */;

// Other code
```
**VHDL Syntax and Example**

```
attribute syn_ramstyle of object : object_type is "string";
```

where `object` can be signals that define RAMs, and labels of component instances. Data type is string.

**Signal Defining the RAM**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity ram4 is
  port (d : in std_logic_vector(7 downto 0);
        addr : in  std_logic_vector(2 downto 0);
        we : in  std_logic;
        clk : in  std_logic;
        ram_out : out std_logic_vector(7 downto 0));
end ram4;
```

```vhdl
library synplify;
architecture rtl of ram4 is
  type mem_type is array (127 downto 0) of std_logic_vector (7 downto 0);
  signal mem : mem_type;
  -- mem is the signal that defines the RAM
  attribute syn_ramstyle : string;
  attribute syn_ramstyle of mem : signal is "block_ram";
  -- Other code
```
**syn_reference_clock**

*Attribute.* The *syn_reference_clock* attribute lets you specify a clock frequency other than that implied by the signal on the clock pin of the register. For example, when flip-flops have an enable with a regular pattern, such as every second clock cycle, use *syn_reference_clock* to have timing analysis treat the flip-flops as if they were connected to a clock at half the frequency.

To use *syn_reference_clock*, define a new clock, then apply its name to the registers you want to change.

**.sdc File Syntax and Examples**

```sdc
define_attribute {register} syn_reference_clock {clk_name}
```

For example:

```sdc
define_attribute {myreg[31:0]} syn_reference_clock {sloClock}
```

You can also use *syn_reference_clock* to constrain multicycle paths through the enable signal. This example shows how you can apply the constraint to all registers with the specified enable signal:

```sdc
define_attribute {find -reg -enable en40} syn_reference_clock {clk2}
```
**syn_replicate**

*Attribute; Xilinx.* Disables replication in Xilinx designs. The Synplify synthesis tool automatically replicates registers during the following optimization processes: meeting maximum fanout requirements, packing I/Os, and improving quality of results.

You can use this attribute to disable replication either globally or on a per-register basis. When you disable replication globally, it disables I/O packing and quality of results optimizations. The synthesis tool uses only buffering to meet maximum fanout guidelines.

You can use this attribute to disable I/O packing on specific registers by setting the attribute to 0. Similarly, you can use it on a register between clock boundaries to prevent replication. For example, the synthesis tool would replicate 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. By setting the attribute for the register to 0, you can disable replication.

**.sdc File Syntax and Example**

```
define_global_attribute syn_replicate = {1 | 0}
```

For example, to disable all replication in the design:

```
define_global_attribute syn_replicate {0}
```

**Verilog Syntax and Example**

```
object /* synthesis syn_replicate = 1 | 0 */;
```

For example:

```
module norep (Reset, Clk, Drive, OK, ADPad, IPad, ADOut);
input Reset, Clk, Drive, OK;
input [31:0] ADOut;
inout [31:0] ADPad;
output [31:0] IPad;
```
reg [31:0] IPad;
reg DriveA /* synthesis syn_replicate = 0 */;
assign ADPad = DriveA ? ADOut : 32'b0;
always @(posedge Clk or negedge Reset)
    if (!Reset)
        begin
            DriveA <= 0;
            IPad   <= 0;
        end
    else
        begin
            DriveA <= Drive & OK;
            IPad   <= ADPad;
        end
endmodule

VHDL Syntax and Example

attribute syn_replicate of object : object_type is true | false ;

For example:

library IEEE;
use ieee.std_logic_1164.all;
entity norep is port (  
    Reset : in std_logic;
    Clk : in std_logic;
    Drive : in std_logic;
    OK : in std_logic;
    ADPad : inout std_logic_vector (31 downto 0);
    IPad : out std_logic_vector (31 downto 0);
    ADOut : in std_logic_vector (31 downto 0));
end norep;

architecture archnorep of norep is
signal DriveA : std_logic;
attribute syn_replicate : boolean;
attribute syn_replicate of DriveA : signal is false;
begin
ADPad <= ADOut when DriveA='1' else (others => 'Z');
process (Clk, Reset)
begin
if Reset='0' then
  DriveA <= '0';
  IPad <= (others => '0');
elsif rising_edge(clk) then
  DriveA <= Drive and OK;
  IPad <= ADPad;
end if;
end process;
end archnorep;
**syn_resources**

*Attribute; Altera Xilinx*. Specifies the resources available for black boxes. It is attached to Verilog black-box modules and VHDL architectures or component definitions. For Altera designs, use this attribute instead of the obsolete *altera_area*.

The value of the attribute is any combination of the following, where *luts* specifies the number of lookup tables, *regs* specifies the number of registers, and *blockrams* specifies the number of RAM resources:

\[
\begin{align*}
    \text{luts} &= \text{integer} \\
    \text{regs} &= \text{integer} \\
    \text{blockrams} &= \text{integer}
\end{align*}
\]

The value listed in the area usage report is the larger of the *luts* or *registers* value. For the Altera Apex families, the Synplify synthesis tool calculates the ATOM resources as the maximum of the larger of either the *luts* or *registers* value.

**.sdc File Syntax and Example**

```plaintext
define_attribute {v:module_name} syn_resources {value}
```

For example:

```plaintext
define_attribute {v:bb} syn_resources {luts=500, regs=400, blockrams=10}
```

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_resources = value */;
```

In Verilog, you can only attach this attribute to a module. Here is an example:

```verilog
module bb (o,i)
/* synthesis
   syn_black_box syn_resources = "Luts=500,Regs=463,Blockrams=10"
*/;
   input i;
   output o;
endmodule
```
module top_bb (o,i);
    input i;
    output o;
    bb u1 (o,i);
endmodule

**VHDL Syntax and Example**

```
attribute syn_resources of object : object_type is string;
```

In VHDL, this attribute can be placed on either an architecture or on a on a component declaration.

```
architecture top of top is
    component decoder
        port (clk : in bit;
            a, b : in bit;
            qout : out bit_vector(7 downto 0)
        );
    end component;

    attribute syn_resources : string;
    attribute syn_resources of decoder: component is
        "Luts=500,Regs=463,Blockrams=10";

    -- Other code
```
synRomStyle (Altera)

Attribute: Altera (APEX, ACEX, and FLEX10K families). You can describe a ROM structure in RTL code using a case statement. By applying the synRomStyle attribute to the signal output value, you can control if the ROM structure is implemented as discrete logic or as an ESB block for APEX or EAB blocks for FLEX and ACEX.

By default, small ROMs (less than seven address bits) are implemented as logic, and large ROMs (seven or more address bits) as ESBs for APEX or EABs for FLEX and ACEX.

You can globally disable automatic use of ESBs/EABs for inferred ROMs by setting altera_auto_use_eab = 0 (FLEX and ACEX) or altera_auto_use_esb = 0 (APEX).

Setting synRomStyle=logic in an APEX, ACEX, or FLEX device forces an implementation using discrete logic primitives. Setting synRomStyle=blockRom forces an ESB implementation when targeting APEX devices, and an EAB implementation when targeting FLEX and ACEX devices.

.sdc File Syntax and Example

define_attribute {romPrimitive} synRomStyle {logic | blockRom | lpm}

For example:

define_attribute {z_20[3:0]} synRomStyle {blockRom}

Verilog Syntax and Examples

object /* synthesis synRomStyle = "logic | blockRom | lpm" */;

The following example shows you how to use the synRomStyle attribute to implement a ROM structure as blockRom, which forces an ESB/EAB implementation.

reg [3:0] z /* synthesis synRomStyle = "blockRom" */;

The following example shows you how to use the synRomStyle attribute to implement a ROM structure as logic, which forces an implementation using discrete logic primitives.

reg [8:0] z /* synthesis synRomStyle = "logic" */;
**VHDL Syntax and Examples**

```vhdl
attribute syn_romstyle of object : object_type is "logic | block_rom | lpm";
```

The following example shows you how to use the `syn_romstyle` attribute to implement a ROM structure as `block_rom`, which will force an EAB/ESB implementation.

```vhdl
signal z : std_logic_vector(3 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "block_rom";

The following example shows you how to use the `syn_romstyle` attribute to implement a ROM structure as `logic`, which forces an implementation using discrete logic primitives.

```vhdl
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```

**Inferring ROMs (Examples)**

The following examples show you how to infer a ROM.

**Example: Inferring ROMs (Verilog)**

```verilog
// Sparsely populated ROM
module rom2(z, a);
output [3:0] z;
input [4:0] a;
reg [3:0] z /* synthesis syn_romstyle = "block_rom" */;
always @(a) begin
  case (a)
    5'b00000: z = 4'b1011;
    5'b00001: z = 4'b0001;
    5'b00100: z = 4'b0011;
    5'b00110: z = 4'b0010;
    5'b00111: z = 4'b1110;
    5'b01001: z = 4'b0111;
    5'b01010: z = 4'b0101;
    5'b01101: z = 4'b0100;
    5'b10000: z = 4'b1100;
    5'b10001: z = 4'b1101;
    5'b10010: z = 4'b1111;
    5'b10011: z = 4'b1110;
    5'b11000: z = 4'b1010;
  endcase
endmodule
```
Example: Inferring ROMs (VHDL)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity rom6 is
    port ( a: in std_logic_vector(3 downto 0);
           z: out std_logic_vector(3 downto 0) );
    attribute syn_romstyle : string;
    attribute syn_romstyle of z : signal is "block_rom";
end rom6;
architecture rtl of rom6 is
begin
    process(a)
    begin
        case a is
            when "0000" => z <= "1010";
            when "0001" => z <= "0100";
            when "0010" => z <= "0111";
            when "0011" => z <= "0010";
            when "0100" => z <= "0101";
            when "0101" => z <= "1001";
            when "0110" => z <= "1011";
            when "0111" => z <= "0011";
            when "1000" => z <= "1101";
            when "1001" => z <= "1111";
            when "1010" => z <= "1110";
            when "1011" => z <= "0110";
            when "1100" => z <= "1000";
            when "1101" => z <= "1110";
            when "1110" => z <= "1100";
            when others => z <= "0000";
        end case;
    end process;
end rtl;
```
syn_romstyle (Xilinx)

Attribute; Xilinx (XC4000 and Virtex families). Allows you to implement ROM architectures using distributed ROM. Infer ROM architectures using a CASE statement in your code.

For the Synplify synthesis tool to implement a ROM, at least half of the available addresses in the CASE statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses.

.sdc File Syntax and Example

The constraint file syntax for the attribute is:

```
define_attribute {rom_primitive} syn_romstyle {select_rom | logic | block_rom}
```

The following constraint file example takes the signal ROM_bit and implements it using Xilinx SelectROM.

```
define_attribute {ROM_bit[3:0]} syn_romstyle {select_rom}
```

Verilog Syntax and Examples

```
object /* syn_romstyle = "select_rom | logic | block_rom" */;
```

This example implements a ROM structure as select_rom, which forces a distributed ROM implementation.

```
reg [3:0] z /* synthesis syn_romstyle = "select_rom" */;
```

This example implements a ROM structure as logic, which forces an implementation using discrete logic primitives:

```
reg [8:0] z /* synthesis syn_romstyle = "logic" */;
```

This example implements a ROM structure as block RAM.

```
reg [8:0] z /* synthesis syn_romstyle = "block_rom" */;
```
**VHDL Syntax and Examples**

```
attribute syn_romstyle of object : object_type is "select_rom | logic | block_rom" ;
```

This example implements a ROM structure as select_rom.

```
signal z : std_logic_vector(3 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "select_rom";
```

This example implements a ROM structure as logic.

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```

This example implements a ROM structure as Xilinx block RAM. Refer to *Mapping ROM into Block RAM* on page H-13 for further details.

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "block_rom";
```
**syn_srlstyle**

*Attribute, Xilinx (Virtex families).* Determines how to implement the sequential shift (seqShift) components.

**.sdc File Syntax**

```
define_attribute {object} syn_srlstyle {registers | select_srl | noextractff_srl}
```

The `syn_srlstyle` values are:

- **registers** – implements seqShift components as registers.
- **select_srl** – implements seqShift components using 16-bit shift register lookup table primitives (SRL16) with flip-flops inserted ahead of the output buffers for improved timing.
- **noextractff_srl** – implements seqShift components using 16-bit shift register lookup table primitives (SRL16) without output flip-flops.

For example:

```
define_attribute {regBank[15:0]} syn_srlstyle {registers}
```

**Verilog Syntax and Example**

```
object /* synthesis syn_srlstyle = "select_srl | registers | noextractff_srl" */ ;
```

In the above syntax, `object` is a register declaration.

This example implements seqShift components as SRL16 without output flip-flops.

```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]

    /* synthesis syn_srlstyle="noextractff_srl" */;
    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

**VHDL Syntax and Example**

attribute syn_srlstyle of object : signal is "select_srl | registers | noextractff_srl";

In the above syntax, `object` is a register.

The example below implements seqShift components as SRL16 primitives without inserting timing improvement flip-flops between the SRL outputs and the output buffers.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity d_p is
  port (
    clk : in std_logic;
    data_out : out std_logic_vector(127 downto 0));
end d_p;
architecture rtl of d_p is
  type dataAryType is array(3 downto 0) of
    std_logic_vector(127 downto 0);
  signal h_data_pip_i : dataAryType;
  attribute syn_srlstyle : string;
  attribute syn_srlstyle of h_data_pip_i : signal
  is "noextractff_srl";
begin
  process (Clk)
  begin
    if (Clk'Event And Clk = '1') then
      h_data_pip_i <= (h_data_pip_i(2 DOWNTO 0)) &
syn_tristatetomux

**Attribute; Xilinx (XC4000 and Virtex families).** Forces each net to be analyzed in the design that is driven by tristate drivers and convert those drivers to a mux. This is done if the number of tristates drivers on the net is less than the value of the integer argument.

All drivers on the nets that could be converted to muxes must be tristate drivers, otherwise the conversion fails and a warning is issued.

**.sdc File Syntax and Example**

```plaintext
define_attribute {object} syn_tristatetomux {integer}
```

For example:

```plaintext
define_attribute {v:tristate2} syn_tristatetomux {4}
```

**Verilog Syntax and Example**

```plaintext
object /* synthesis syn_tristatetomux = integer*/;
```

For example:

```plaintext
module tristate2 (input0, input1, input2,
    input3, enable, qout)
    /* synthesis syn_tristatetomux = 4 */;
    input [7:0] input0, input1, input2, input3;
    input [3:0] enable;
    output [7:0] qout;
    wire [7:0] temp;
    assign temp = (enable[0]) ? input0 : 8'hZZ;
    assign temp = (enable[1]) ? input1 : 8'hZZ;
    assign temp = (enable[2]) ? input2 : 8'hZZ;
    assign temp = (enable[3]) ? input3 : 8'hZZ;
    assign qout = temp & input0;
endmodule
```
VHDL Syntax and Example

attribute syn_tristatetomux of object : object_type is integer;

Here is a VHDL example of multiple tristate drivers on a bus.

```vhdl
library ieee;
library synplify;
use ieee.std_logic_1164.all;
entity tristate2 is
  port (input3,
       input2,
       input1,
       input0 : in std_logic_vector (7 downto 0);
       enable : in std_logic_vector (3 downto 0);
       qout : out std_logic_vector (7 downto 0) );
end tristate2;
architecture multiple_drivers of tristate2 is
attribute syn_tristatetomux : integer;
attribute syn_tristatetomux of multiple_drivers :
  architecture is 4;
signal temp : std_logic_vector (7 downto 0);
beg
  temp <= input3 when enable(3) = '1'
    else "ZZZZZZZZ" ;
  temp <= input2 when enable(2) = '1'
    else "ZZZZZZZZ" ;
  temp <= input1 when enable(1) = '1'
    else "ZZZZZZZZ" ;
  temp <= input0 when enable(0) = '1'
    else "ZZZZZZZZ" ;
  qout <= temp and input0;
end multiple_drivers;
```
**syn_useenables**

*Attribute; Actel (54SX), Altera, Lattice, Xilinx.* Generates register instances with clock enable pins. By default, the Synplify synthesis tool tries to use the enable pin. You set *syn_useenables* to 0 to turn off clock-enable extraction.

**.sdc File Syntax and Example**

The constraint file syntax for the attribute is:

```
define attribute {register | signal} syn_useenables {0 | 1}
```

For example:

```
define_attribute {q[3:0]} syn_useenables {0}
```

**Verilog Syntax and Example**

```
object /* synthesis syn_useenables = 0 | 1 */;
```

where *object* can be a component register or signal. Data type is Boolean.

For example:

```
reg [3:0] q /* synthesis syn_useenables = 0 */;
always @(posedge clk)
  if (enable)
    q <= d;
```
**VHDL Syntax and Example**

```
attribute syn_useenables of object : object_type is true | false;
```

where `object` can be labels of component registers or signals.

For example:

```vhdl
signal q_int : std_logic_vector(3 downto 0);
attribute syn_useenables of q_int : signal is false;
... begin
... process(clk)
begin
  if (clk'event and clk = '1') then
    if (enable = '1') then
      q_int <= d;
    end if;
  end if;
end process;
```


**syn_useioff (Altera)**

*Attribute; Altera (APEX families).* Controls I/O register packing when targeting Apex families. `syn_useioff` is a boolean attribute. A value of 1 enables register packing, and 0 disables register packing. By default, `syn_useioff` is disabled (not enabled).

**Note:** With Altera Stratix, when this attribute is enabled, registers are not packed into Multiply/Accumulate (MAC) blocks.

You can place this attribute on individual ports, or apply it globally.

**.sdc File Syntax and Examples**

```sdc
define_attribute {port_name} syn_useioff {1 | 0}
define_global_attribute syn_useioff {1 | 0}
```

For example:

```sdc
define_attribute {z[3:0]} syn_useioff {1}
define_global_attribute syn_useioff {1}
```

**Verilog Syntax and Examples**

```verilog
object /* synthesis syn_useioff = 1 | 0 */;
```

To use this attribute globally, apply it to the module.

```verilog
module test (a, b, clk, rst, d) /* synthesis syn_useioff = 1 */;
```

To use this attribute on individual ports, apply it to individual port declarations.

```verilog
module test (a, b, clk, rst, d);
input a;
inout b /* synthesis syn_useioff = 1 */;
```
VHDL Syntax and Examples

attribute syn_useioff of object : object_type is true | false;

To use this attribute globally, apply it to the architecture.

Entity test is port ( 
   
   
   
); 
end test;

architecture archtest of test is 
   signal temp  : std_logic; 
   signal temp1 : std_logic; 
   signal temp2 : std_logic; 
   signal temp3 : std_logic; 

   attribute syn_useioff : boolean; 
   attribute syn_useioff of archtest : architecture is true; 

To use this attribute on individual ports, apply it to the ports in the entity section.

entity test is port ( 
   a : in std_logic; 
   
   
   
); 

   attribute syn_useioff : boolean; 
   attribute syn_useioff of a : signal is true; 
end test,
**syn_useioff (Lattice ORCA)**

*Attribute; Lattice ORCA (Orca3).* Controls I/O register packing when targeting the Orca Series3 family. `syn_useioff` is a Boolean attribute: 1 enables (default) and 0 disables register packing.

You can place this attribute on individual ports, or apply it globally. The attribute is supported in Verilog, VHDL, and .sdc file formats.

### .sdc File Syntax and Example

- `define_attribute {port_name} syn_useioff {0 | 1}`
- `define_global_attribute syn_useioff {0 | 1}`

For example:

```
define_attribute {z[3:0]} syn_useioff {1}
define_global_attribute syn_useioff {1}
```

### Verilog Syntax and Examples

*object synthesis syn_useioff = {1 | 0} */;

If you need to use this attribute globally, apply it to the module.

```
module test (a, b, clk, rst, d) /* synthesis syn_useioff = 1 */;
```

If you need to use this attribute on individual ports, apply it to individual port declarations.

```
module test (a, b, clk, rst, d);
input a;
input b /* synthesis syn_useioff = 1 */;
```
**VHDL Syntax and Examples**

```
attribute syn_useioff of object : object_type is true | false;
```

If you want to use this attribute globally, apply it to the architecture:

```
architecture archtest of test is
  signal temp : std_logic;
  signal temp1 : std_logic;
  signal temp2 : std_logic;
  signal temp3 : std_logic;

  attribute syn_useioff : boolean;
  attribute syn_useioff of archtest : architecture is true;
end archtest;
```

If you want to use this attribute on individual ports, apply it to the ports in the entity section.

```
entity test is port (a : in std_logic;
  .
  .
  .
);

  attribute syn_useioff : boolean;
  attribute syn_useioff of a : signal is true;
end test;
```
**syn_useioff (QuickLogic)**

*Attribute; Eclipse, QuickDSP, and QuickSD.* Controls I/O register packing when you target the listed QuickLogic families. `syn_useioff` is a Boolean attribute: 1 enables (default) and 0 disables register packing.

You can place this attribute on individual ports, or apply it globally. When applied globally, the Synplify synthesis tool packs all input, output, and I/O registers into pads. When applied to a port, the synthesis tool packs all flip-flops attached to the port into the I/O pad. You can enter the attribute in Verilog, VHDL, and `.sdc` file formats.

### `.sdc` File Syntax and Example

```plaintext
define_attribute {port_name} syn_useioff {0 | 1}
define_global_attribute syn_useioff {0 | 1}
```

For example:

```plaintext
define_attribute {z[3:0]} syn_useioff {1}
define_global_attribute syn_useioff {1}
```

### Verilog Syntax and Examples

```plaintext
object synthesis syn_useioff = {1 | 0} */ ;
```

If you need to use this attribute globally, apply it to the module.

```plaintext
module test (a, b, clk, rst, d) /* synthesis syn_useioff = 1 */;
```

If you need to use this attribute on individual ports, apply it to individual port declarations.

```plaintext
module test (a, b, clk, rst, d);
input a;
input b /* synthesis syn_useioff = 1 */;
```
**VHDL Syntax and Examples**

```
attribute syn_useioff of object : object_type is true | false;
```

If you want to use this attribute globally, apply it to the architecture.

```
architecture rtl of test is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```

The next example sets the attribute on a 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;
```
syn_useioff (Xilinx)

Attribute; Xilinx (Virtex, XC4000 and Spartan 2 families). Overrides the default behavior to pack I/Os based on timing preferences. When syn_useioff is set to 1 (or true), flip-flops are embedded in the IOBs.

.sdc File Syntax and Examples

```plaintext
define_attribute {port} syn_useioff {1 | 0}
define_global_attribute syn_useioff {1 | 0}
```

For example, to embed flip-flops locally in IOBs:

```plaintext
define_attribute {p:q[3:0]} syn_useioff {1}
```

To embed flip-flops globally:

```plaintext
define_global_attribute syn_useioff {1}
```

Verilog Syntax and Examples

```plaintext
object /* synthesis syn_useioff = 1 | 0 */;
```

This example sets the attribute on a port:

```plaintext
module test(d, clk, q);
input [3:0] d;
input clk;
output [3:0] q /* synthesis syn_useioff = 1 */;
reg q;
```

This example sets the attribute globally:

```plaintext
module test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

VHDL Syntax and Examples

```plaintext
attribute syn_useioff of object : object_type is true | false;
```

```plaintext
architecture rtl of test is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```
This example sets the attribute on a port:

```vhdl
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;
```

This example sets the attribute globally:

```vhdl
architecture rtl of test is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```
xc_alias

**Attribute; Xilinx (except Virtex).** Changes cell names in XNF writer.

```plaintext
define_attribute {cell_name} xc_alias {alias_name}
```

**Verilog Syntax and Example**

```plaintext
object /* synthesis xc_alias = "string" */ ;
```

For example:

```plaintext
module AND4 (O, IO, I1, I2, I3)
    /*synthesis syn_black_box xc_alias = "AND" */;
```

**VHDL Syntax and Example**

```plaintext
attribute xc_alias of object : object_type is "string" ;
```

For example:

```plaintext
component AND4
    port (O : out std_logic;
         I0, I1, I2, I3 : in std_logic);
end component;
attribute xc_alias of AND4 : component is "AND" ;
```
xc_clockbuftype

*Attribute; Xilinx (Virtex families).* Uses the Clock Delay Locked Loop primitive (CLKDLL) for a clock port. This is inferred as a buffer called BUFGDLL, which includes the CLKDLL primitive.

**.sdc File Syntax and Example**

```plaintext
define_attribute {port} xc_clockbuftype {BUFGDLL}
```

For example:

```plaintext
define_attribute {clk} xc_clockbuftype {BUFGDLL}
```

**Verilog Syntax and Example**

```plaintext
object /* synthesis xc_clockbuftype = "BUFGDLL" */;
```

For example:

```plaintext
module test_clkbuftype(d, clk, rst, q);
input [3:0] d;
input clk /* synthesis xc_clockbuftype = "BUFGDLL" */;
input rst;
output [3:0] q;

// Other code
```
VHDL Syntax and Example

```vhdl
attribute xc_clockbuftype of object : object_type is "BUFGDLL";
```

For example:

```vhdl
library ieee, synplify;
use ieee.std_logic_1164.all;
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 : string;
  attribute xc_clockbuftype of clk : signal is "BUFGDLL";
end test_clkbuftype;
```

EDIF Output Example

Applying this attribute causes the resultant EDIF output file to appear similar to the following:

```edif
(instance clk_ibuf (viewRef PRIM (cellRef BUFGDLL
  (libraryRef VIRTEX))
)
xc_fast

Attribute; Xilinx (XC4000 families). Use this attribute to make the transition time of the output driver fast. The default is for the transition time to be slow (see xc_slow on page 7-136).

The Synplify synthesis tool provides attributes that are passed into the XNF or EDIF netlist for Xilinx placement and routing. These attributes affect the input setup times and output transition times for your I/Os. The xc_fast attribute gets passed to the Xilinx I/O Block Parameter in the XNF or EDIF netlist called “FAST”.

The transition time of the output driver can be programmed to be either fast or slow for XC4000 devices. Use the xc_fast attribute to decrease the transition time for the output driver.

.sdc File Syntax and Example
The constraint file syntax for the attribute is:

```
define_attribute {output_port} xc_fast {1}
```

For example:
```
define_attribute {DATA0[5]} xc_fast {1}
```

Verilog Syntax and Example

```
object /* synthesis xc_fast = 0 | 1 */;
```

For example:
```
output [5:0] DATA0 /* synthesis xc_fast = 1 */;
```
VHDL Syntax and Example

    attribute xc_fast of object : object_type is true | false;

For example:

    entity prep2_2 is
        port (DATA0 : out std_logic_vector (5 downto 0);
             .
             .
             .);
    attribute xc_fast of DATA0 : signal is true;
    end prep2_2;
xc_fast_auto

Attribute; Xilinx (Virtex families). By default (xc_fast_auto value of 1), the Synplify synthesis tool infers the fast output buffers OBUF_F_24, OBUFT_F_24, and IOBUF_F_24. Use this global attribute to force slow buffers to be inferred. In HDL source code you specify xc_fast_auto for the top-level module or architecture. You can override this attribute on an individual basis by using the xc_padtype attribute.

The xc_fast_auto attribute has no affect on output ports specified with the xc_padtype attribute.

.sdc File Syntax and Example

define_global_attribute xc_fast_auto {0 | 1}

For example:

define_global_attribute xc_fast_auto {0};

Verilog Syntax and Example

object /* synthesis xc_fast_auto = 0 | 1 */;

For example:

module top (q, d, addr, we, clk) /*synthesis xc_fast_auto=0 */;

// Other code
**VHDL Syntax and Example**

```vhdl
attribute xc_fast_auto of object : object_type is false | true;
```

For example:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity top 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 beh of top is
attribute xc_fast_auto : boolean;
attribute xc_fast_auto of beh : architecture is false;

-- Other code
```
xc_global_buffers

Attribute: Xilinx. Controls the number of global buffers used in a design. The Synplify synthesis tool automatically adds global buffers for clock nets with high fanout. Use this attribute to specify a maximum number of buffers and restrict the amount of global buffer resources added. Also, if there is a black box in the design that has global buffers, you can use xc_global_buffers to prevent the synthesis tool from inferring clock buffers and exceeding the number of global resources. You can only specify this attribute through a constraint file (cannot be specified in HDL source code).

.sdc File Syntax and Example

```plaintext
define_global_attribute xc_global_buffers {maximum}
```

For example:

```plaintext
define_global_attribute xc_global_buffers {3}
```
xc_loc

*Attribute; Xilinx. Specifies the location (placement) of ports. Refer to the Xilinx databook for valid placement locations.*

You cannot use xc_loc to specify locations of slices of vector (bus) ports.

.sdc File Syntax and Example

```
define_attribute {port_design_name} xc_loc {placements}
```

The following example is a constraint file assignment of a pad location to all bits of an 5-bit bus.

```
define_attribute {DATA0[4:0]} xc_loc {P14,P12,P11,P5,P21}
```

Verilog Syntax and Example

```
object /* synthesis xc_loc = "placements" */ ;
```

For example:

```
module prep2_2 (...);
    input [4:0] DATA0 /* synthesis xc_loc = "P14,P12,P11,P5,P21" */;
endmodule
```

VHDL Syntax and Example

```attribute xc_loc of object : object_type is "placements" ;
```

For example:

```entity prep2_2 is
    port (DATA0 : in std_logic_vector (4 downto 0);
        ...
    );
attribute xc_loc of DATA0 : signal is "P14,P12,P11,P5,P21";
end prep2_2;
```
xc_map

Attribute; Xilinx (XC4000 and Virtex families). Specifies that a design unit is either an fmap, hmap, or lut primitive. This attribute is used only in conjunction with the xc_rloc and xc_uset attributes. See Specifying Relative Location on page H-25 for further information.

.sdc File Syntax and Example

    define_attribute {v:primitive_name} xc_map {fmap | hmap | lut}

For example:

    define_attribute {v:hmap_xor4} xc_map {fmap}

Verilog Syntax and Example

    object/* synthesis xc_map = "fmap | hmap | lut" */;

For example:

    module fmap_xor4(z, a, b, c, d) /* synthesis xc_map="fmap" */;

VHDL Syntax and Example

    attribute xc_map of object : object_type is "fmap | hmap | lut";

For example:

    architecture rtl of fmap_xor4 is
    attribute xc_map : string;
    attribute xc_map of rtl : architecture is "fmap";
xc_ncf_auto_relax

Attribute; Xilinx (XC4000, XC5200, Virtex families). This attribute controls the automatic relaxation of constraints that are forward-annotated to the .ncf file. A value of 0 disables this feature. The default value is 1. See also *Relaxing Forward-annotated Constraints on page 6-43*, for the use of this attribute.

**.sdc File Syntax and Example**

```
define_global_attribute xc_ncf_auto_relax { 0 | 1}
```

For example:

```
define_global_attribute xc_ncf_auto_relax {0}
```

**Verilog Syntax and Example**

```
* synthesis xc_ncf_auto_relax = 0 | 1 */
```

For example:

```
module test_relax (clk,rst,a,b,out1)
    /* synthesis xc_ncf_auto_relax = 0 */;
    // Other code
endmodule
```
VHDL Syntax and Example

\[
\text{attribute xc_ncf_auto_relax of object : object_type is false | true;}
\]

For example:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity test_relax is
  port(
    clk, rst, a, b : in std_logic;
    out1 : out std_logic
  );
end test_relax;

architecture behave of test_relax is
attribute xc_ncf_auto_relax : boolean;
attribute xc_ncf_auto_relax of behave : architecture is false;

-- Other code
end behave;
```
xc_nodelay

Attribute; Xilinx (XC4000, Spartan, Spartan-XL, Spartan2, Virtex and Virtex2 families). Removes the input delay that Xilinx inserts for flip-flops and latches. By default Xilinx inserts the input delay. The input delay is inserted in order to guarantee the hold time requirement on the flip-flops at the inputs.

The Synplify synthesis tool provides attributes that are passed into the EDIF/XNF netlist for the Xilinx place-and-route tool that affect the input setup times and output transition times for your I/Os. The xc_nodelay timing attribute gets passed to the Xilinx I/O Block parameter in the EDIF/XNF netlist called NODELAY.

.sdc File Syntax and Example

```
    define_attribute {input_port_name} xc_nodelay {1 | 0}
```

For example:

```
    define_attribute {din} xc_nodelay {1}
```
xc_padtype

Attribute; Xilinx (Virtex families). Specifies an I/O buffer standard. For example, applying AGP to an IBUF would result in the use of an IBUF_AGP instead of an IBUF primitive.

.sdc File Syntax and Examples

define_attribute {port} xc_padtype {buffers_standards}

Note the required underscore (_) between buffers and standards.

Examples:

define_attribute {a[3:0]} xc_padtype {IBUF_GTLP}
define_attribute {bidir[3:0]} xc_padtype {IOBUF_AGP}
define_attribute {q[3:0]} xc_padtype {OBUF_CTT}

Verilog Syntax and Example

object /* synthesis xc_padtype = "buffers_standards" */ ;

For example:

module test_padtype(a, b, clk, rst, en, bidir, q);
    input [3:0] a /* synthesis xc_padtype = "IBUF_AGP" */ , b;
    input clk, rst, en;
    inout [3:0] bidir /* synthesis xc_padtype = "IOBUF_CTT" */ ;
    output [3:0] q /* synthesis xc_padtype = "OBUF_F_12" */ ;

    // Other code
VHDL Syntax and Example

attribute xc_padtype of object : object_type is "buffers_standards";

library ieee, synplify;
use ieee.std_logic_1164.all;
entity test_padtype is
    port( a : in std_logic_vector(3 downto 0);
         b : in std_logic_vector(3 downto 0);
         clk, rst, en : in std_logic;
         bidir : inout std_logic_vector(3 downto 0);
         q : out std_logic_vector(3 downto 0)
    );
attribute xc_padtype : string;
attribute xc_padtype of a : signal is "IBUF_SSTL3_I";
attribute xc_padtype of bidir : signal is "IOBUF_HSTL_III";
attribute xc_padtype of q : signal is "OBUF_S_8";
xc_props

Attribute: Xilinx. Specifies Xilinx attributes to forward-annotate to the gate-level netlist.

.sdc File Syntax and Example

```
define_attribute {i:instance} xc_props {property=value}
```

For example:
```
define_attribute {i:RAM1} xc_props {INIT=FFFF}
```

Verilog Syntax and Example

```
object /* synthesis xc_props = "property=value" */ ;
```

For example:
```
RAM16X1S RAM1(...) /* synthesis xc_props = "INIT=0000" */;
```

VHDL Syntax and Example

```
attribute xc_props of object : object_type is "property=value" ;
```

For example:
```
attribute xc_props of RAM1 : label is "INIT=0000";
--RAM1 is the name of an instance
```
xc_pullup / xc_pulldown

Attribute; Xilinx (XC4000 and Virtex families). Specifies that a port is either a pull-up or pull-down.

.sdc File Syntax and Example

define_attribute {port_name} xc_pullup {1}
define_attribute {port_name} xc_pulldown {1}

For example:

define_attribute {data[3:0]} xc_pullup {1}
define_attribute {qrs[3:0]} xc_pulldown {1}

Verilog Syntax and Example

object /* synthesis xc_pulldown = 1 | 0 */ ;

For example:

module dff1(clk, reset, set, data, qrs);
  input clk, reset, set;
  input [3:0] data /* synthesis xc_pullup = 1 */;
  output [3:0] qrs /* synthesis xc_pulldown = 1 */;
  reg [3:0] qrs;

  // Other code
Attributes

VHDL Syntax and Example

attribute xc_pulldown of object : object_type is true | false ;

For example:

library ieee;
use ieee.std_logic_1164.all;
entity dff1 is
  port (clk, reset, set : in std_logic;
        data : in std_logic_vector(3 downto 0);
        qrs: out std_logic_vector(3 downto 0));
attribute xc_pullup : boolean;
attribute xc_pullup of data : signal is true;
attribute xc_pulldown of qrs : signal is true;
end dff1;
**xc_rloc**

*Attribute; Xilinx (XC4000, Virtex families).* Specifies the relative locations of all the instances specified with the same *xc_uset* value. This attribute is used only in conjunction with the *xc_uset* and *xc_map* attributes. See *Specifying Relative Location* on page H-25 for more information.

**.sdc File Syntax and Example**

```markdown
define_attribute {design_name} xc_rloc {instance_name}
```

Examples:

```markdown
define_attribute {x03} xc_uset {SET1}
define_attribute {x03} xc_rloc {R0C0.f}
define_attribute {x47} xc_uset {SET1}
define_attribute {x47} xc_rloc {R0C0.g}
define_attribute {zz} xc_uset {SET1}
define_attribute {zz} xc_rloc {R0C0.h}
```

**Verilog Syntax and Example**

```verilog
object /* synthesis xc_rloc = "instance_name" */;
```

For example:

```verilog
module clb_xor9(z, a);
output z;
input [8:0] a;
wire z03, z47;

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]);
endmodule
```
VHDL Syntax and Example

attribute xc_rloc of object : object_type is "instance_name" ;

For example:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity clb_xor9 is
  port ( a : in std_logic_vector(8 downto 0);
         z : out std_logic
       );
end clb_xor9;

architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component hmap_xor3
  port ( a : in std_logic;
         b : in std_logic;
         c : in std_logic;
         z : out std_logic
       );
end component;

component fmap_xor4
  port ( a : in std_logic;
         b : in std_logic;
         c : in std_logic;
         d : in std_logic;
         z : out std_logic
       );
end component;

attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";

begin
  x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
  x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
  zz : hmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```

```
**xc_slow**

*Attribute; Xilinx (XC4000 families)*. Specifies that the transition time of the driver of an output port is slow (the default). You can speed up the transition time with the *xc_fast* attribute.

The Synplify synthesis tool provides attributes that are passed into the XNF/EDIF netlist for Xilinx placement and routing that affect the input setup times and output transition times for your I/Os. The *xc_slow* timing attribute gets passed to the Xilinx I/O Block Parameter in the XNF/EDIF netlist called *SLOW*.

The transition time of the output driver can be programmed to either fast or slow for XC4000 devices. The *xc_slow* attribute increases the transition time which reduces the noise level in the circuit.

**.sdc File Syntax and Example**

```plaintext
define_attribute {output_port} xc_slow {1 | 0}

For example:

define_attribute {DATA0[5]} xc_slow {1}
```

**Verilog Syntax and Example**

```plaintext
object /* synthesis xc_slow = 1 | 0 */;

For example:

output [7:0] DATA0 /* synthesis xc_slow = 1 */;
```

**VHDL Syntax and Example**

```plaintext
attribute xc_slow of object : object_type is true;

For example:

attribute xc_slow of DATA0: signal is true;
```
**xc_uset**

*Attribute; Xilinx (XC4000, Virtex families).* Assigns a group name to component instances. This attribute is used with the `xc_rloc` and `xc_map` attributes to specify a relative location to a group of components instances. See *Specifying Relative Location* on page H-25 for more information.

**.sdc File Syntax and Example**

```
define_attribute {instance_name} xc_uset {group_name}
```

Examples:

```
define_attribute {x03} xc_uset {SET1};
define_attribute {x03} xc_rloc {ROCO.f};
define_attribute {x44} xc_uset {SET1};
define_attribute {x44} xc_rloc {ROCO.g};
define_attribute {zz} xc_uset {SET1};
define_attribute {zz} xc_rloc {ROCO.h};
```

**Verilog Syntax and Example**

```
object /* synthesis xc_uset = "group_name" */;
```

For example:

```
module clb_xor9(z, a);
    output z;
    input [8:0] a;
    wire z03, z47;
    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]);
endmodule
```
**VHDL Syntax and Example**

```
attribute xc_uset of object : object_type is "group_name";
```

For example:

```vhdl
library synplify;
architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component hmap_xor3
  port (a, b, c : in std_logic;
       z : out std_logic);
end component;
component fmap_xor4
  port (a, b, c : in std_logic;
       z : out std_logic);
end component;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";
begin
  x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
  x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
  zz : hmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```
Directives

The individual synthesis directives are described in this section, in alphabetical order. Each directive description includes the following:

- Technology support
- Directive definition
- File syntax and examples for Verilog and VHDL source code

For an alphabetical summary of the directives, see *Directive Summary (Alphabetical)* on page 7-7. For a summary of the directives by vendor, refer to the appropriate vendor chapter or see *Attribute and Directive Summary by Vendor* on page 7-3.

For a general information on specifying directives, see *Specifying Directives (and Attributes) in HDL* on page 7-14.
black_box_pad_pin

*Directive.* Specifies pins on a user-defined black-box component, as I/O pads that are visible to the environment outside of the black box. If there is more than one port that is an I/O pad, list the ports inside double-quotes separated by commas. (See Verilog and VHDL syntax and examples.)

You can use this directive with any of the following black-box directives:

- black_box_tri_pins
- syn_black_box
- syn_isclock
- syn_tco<n>
- syn_tpd<n>
- syn_tsu<n>

To instantiate an I/O from your programmable logic vendor, you usually do not need to define a black box and this attribute. The Synplify synthesis tool provides predefined black boxes for vendor I/Os. Refer to your vendor section under FPGA and CPLD Support for more information.

**Verilog Syntax and Example**

```verilog
/* synthesis syn_black_box black_box_pad_pin = "port_list" */ ;
```

For example:

```verilog
module BBDLHS(D, E, GIN, GOUT, PAD, Q)
    /* synthesis syn_black_box black_box_pad_pin="PAD" */ ;
```
**VHDL Syntax and Example**

```vhdl
attribute black_box_pad_pin of object : object_type is "port_list";
```

where *object* can be architectures or component declarations of black boxes. Data type is a string and *port_list* is the name or names of the ports on black boxes that are I/O pads (enclosed in double quotes). For example:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
package components is
  component BBDLHS
    port(
      D: in std_logic;
      E: in std_logic;
      GIN : in std_logic_vector(2 downto 0);
      Q : out std_logic
    );
  end component;

attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS : component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of BBDLHS : component is "GIN(2:0),Q";
end components;
```
black_box_tri_pins

Directive. Specifies that an output port, on a component defined as a black box, is tristate. This directive is used to eliminate multiple driver errors when the output of a black box has more than one driver. A multiple driver error is issued unless you use this directive to specify that the outputs are tristate. If there is more than one port that is a tristate, list all of the ports within double-quotes separated by commas.

You can use this directive with any of the following black-box directives:

- black_box_pad_pin
- syn_black_box
- syn_isclock
- syn_tco<n>
- syn_tpd<n>
- syn_tsu<n>

Verilog Syntax and Examples

```verilog
object /* synthesis syn_black_box black_box_tri_pins = "port_list" */ ;
```

For example:

```verilog
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
    /* synthesis syn_black_box black_box_tri_pins="PAD" */;
```

The `port_list` can be a single port name as shown in the previous example, or a list of multiple pins separated by commas.

```verilog
module bb1(D,E,tri1,tri2,tri3,Q)
    /* synthesis syn_black_box black_box_tri_pins="tri1,tri2,tri3" */;
```

For a bus, you must specify the port name followed by all the bits on the bus:

```verilog
module bb1(D,bus1,E,GIN,GOUT,Q)
    /* synthesis syn_black_box black_box_tri_pins="bus1[7:0]" */;
```
VHDL Syntax and Examples

attribute black_box_tri_pins of object : object_type is "port_list";

where object can be component declarations or architectures. Object_type is string and port_list is the name, or names of tristate output ports (enclosed in double quotes). For example:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
package components is
component BBDLHS
port(
    D: in std_logic;
    E: in std_logic;
    GIN : in std_logic;
    GOUT : in std_logic;
    PAD : inout std_logic;
    Q: out std_logic
);
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS : component is true;
attribute black_box_tri_pins : string;
attribute black_box_tri_pins of BBDLHS : component is "PAD";
end components;
```

Multiple pins on the same component can be specified in a list separated by commas as follows:

```vhdl
attribute black_box_tri_pins of bb1 : component is "tri,tri2,tri3";
```

To apply this attribute to a port which is a bus, you must specify all the bits on the bus:

```vhdl
attribute black_box_tri_pins of bb1 : component is "bus1[7:0]";
```
full_case

Directive. For Verilog designs only. When used with a case, casex, or casez statement, indicates that all possible values have been given, and that no additional hardware is needed to preserve signal values.

Verilog Syntax and Examples

```
object /* synthesis full_case */
```

where object can be case, casex, or casez statement declarations.

Suppose you want to use a casez statement to create a 4-input multiplexor with a pre-decoded select bus (a decoded select bus has exactly one bit enabled at a time) as follows:

```
module muxnew1 (out, a, b, c, d, select);
    output out;
    input a, b, c, d;
    input [3:0] select;
    reg out;

    always @ (select or a or b or c or d)
    begin
        casez (select)
            4’b??1?: out = a;
            4’b??1?: out = b;
            4’b?1??: out = c;
            4’b1????: out = d;
        endcase
    end
endmodule
```

Notice you are not telling the Synplify synthesis tool what to do if the select bus has all zeros. If the select bus is being driven from outside the current module, the current module has no information about the legal values of select, and the synthesis tool must preserve the value of the output out when all bits of select are zero. Preserving the value of out requires the tool to add extraneous level-sensitive latches if out is not assigned elsewhere through every path of the always block. A warning message similar to the following is issued:

"Latch generated from always block for signal out, probably missing assignment in branch of if or case."
To instruct the synthesis tool not to preserve the value of `out` when all bits of `select` are zero, it is easiest to use a default in the `casez` with an assignment of `'bx` (a `'bx` in an assignment is treated as a “don’t care”). This default assignment takes place every pass through the `casez` statement in which the select bus does not match one of the explicitly given values; therefore no extraneous level-sensitive latches are needed.

By using a default assignment, you stay within the Verilog language, and avoid using a synthesis directive to get the desired hardware. The technique of assigning a `'bx` in the default is also beneficial for simulation because if the select becomes invalid, your debugging will uncover the output being assigned a `'bx`.

```-verilog
module muxnew2 (out, a, b, c, d, select);
  output out;
  input a, b, c, d;
  input [3:0] select;
  reg out;

  always @(select or a or b or c or d)
  begin
    casez (select)
      4'b???1: out = a;
      4'b??1?: out = b;
      4'b?1??: out = c;
      4'b1???: out = d;
      default: out = 'bx;
    endcase
  end
endmodule
```
Use the `full_case` directive to achieve the same effect as the `default` case, and instruct the synthesis tool not to preserve the value of `out` when all bits of `select` are zero.

```verilog
module muxnew3 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)
begin
  casez (select) /* synthesis full_case */
    4'b???1: out = a;
    4'b??1?: out = b;
    4'b?1??: out = c;
    4'b1???: out = d;
  endcase
end
endmodule
```

If the select bus is decoded in the same module as the `case` statement, the synthesis tool automatically determines that all possible values are specified, and the `full_case` directive is unnecessary. Also using the `full_case` directive can lead to mismatches between pre- and post-synthesis simulation because the simulator does not use this directive.
parallel_case

Directive. For Verilog designs only. Forces a parallel multiplexed structure rather than a priority encoded structure. This is useful because case statements are defined to work in priority order; defined to execute the first (and only the first) statement with a tag that matches the select value.

If the select bus is being driven from outside the current module, the current module has no information about the legal values of select, and a chain of disabling logic must be created so that a match on a statement’s tag disables all following statements. If, for example, you know that the only legal values of select are 4'b1000, 4'b0100, 4'b0010, and 4'b0001, then only one of the tags can be matched at a time, and tag matching logic can be parallel and independent instead of chained.

You specify the directive as a comment immediately following the select value of the case statement.

Verilog Syntax and Example

\[ \text{object} /* \text{synthesis parallel_case} */ \]

where object can be applied to case, casex, or casez statement declarations.

```verilog
module muxnew4 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)
begin
    casez (select) /* synthesis parallel_case */
        4'b??1?: out = a;
        4'b??1?: out = b;
        4'b?1??: out = c;
        4'b1???: out = d;
        default: out = 'bx;
    endcase
end
endmodule
```
If the select bus is decoded within the same module as the `case` statement, the parallelism of the tag matches is automatically determined, and the `this` directive is unnecessary.
**syn_black_box**

*Directive.* Specifies that a module or component is a black box with only its interface defined for synthesis. The contents of a black box cannot be optimized during synthesis. A module can be a black box whether or not it is empty. This directive has a Boolean value, implied to be 1 or true. Common uses of `syn_black_box` include the following:

- Vendor primitives and macros (including I/Os).
- User-designed macros whose functionality is defined in a schematic editor, IP, or another input source in which the place-and-route tool merges design netlists from different sources.

To instantiate vendor I/Os and other vendor macros, you usually do not need to define a black box since the Synplify synthesis tool provides pre-defined black boxes for the vendor macros.

You can use this directive with any of the following black-box attributes:

- `black_box_pad_pin`
- `black_box_tri_pins`
- `syn_isclock`
- `syn_tco<n>`
- `syn_tpd<n>`
- `syn_tsu<n>`

See *Instantiating Black Boxes in VHDL* on page 9-86 for more information on black boxes.

**Verilog Syntax and Example**

```verilog
object /* synthesis syn_black_box */;
```

where `object` can be module declarations. For example:

```verilog
module bl_box(out,data,clk) /* synthesis syn_black_box */;
// Other code
```
**VHDL Syntax and Example**

```
attribute syn_black_box of object : object_type is true ;
```

where `object` can be component declarations, labels of instantiated components to be defined as black boxes, architectures, and components. Data type is Boolean.

```
library synplify;
architecture top of top is

component ram4
  port (myclk : in bit;
       opcode : in bit_vector(2 downto 0);
       a, b : in bit_vector(7 downto 0);
       rambus : out bit_vector(7 downto 0)
  );
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of ram4 : component is true;

-- Other code
```
syn_enum_encoding

**Directive.** For VHDL designs. Defines how enumerated data types are implemented. The type of implementation affects the performance and device utilization.

If FSM Compiler is enabled, do not use this directive to specify the encoding styles of extracted state machines; use the **syn_encoding** attribute instead. However, if you have enumerated data types and you turn off the FSM Compiler so that no state machines are extracted, the **syn_enum_encoding** style is implemented in the final circuit. See **syn_encoding Versus syn_enum_encoding** on page 7-154 for more information.

Values for **syn_enum_encoding** can be:

- **default** – automatically assigns an encoding style based on the number of states:
  - sequential for 0-4 enumerated types
  - onehot for 5-24 enumerated types
  - gray for >24 enumerated types
- **sequential** – More than one bit of the state register can change at a time, but because more than one bit can be hot the value must be decoded to determine the state. For example,
  
  000, 001, 010, 011, 100

- **onehot** – Only two bits of the state register change (one goes to '0', and one goes to '1') and only one of the state registers is hot (driven by a '1') at a time. For example,
  
  0000, 0001, 0010, 0100, 1000

- **gray** – Only one bit of the state register changes at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example,
  
  000, 001, 011, 010, 110

A message is displayed in the log file when you use the **syn_enum_encoding** directive, for example,

```
@N:"c:\design\..":17:11:17:12|Using onehot encoding for type mytype (red="10000000")
```
**Effect of Encoding Styles**

The following figure provides an example of two versions of a design: one using the default encoding style, the other overriding the default with the `syn_enum_encoding` directive for an enumerated data type defining a set of eight colors.

![Diagram showing two versions of a design](image)

*syn_enum_encoding = "default"
Based on 8 states, onehot assigned*

```
sel[2:0] [2:0] [2:0] [0:2] [0:2] color[0:2]
```

*color[0:2]*

*syn_enum_encoding = "sequential"*

Figure 7-5: `syn_enum_encoding`

In this example, using the default value for `syn_enum_encoding`, onehot is assigned because there are 8 states in this design. The onehot style implements the output `color` as 8 bits wide and creates decode logic to convert the input `sel` to the output. Using sequential for `syn_enum_encoding`, the logic is reduced to a buffer. The size of output `color` is 3 bits.

See the following section *VHDL Syntax and Examples* for the source code used to generate the schematics in Figure 7-5.
VHDL Syntax and Examples

```vhdl
attribute syn_enum_encoding of object : object_type is "value";
```

Where `object` can be enumerated types and `value` can be: default, sequential, onehot, gray.

Here is the code used to generate the second schematic in Figure 7-5. (The first schematic will be generated instead, if "sequential" is replaced by "onehot" as the `syn_enum_encoding` value.)

```vhdl
package testpkg is
type mytype is (red, yellow, blue, green, white,
violet, indigo, orange);
attribute syn_enum_encoding : string;
attribute syn_enum_encoding of mytype : type is "sequential";
end package testpkg;

library IEEE;
use IEEE.std_logic_1164.all;
use work.testpkg.all;
entity decoder is
port(
    sel : in std_logic_vector(2 downto 0);
    color : out mytype
);
end decoder;
architecture rtl of decoder is
begin
    process(sel)
    begin
        case sel is
            when "000" => color <= red;
            when "001" => color <= yellow;
            when "010" => color <= blue;
            when "011" => color <= green;
            when "100" => color <= white;
            when "101" => color <= violet;
            when "110" => color <= indigo;
            when others => color <= orange;
        end case;
    end process;
end rtl;
```

Here is an example using “gray” for data type `state`.

library synplify;
package my_states is
  type state is (Xstate, st0, st1, st2, st3, st4, st5, st6, st7,
  st8, st9, st10, st11, st12, st13, st14, st15);

  -- Define the attribute as a gray encoding style
  attribute syn_enum_encoding : string;
  attribute syn_enum_encoding of state : type is "gray";

end my_states;

**syn_encoding Versus syn_enum_encoding**

To implement a state machine with a particular encoding style when the FSM Compiler is enabled, you need to use the syn_encoding attribute. This attribute affects how the technology mapper implements state machines in the final netlist. The syn_enum_encoding directive only affects how the compiler interprets the associated enumerated data types. Therefore, the encoding defined by syn_enum_encoding is *not propagated* to the implementation of the state machine.

However, when FSM Compiler is disabled, the value of syn_enum_encoding is implemented in the final circuit.
**syn_isclock**

*Directive.* Specifies an input port on a black box is a clock.

Use the `syn_isclock` directive to specify that an input port on a black box is a clock, even though its name does not correspond to one of the recognized names. It will connect it to a clock buffer if appropriate. The data type is Boolean. You can use this directive with any of the following black-box attributes:

- `black_box_pad_pin`
- `black_box_tri_pins`
- `syn_black_box`
- `syn_tco<n>`
- `syn_tpd<n>`
- `syn_tsu<n>`

**Verilog Syntax and Examples**

```verilog
object/* synthesis syn_isclock = 0 | 1 */;
```

where `object` is input ports on black boxes.

```verilog
module ram4 (myclk,out,opcode,a,b) /* synthesis syn_black_box */;
output [7:0] out;
input myclk /* synthesis syn_isclock = 1 */;
input [2:0] opcode;
input [7:0] a, b;

//Other code
```
VHDL Syntax and Examples

\[ \text{attribute syn_isclock of object object_type is true;} \]

where \( \text{object} \) can be black-box input ports.

Example:

```vhdl
library synplify;

entity ram4 is
  port (myclk : in bit;
        opcode : in bit_vector(2 downto 0);
        a, b : in bit_vector(7 downto 0);
        rambus : out bit_vector(7 downto 0));
  attribute syn_isclock : boolean;
  attribute syn_isclock of myclk: signal is true;
  -- Other code
```

```
**syn_keep**

*Directive.* Keeps the specified net intact during optimization. This directive preserves a net throughout synthesis. When you use this attribute, the compiler places a temporary *keep* buffer primitive on the net as a placeholder throughout synthesis. You can view this buffer in the schematic views (see *Effects of Using syn_keep* on page 7-158 for an example). However the buffer is not part of the final netlist, so no extra logic is generated.

As a result of optimization, the compiler might remove some nets, although it maintains ports, registers, and instantiated components. If a net needs to be preserved for simulation results or to obtain a different synthesis implementation, use *syn_keep* on the net.

There are other situations that might require you to use *syn_keep*. For example, use this attribute to prevent duplicate cells from being merged during optimization.

You can also use *syn_keep* as a placeholder to apply the *-through* option of the *define_multicycle_path* or *define_false_path* timing constraint allowing you to specify a unique path as a multicycle or false path. Apply the constraint to the *keep* buffer.

Do not apply *syn_keep* to a *reg* or signal that will become a sequential object.

---

**Note:** With Altera Stratix, a *syn_keep* between the adder and the multiplier of a Multiply/Accumulate (MAC) block will prevent inferencing of *altmult_add* and *altmult_accum*, but not plain multipliers.
**Effects of Using syn_keep**

The following figure shows the technology view for two versions of a design. One version shows `syn_keep` set on two registers, `out1` and `out2`, to prevent sharing. The other version is without `syn_keep`.

![With syn_keep](image1.png)

![Without syn_keep](image2.png)

Figure 7-6: `syn_keep` directive used to prevent sharing

`syn_keep` was applied at the input of the registers to obtain registered outputs for `out1` and `out2`. Without `syn_keep`, `out1` and `out2` optimize to one register. See the following HDL syntax and example sections for the source code used to generate the schematics in the figure above.

**Usage Compared: syn_keep, syn_preserve, syn_noprune**

This comparison may help you to keep the three directives `syn_keep`, `syn_preserve` and `syn_noprune` straight:

- `syn_keep` assures that 1) a wire will be kept during synthesis and 2) no optimizations will cross the wire. It is usually used to break unwanted optimizations and to assure manually created replicas. It works only on nets and combinational logic.
- `syn_preserve` assures that registers are not optimized away.
• syn_noprune assures that black boxes are not optimized away, unless the output is driven by them.

Verilog Syntax and Example

```verilog
object /* synthesis syn_keep = 1 */ ;
```

where `object` can be `wire` or `reg` declarations. Make sure that there is a space between the object name and the beginning of the comment slash, or the Synplify synthesis tool will not honor the attribute.

Here is the source code used to produce the results shown in Effects of Using `syn_keep` on page 7-158

```verilog
module example2(out1, out2, clk, in1, in2);
output out1, out2;
input clk;
input in1, in2;
wire and_out;
wire keep1 /* synthesis syn_keep=1 */;
wire keep2 /* synthesis syn_keep=1 */;
reg out1, out2;
assign and_out=in1&in2;
assign keep1=and_out;
assign keep2=and_out;
always @(posedge clk)begin
  out1<=keep1;
  out2<=keep2;
end
endmodule
```
VHDL Syntax and Example

attribute syn_keep of object : object_type is true;

where object can be single and multiple-bit signals.

Here is the source code used to produce the schematics shown in Effects of Using syn_keep on page 7-158.

```vhdl
entity example2 is
  port (  
    in1, in2 : in bit;  
    clk : in bit;  
    out1, out2 : out bit  
  );
end example2;

architecture rt1 of example2 is
attribute syn_keep : boolean;
signal and_out, keep1, keep2 : bit;
attribute syn_keep of keep1, keep2 : signal is true;
begin
  and_out <= in1 and in2;
  keep1 <= and_out;
  keep2 <= and_out;
  process(clk)
  begin
    if (clk'event and clk = '1') then
      out1 <= keep1;
      out2 <= keep2;
    end if;
  end process;
end rt1;
```


**syn_macro**

*Directive; QuickLogic only.* Prevents instantiated macros from being merged or otherwise optimized away. This directive can only be placed in your HDL source code and not defined in the SCOPE spreadsheet.

**Verilog Syntax and Example**

```verilog
object/* synthesis syn_macro = 1 | 0 */;
```

For example:

```verilog
multipler_module
  decoder_macro1 (out, in, clk_in) /* synthesis syn_macro=1 */,
  decoder_macro2 (out, in, clk_in) /* synthesis syn_macro=1 */,
  decoder_macro3 (out, in, clk_in) /* synthesis syn_macro=1 */;
```

**VHDL Syntax and Example**

```vhdl
attribute syn_macro of object : object_type is true;
```

For example:

```vhdl
architecture top of top is
  component decoder_macro
    port (clk : in bit;
          opcode : in bit_vector(2 downto 0);
          a : in bit_vector(7 downto 0);
          data0 : out bit_vector(7 downto 0))
    );
  end component;

  -- The components u1, u2, u3, and u4 are instantiations
  -- of the decoder_macro which are specified later the body of
  -- this architecture.
  attribute syn_macro : boolean;
  attribute syn_macro of u1 : label is true;
  attribute syn_macro of u2 : label is true;
  attribute syn_macro of u3 : label is true;
  attribute syn_macro of u4 : label is true;

  -- Other code
```
**syn_noprune**

*Directive.* Prevents instance optimization for black-box modules (including technology-specific primitives) with unused output ports. During optimization, if a module does not drive any logic, it is removed by the Synplify synthesis tool. If you want to keep the black-box instance of the module in the design, use the `syn_noprune` directive.

Effects of using `syn_noprune`

The following figure shows the technology view for two versions of a design: one version using `syn_noprune` on black-box instance U1, and one without `syn_noprune`.

![Figure 7-7: syn_noprune directive used to prevent instance optimization](image)

With `syn_noprune`, module U1 remains in the design. Without `syn_noprune` the module is optimized away. See the following HDL syntax and example sections for the source code used to generate the schematics in this figure.
Usage Compared: syn_keep, syn_preserve, syn_noprune

This comparison may help you to keep the three directives syn_keep, syn_preserve and syn_noprune straight:

- **syn_keep** assures that 1) a wire will be kept during synthesis and 2) no optimizations will cross the wire. It is usually used to break unwanted optimizations and to assure manually created replications. It works only on nets and combinational logic.

- **syn_preserve** assures that registers are not optimized away.

- **syn_noprune** assures that black boxes are not optimized away, unless the output is driven by them.

.sdc File Syntax and Example

```plaintext
define_attribute {module | instance} syn_noprune {0|1}
```

For example:

```plaintext
define_attribute {u101} syn_noprune {1}
```

Verilog Syntax and Examples

```plaintext
object /* synthesis syn_noprune = 1 */;
```

where `object` can be a module declaration or an instance. The data type is Boolean.

The following example shows the source code used for the schematics in Figure 7-7.

```plaintext
module top(a1,b1,c1,d1,y1,clk);

output y1;
input a1,b1,c1,d1;
input clk;
wire x2,y2;
reg y1;
syn_noprune u1(a1,b1,c1,d1,x2,y2) /* synthesis syn_noprune=1 */;

always @(posedge clk)
  y1<= a1;
endmodule
```
In this example, `syn_noprune` can be applied in two places, on the module declaration of `syn_noprune` or in the top-level instantiation. The most common place to use `syn_noprune` is in the declaration of the module. By placing it here, all instances of the module are protected.

```
module syn_noprune (a,b,c,d,x,y); /* synthesis syn_noprune=1 */;

// Other code
```

Here is an example of using `syn_noprune` on black-box instances. If your design uses multiple instances with a single module declaration, the synthesis comment must be placed before the comma (,) following the port list for each of the instances.

```
my_design my_design1(out,in,clk_in) /* synthesis syn_noprune=1 */;
my_design my_design2(out,in,clk_in) /* synthesis syn_noprune=1 */;
```

In this example, only the instance `my_design2` will be removed if the output port is not mapped.

```
my_design
  my_design1 (out, in, clk_in) /* synthesis syn_noprune=1 */,
  my_design2 (out, in, clk_in),
  my_design3 (out, in, clk_in) /* synthesis syn_noprune=1 */;
```

**VHDL Syntax and Example**

```
attribute syn_noprune of object : object_type is true | false ;
```

where the data type is boolean, and `object` can be architectures, components or labels of instantiated components. See *Architectures on page 7-166*, *Component Declaration on page 7-166*, *Component Instance on page 7-166*, for details of the objects.

The following example shows the source code used for the schematics in Figure 7-7 on page 7-162.
library ieee;
use ieee.std_logic_1164.all;
entity noprune is
  port (a, b, c, d : in std_logic;
        x, y : out std_logic);
end noprune;
architecture behave of noprune is
begin
  -- Empty architecture represents a black box.
end behave;

library ieee;
use ieee.std_logic_1164.all;
entity top is
  port (a1, b1 : in std_logic;
        c1, d1, clk : in std_logic;
        y1 : out std_logic);
end;
architecture behave of top is
component noprune
  port (a, b, c, d : in std_logic;
        x, y : out std_logic);
end component;
signal x2, y2 : std_logic;
attribute syn_noprune : boolean;
attribute syn_noprune of u1 : label is true;
begin
  u1: noprune port map(a1, b1, c1, d1, x2, y2);
  process begin
    wait until (clk = '1') and clk'event;
    y1 <= a1;
  end process;
end;
Architectures

The `syn_noprune` attribute is normally associated with the names of architectures. Once it is associated, any component instantiation of the architecture (design unit) is protected from being deleted.

```vhdl
library synplify;
architecture mydesign of rtl is

attribute syn_noprune : boolean;
attribute syn_noprune of mydesign : architecture is true;

-- Other code
```

Component Declaration

Here is an example:

```vhdl
architecture top_arch of top is
component gsr
  port (gsr : in std_logic);
end component;

attribute syn_noprune : boolean;
attribute syn_noprune of gsr: component is true;
```

See *Instantiating Black Boxes in VHDL* on page 9-86, for more information.

Component Instance

The `syn_noprune` attribute works the same on component instances as with a component declaration.

```vhdl
architecture top_arch of top is
component gsr
  port (gsr : in bit);
end component;

attribute syn_noprune : boolean;
attribute syn_noprune of ul_gsr: label is true;
```
**syn_preserve**

*Directive.* Prevents sequential optimization such as constant propagation, inverter push-through and FSM extraction.

---

**Note:** With Altera Stratix, when this directive is enabled, registers are not packed into Multiply/Accumulate (MAC) blocks.

---

Use `syn_preserve` when you need to keep registers for simulation purposes or when you want to preserve the logic of registers driven by constant ‘1’ or ‘0’. To preserve the associated flip-flop and to prevent optimization of the signal, you can set `syn_preserve` on individual registers or on the module/architecture so that the directive is applied to all registers in the module. For example, assume that the input of a flip-flop is always driven to the same value, such as logic ‘1’. The Synplify synthesis tool ties that signal to VCC and removes the flip-flop. Using `syn_preserve` on the registered signal prevents the removal of the flip-flop.

Another use for this attribute is to preserve a particular state machine. When you enable the symbolic FSM compiler for your entire design and state-machine optimizations are performed, you can use `syn_preserve` to retain a particular state machine during optimization.
Effects of using syn_preserve

The following figure shows an example where reg1 and out2 are preserved during optimization using syn_preserve.

Figure 7-8: How syn_preserve retains registers during optimization

Without syn_preserve, reg1 and reg2 are shared because they are driven by the same source. out2 obtains the result of the AND of reg2 and NOT reg1. This is equivalent to the AND of reg1 and NOT reg1 which is a ‘0’. As this is a constant, register out2 is also removed, and output out2 is always ‘0’.

When registers are removed during synthesis, a warning message is displayed in the log file. For example,

@W:...Register bit out2 is always 0, optimizing ...

See the HDL syntax and example sections below for the source code used to generate the schematics in the figure above.

Usage Compared: syn_keep, syn_preserve, syn_noprune

This comparison may help you to keep the three directives syn_keep, syn_preserve and syn_noprune straight:

- syn_keep assures that 1) a wire will be kept during synthesis and 2) no optimizations will cross the wire. It is usually used to break
unwanted optimizations and to assure manually created replications. It works only on nets and combinational logic.

- syn_preserve assures that registers are not optimized away.
- syn_noprune assures that black boxes are not optimized away, unless the output is driven by them.

Verilog Syntax and Examples

```
object /* synthesis syn_preserve = 0 | 1 */ ;
```

where object can be register definition signals or modules.

In the following example, syn_preserve is used on a registered signal so that the flip-flop is not removed and optimization does not occur across the register. This is useful when you are not finished with the design but want to synthesize to find the area utilization.

```
reg foo /* synthesis syn_preserve = 1 */;
```

Following is an example of using syn_preserve for a state register.

```
reg [3:0] curstate /* synthesis syn_preserve = 1 */;
```

The following example shows the source code used for the schematics in Figure 7-8 on page 7-168.

```
module syn_preserve (out1,out2,clk,in1,in2)
    /* synthesis syn_preserve=1 */;

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

    reg out1;
    reg out2;
    reg reg1;
    reg reg2;

    always@ (posedge clk)begin
        reg1 <= in1 & in2;
```
syn_preserve Directives

VHDL Syntax and Examples

```vhdl
attribute syn_preserve of object : object_type is true;
```

where `object` can be output ports and internal signals that hold the value of state registers or architectures.

```vhdl
library ieee, synplify;
use ieee.std_logic_1164.all;
entity simpledff is
  port (q : out std_logic_vector(7 downto 0);
        d : in std_logic_vector(7 downto 0);
        clk : in std_logic);
  -- Turn on flip-flop preservation for the q output
  attribute syn_preserve : boolean;
  attribute syn_preserve of q : signal is true;
end simpledff;
architecture behavior of simpledff is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      -- Notice the continual assignment of "11111111" to q.
      q <= (others => '1');
    end if;
  end process;
end behavior;
```

In this example, `syn_preserve` is used on the signal `curstate` that is later used in a state machine to hold the value of the state register.

```vhdl
architecture behavior of mux is
begin
  signal curstate : state_type;
  attribute syn_preserve of curstate : signal is true;
  -- Other code
```
The following example shows the source code for the schematics in Figure 7-8 on page 7-168.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity mod_preserve is
  port ( 
    out1 : out std_logic;
    out2 : out std_logic;
    in1,in2,clk : in std_logic 
  );
end mod_preserve;
architecture behave of mod_preserve is
attribute syn_preserve : boolean;
attribute syn_preserve of behave: architecture is true;
signal reg1 : std_logic;
signal reg2 : std_logic;
begin process begin
  wait until clk'event and clk = '1';
  reg1 <= in1 and in2;
  reg2 <= in1 and in2;
  out1 <= not ( reg1);
  out2 <= (not (reg1) and reg2) ;
end process;
end behave;
```
syn_sharing

Directive. Enables/disables the resource sharing of operators inside a module during synthesis. Values for syn_sharing are 1 or 0, for Verilog, and true or false, for VHDL. By default, the attribute is enabled (value 1 for Verilog, true for VHDL). If the Resource Sharing check box in the Project view (see Action Buttons and Options on page 3-97) is disabled, you can still enable resource sharing using the syn_sharing directive.

Verilog Syntax and Example

```
object /* synthesis syn_sharing = 1 | 0 */ ;
```

where `object` can be module definitions.

```
module my_design(out,in,clk_in) /* synthesis syn_sharing=0 */;

// Other code
```

VHDL Syntax and Example

```
attribute syn_sharing of object : object_type is "true | false" ;
```

where `object` can be architecture names.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity alu is
    port ( a, b : in std_logic_vector (7 downto 0);
        opcode: in std_logic_vector (1 downto 0);
        clk: in std_logic;
        result: out std_logic_vector (7 downto 0) );
end alu;
architecture behave of alu is
    begin
    -- Behavioral source code for the design goes here.
end behave;
```
syn_state_machine

*Directive.* Enables/disables state-machine optimization on individual state registers in the design. If you disable FSM Compiler such that the state-machines in your design *are not* automatically extracted, but you would like some of them extracted, you can use this directive on just those individual state-registers. Conversely, if FSM Compiler is enabled, but there are state machines in your design that you do not want extracted, you can use `syn_state_machine` to override extraction on just those individual state registers.

Also, when FSM Compiler is enabled, all state machines are usually detected during synthesis. However, on occasion there are cases in which certain state machines are not detected. You can use this directive to declare those undetected registers as state machines.

The following figure shows an example of two implementations of a state machine: one with `syn_state_machine` enabled, the other with the directive disabled.

![syn_state_machine example](image)

Figure 7-9: syn_state_machine directive example
See the following HDL syntax and example sections below for the source code used to generate the schematics in the figure above.

See also:

- `syn_encoding` on page 7-62 for information on overriding default encoding styles for state machines.
- For VHDL designs, `syn_encoding Versus syn_enum_encoding` on page 7-154 for usage information about these two directives.

**Verilog Syntax and Examples**

```verilog
object /* synthesis syn_state_machine = 0 | 1 */;
```

where `object` can be state registers. Data type is Boolean: 0 does not extract an FSM, 1 extracts an FSM.

Following is an example of `syn_state_machine` applied to register `OUT`.

```verilog
module prep3 (CLK, RST, IN, OUT);
input CLK, RST;
input [7:0] IN;
output [7:0] OUT;
reg [7:0] OUT;
reg [7:0] current_state /* synthesis syn_state_machine=1 */;

// Other code
```

Here is the source code used for the example in Figure 7-9 on page 7-173.

```verilog
module FSM1 (clk, in1, rst, out1);
input clk, rst, in1;
output [2:0] out1;

`define s0 3'b000
`define s1 3'b001
`define s2 3'b010
`define s3 3'bxxx

reg [2:0] out1;
reg [2:0] state /* synthesis syn_state_machine = 1 */;
```
reg [2:0] next_state;

always @(posedge clk or posedge rst)
  if (rst) state <= `s0;
  else state <= next_state;

// Combined Next State and Output Logic
always @(state or in1)
case (state)
  `s0  : begin
    out1 <= 3'b000;
    if (in1) next_state <= `s1;
    else    next_state <= `s0;
  end
  `s1  : begin
    out1 <= 3'b001;
    if (in1) next_state <= `s2;
    else    next_state <= `s1;
  end
  `s2  : begin
    out1 <= 3'b010;
    if (in1) next_state <= `s3;
    else    next_state <= `s2;
  end
  default : begin
    out1 <= 3'bxxx;
    next_state <= `s0;
  end
endcase
endmodule

VHDL Syntax and Examples

attribute syn_state_machine of object : object_type is true|false ;

where object can be signals that hold the values of the state machines. For example:

attribute syn_state_machine of current_state: signal is true;

Following is the source code used for the example in Figure 7-9 on page 7-173.
library ieee;
use ieee.std_logic_1164.all;
entity FSM1 is
  port(
    clk,rst,in1 : in std_logic;
    out1 : out std_logic_vector (2 downto 0)
  );
end FSM1;
architecture behave of FSM1 is
type state_values is ( s0, s1, s2,s3 );
signal state, next_state: state_values;
attribute syn_state_machine : boolean;
attribute syn_state_machine of state : signal is false;
begin
  process (clk, rst)
  begin
    if rst = '1' then
      state <= s0;
    elsif rising_edge(clk) then
      state <= next_state;
    end if;
  end process;
  process (state, in1) begin
    case state is
    when s0 =>
      out1 <= "000";
      if in1 = '1' then next_state <= s1;
      else next_state <= s0;
      end if;
    when s1 =>
      out1 <= "001";
      if in1 = '1' then next_state <= s2;
      else next_state <= s1;
      end if;
    when s2 =>
      out1 <= "010";
      if in1 = '1' then next_state <= s3;
      else next_state <= s2;
      end if;
    when others =>
      out1 <= "XXX"; next_state <= s0;
    end case;
  end process;
end behave;
**syn_tco<n>**

*Directive.* Supplies clock-output timing-delay information through a black box. You can use `syn_tco` with any of the following black-box directives:

- `black_box_pad_pin`
- `black_box_tri_pins`
- `syn_black_box`
- `syn_isclock`
- `syn_tpd<n>`
- `syn_tsu<n>`

**Verilog Syntax and Example**

```verilog
object /* syn_tco<n> = "[!]clock->bundle = value" */ ;
```

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are assumed to be in ns.

```
"[!]clock->bundle = value"
```

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

An example defining `syn_tco<n>` with other black-box constraints follows:

```verilog
module ram32x4 (z,d,addr,we,clk);
/* synthesis syn_black_box syn_tco1="clk->z[3:0]=4.0"
    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
```
**VHDL Syntax and Examples**

```vhdl
attribute syn_tco of object : object_type is "[!]clock -> bundle = value";
```

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are assumed to be in ns.

```
"[!]clock -> bundle = value"
```

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

There are 10 instances of each of these timing constraints available, for example: `syn_tco1`, `syn_tco2`, ..., `syn_tco10`. If you need more than 10, you can declare the desired amount (start with an integer greater than 10), for example:

```vhdl
attribute syn_tco11 : string;
attribute syn_tco12 : string;
```

In addition to the syntax used in the code below, you can also use the following Verilog-style syntax to specify this attribute:

```vhdl
attribute syn_tco of inputfifo_coregen : component is
   "rd_clk->dout[48:0]=3.0";
```

Following is an example of assigning `syn_tco<n>` along with some of the other black-box constraints:

```vhdl
architecture top of top is
component rcf16x4z port (
   ad0, ad1, ad2, ad3 : in std_logic;
   di0, di1, di2, di3 : in std_logic;
   clk, wren, wpe : in std_logic;
   tri : in std_logic;
   do0, do1, do2, do3 : out std_logic);
end component;
```
attribute syn_tco1 of rcf16x4z : component is 
  "clk -> do0,do1 = 4.0";
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 -> clk = 1.2";
attribute syn_tsu2 of rcf16x4z : component is 
  "wren,wpe -> clk = 0.0";

-- Other code
**syn_tpd<n>**

*Directive.* Supplies information on timing propagation for combinatorial delay through a black box.

You can use *syn_tpd* with any of the following black-box directives:

- `black_box_pad_pin`
- `black_box_tri_pins`
- `syn_black_box`
- `syn_isclock`
- `syn_tco<n>`
- `syn_tsu<n>`

**Verilog Syntax and Example**

```verilog
object /* syn_tpd = "![clock] -> bundle = value" */;
```

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are assumed to be in ns.

```
"![clock] -> bundle = value"
```

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is `A,B,C` which lists three signals.

Following is an example of defining `syn_tpd<n>` along with some of the other black-box timing constraints:

```verilog
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
```
VHDL Syntax and Examples

attribute syn_tpd\textsubscript{n} of object : object_type is "[!]clock -> bundle = value";

There are 10 instances of each of these timing constraints available, for example: syn_tpd1, syn_tpd2, ... syn_tpd10.

If you need more than 10, you can declare the desired amount (start with an integer greater than 10), for example:

attribute syn_tpd11 : string;
attribute syn_tpd12 : string;

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are assumed to be in ns.

"[!]clock -> bundle = value"

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is A,B,C which lists three signals.

In addition to the syntax used in the code below, you can also use the following Verilog-style syntax to specify this attribute:

attribute syn_tpd1 of inputfifo_coregen : component is "rd_clk->dout[48:0]=3.0";

Following is an example of assigning syn\textsubscript{tpd\textsubscript{n}} along with some of the of the black-box constraints:

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
component rcf16x4z port (  
ad0, ad1, ad2, ad3 : in std_logic;
di0, di1, di2, di3 : in std_logic;
clk, 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 -> clk = 1.2"
attribute syn_tsu2 of rcf16x4z : component is
   "wren,wpe -> clk = 0.0"

-- Other code
**syn_tristate**

*Directive.* Specifies that an output port, on a module defined as a black box, is tristate. Use this directive to eliminate multiple driver errors if the output of a black box has more than one driver. A multiple driver error is issued unless you use this directive to specify that the outputs are tristate.

**Verilog Syntax and Examples**

```verilog
object /* synthesis syn_tristate = 0 | 1 */;
```

where *object* can be black-box output ports. For example:

```verilog
module BUFE(O, I, E); /* synthesis syn_black_box */
    output O /* synthesis syn_tristate = 1 */;

    // Other code
```
**syn_tsu<n>**

*Directive.* Supplies information on timing setup delay required for input pins (relative to the clock) in a black box. You can use `syn_tsu` with any of the following black-box directives:

- `black_box_pad_pin`
- `black_box_tri_pins`
- `syn_black_box`
- `syn_isclock`
- `syn_tco<n>`
- `syn_tpd<n>`

### Verilog Syntax and Example

```
object /* syn_tsu<n> = 
    "[!]clock -> bundle = value" */;
```

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are assumed to be in ns.

```
"[!]clock -> bundle = value"
```

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is `A,B,C` which lists three signals.

Following is an example of defining `syn_tsu<n>` along with some of the other black-box constraints:

```verilog
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
```
VHDL Syntax and Examples

```vhdl
attribute syn_tsun of object : object_type is "[!]clock -> bundle = value";
```

There are 10 instances of each of these timing constraints available, for example: `syn_tsu1`, `syn_tsu2`, ... `syn_tsu10`.

If you need more than 10, you can declare the desired amount (start with an integer greater than 10), for example:

```vhdl
attribute syn_tsu11 : string;
attribute syn_tsu12 : string;
```

A bundle is a collection of buses and scalar signals. To assign values to bundles, use the following syntax. The values are assumed to be in ns.

```
"[!]clock -> bundle = value"
```

The optional exclamation mark (!) indicates a negative edge for a clock. The objects of a bundle must be separated by commas with no spaces between. A valid bundle is `A,B,C` which lists three signals.

In addition to the syntax used in the code below, you can also use the following Verilog-style syntax to specify this attribute:

```vhdl
attribute syn_tsu1 of inputfifo_coregen : component is
    "rd_clk->dout[48:0]=3.0";
```

Following is an example of assigning `syn_tsu<n>` along with some of the other black-box constraints:
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
  component rcf16x4z port (
    ad0, ad1, ad2, ad3 : in std_logic;
    di0, di1, di2, di3 : in std_logic;
    clk, wren, wpe : in std_logic;
    tri : in std_logic;
    do0, do1, do2, do3 : out std_logic);
end component;

attribute syn_tco1 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 -> clk = 1.2";
attribute syn_tsu2 of rcf16x4z : component is
  "wren,wpe -> clk = 0.0";

-- Other code
**translate_off/translate_on**

*Directive.* Used for compatibility with synthesis tools from other vendors than Synplicity, `translate_off` and `translate_on`, allow you to synthesize designs originally written for use with other synthesis tools, without needing to modify source code. All source code that is between these two directives is ignored during synthesis.

Another use of these directives is to prevent the synthesis of stimulus source code that only has meaning for logic simulation. You can use `translate_off/translate_on` to skip over simulation-specific lines of code that are not synthesizable.

When you use `translate_off` in a module, the Synplify synthesis tool halts the synthesis of all source code that follows until `translate_on` is encountered. Every `translate_off` must have a corresponding `translate_on`. These directives cannot be nested, therefore, the `translate_off` directive can only be followed by a `translate_on` directive.

**Verilog Syntax and Example**

For Verilog designs, you can use the `synthesis` macro with the Verilog `ifdef` directive instead of the `translate_on/off` directives. See *synthesis Macro* on page 8-45 for information.

The Verilog syntax for these directives is as follows:

```verilog
/* synthesis translate_off */

/* synthesis translate_on */
```

For example:

```verilog
module adder8(cout, sum, a, b, cin);

    // Place code here that you WANT synthesized
    /* synthesis translate_off */

    // Place code here that you DO NOT want synthesized.
```
translate_off/translate_on

Directives

/* synthesis translate_on */

    // Place code here that you WANT synthesized.

endmodule

VHDL Syntax and Example

synthesis translate_off

synthesis translate_on

For example:

architecture behave of ram4 is
begin

    -- synthesis translate_off
    stimulus: process (clk, a, b)
    
    -- Source code you DO NOT want synthesized

end process;
    -- synthesis translate_on

    -- Other source code you WANT synthesized
xc_isgsr

Directive; Xilinx (XC4000 families). Specifies that a port on a black box is connected to an internal STARTUP block. Use this directive with designs targeting XC4000 families where the Synplify synthesis tool infers a STARTUP block.

.sdc File Syntax and Example

```plaintext
define_attribute {instance.reset_port} xc_isgsr {1 | 0}
```

For example:

```plaintext
define_attribute {bbgsr.gsrin} xc_isgsr {1}
```

Verilog Syntax and Example

```plaintext
object/* synthesis xc_isgsr = {1|0} */;
```

For example:

```plaintext
module bbgsr(gsrin, a, b); /* synthesis syn_black_box */
input gsrin /* synthesis xc_isgsr = 1 */;
// Other code
```
**VHDL Syntax and Example**

```vhdl
attribute xc_isgsr of object : object_type is true | false;

For example:

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

entity bbgsr is
    port( gsrin : in std_logic;
         b: in std_logic;
         a : out std_logic
    );

attribute xc_isgsr : boolean;
attribute xc_isgsr of gsrin : signal is true;
end bbgsr;

architecture bb of bbgsr is
attribute syn_black_box : boolean;
attribute syn_black_box of bb : architecture is true;

-- Other code
```
CHAPTER 8

Verilog Language Support

This chapter discusses Verilog support in the Synplify synthesis tool, including the following topics:

- Language Constructs on page 8-2
- Verilog Synthesis Guidelines on page 8-7
- Verilog Module Template on page 8-9
- Scalable Modules on page 8-11
- Built-in Gate Primitives on page 8-14
- Combinational Logic on page 8-15
- Sequential Logic on page 8-19
- Verilog State Machines on page 8-29
- RAM Inference on page 8-34
- Instantiating Black Boxes in Verilog on page 8-40
- PREP Verilog Benchmarks on page 8-42
- Hierarchy: Structural Verilog on page 8-43
- Synthesis Attributes and Directives on page 8-47
Language Constructs

The following sections define the supported, unsupported, and ignored language constructs for Verilog.

Supported Verilog Language Constructs

The Synplify synthesis tool supports the following Verilog constructs:

- Net types: wire, tri, supply1, supply0; register types: reg, integer, time (64 bit reg); arrays of reg.
- Continuous assignments.
- Gate primitive and module instantiations.
- always blocks, user tasks, user functions.
- inputs, outputs, and inouts to a module.
- All operators (+, -, *, /, %, <, >, <=, >=, ==, !=, &&, ||, !, ~, &, ~&, |,
  ^, ^=, &^, |^, <<=, >>=, ?:, {}, {{}) [Note: / and % are supported for compile-time constants and constant powers of 2.]
- Procedural statements: if-else-if, case, casex, casez, for, repeat, while, forever, begin, end, fork, join.
- Procedural assignments: blocking assignments =, nonblocking assignments <= (Note: <= cannot be mixed with = for the same register. We recommend parameter override: # and defparam (down one level of hierarchy only).
- Compiler directives: `define, `ifdef, `else, `endif, `include, `undef
- Miscellaneous:
  - Integer ranges and parameter ranges.
  - Local declarations to begin-end block.
  - Variable indexing of bit vectors on the left and right sides of assignments.
Unsupported Language Constructs

The Synplify synthesis tool does not support the following Verilog constructs. If found, it generates an error message and halts.

• Net types: trireg, wor, trior, wand, triand, tri0, tri1, and charge strength; register type: real.
• Built-in unidirectional and bidirectional switches, and pull-up, pull-down.
• Procedural statements: assign, deassign, , wait.
• Named events and event triggers.
• UDPs and specify blocks.
• force, release, and hierarchical net names (for simulation only).

Ignored Language Constructs

The Synplify synthesis tool ignores the following Verilog constructs and continues the synthesis run.

• delay, delay control, and drive strength.
• scalared, vectored.
• initial block.
• Compiler directives (except for `define, `ifdef, `else, `endif, `include, and `undef, which are supported).
• Calls to system tasks and system functions (they are only for simulation).

Verilog 2001 Support

You can choose which Verilog standard to use for a project or given files within a project: Verilog '95 or Verilog 2001 (see File Options Dialog Box on page 3-78 and VHDL and Verilog Panels on page 3-36). The Synplify synthesis tool supports the following Verilog 2001 features:
Table 8-1: Supported Verilog 2001 features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combined data and port types</td>
<td>You can combine module data and port type declarations, for conciseness.</td>
</tr>
<tr>
<td>Comma-separated sensitivity list</td>
<td>Commas are allowed as separators in sensitivity lists (as in other Verilog lists).</td>
</tr>
<tr>
<td>Wildcards in sensitivity list</td>
<td>You can use @* or @(*) to include all signals in a procedural block, eliminating mismatches between RTL and post-synthesis simulation.</td>
</tr>
<tr>
<td>Signed arithmetic expressions</td>
<td>Data types net and reg, module ports, integers of different bases and signals can all be signed. Signed signals can be assigned and compared. Signed operations can be performed for vectors of any length.</td>
</tr>
<tr>
<td>Power (exponent) operator (**)</td>
<td>Implemented as a left shift. Only base 2 is supported.</td>
</tr>
<tr>
<td>Generate</td>
<td>You use it to create multiple instances of an object in a module. You can use generate with loops and conditional statements.</td>
</tr>
</tbody>
</table>

Example: Combined Data, Port Types (ANSI C-style Modules)

Verilog '95

```verilog
module adder_16 (sum, cout, cin, a, b);
output [15:0] sum;
output cout;
input [15:0] a, b;
input cin;

reg [15:0] sum;
reg cout;
wire [15:0] a, b;
wire cin;
```
Verilog 2001

    module adder_16(output reg [15:0] sum;
                  output reg cout;
                  input wire cin;
                  input wire [15:0] a, b);

Example: Comma-separated Sensitivity List

Verilog '95

always @(a or b or cin)
    sum = a + b + cin;

    always @(posedge clock or negedge reset)
        if (!reset)
            q <= 0;
        else
            q <= d;

Verilog 2001

    always @(a, b or cin)
        sum = a + b + cin;

    always @(posedge clock, negedge reset)
        if (!reset)
            q <= 0;
        else
            q <= d;

Example: Wildcard (*) in Sensitivity List

Verilog '95

    always @(a or b or cin)
        sum = a + b + cin;
Verilog 2001

// One style
always @(*)
    sum = a + b + cin;

// Another style
always @*
    sum = a + b + cin;

Examples: Signed Signals (Verilog 2001)

Declaration
module adder
(output reg signed [31:0] sum,  
wire signed input [31:0] a, b;

Assignment
wire signed [3:0] a = 4'sb1001;

Comparison
wire signed [1:0] sel;
parameter p0 = 2'sb00, p1 = 2'sb01,  
p2 = 2'sb10, p3 = 2'sb11;
    case sel
        p0: ...
        p1: ...
        p2: ...
        p3: ...
    endcase

Examples: Generate Statement (Verilog 2001)

// for loop
generate
genvar i;
    for (i=0; i<=7; i=i+1)
      begin :inst
        adder8 add (sum [8*i+7 : 8*i], c0[i+1],  
        a[8*i+7 : 8*i], b[8*i+7 : 8*i], c0[i]);
      end
endgenerate
Verilog Synthesis Guidelines

General Synthesis Guidelines

Some general guidelines are presented here to help you synthesize your Verilog design. See Verilog Module Template on page 8-9 for additional information.

- Top-level module – The Synplify synthesis tool always picks the last module compiled that is not referenced in another module as the top-level module. Module selection can be overridden from the Verilog panel of the Options for Implementation dialog box.

- Simulate your design before synthesis – you should simulate your design before synthesis to expose logic errors. Logic errors that you do not catch are passed through the synthesis tool, and the synthesized results will contain the same logic errors.

- Simulate your design after placement and routing – Have the place-and-route tool generate a post placement and routing (timing-accurate) simulation netlist, and do a final simulation before programming your devices.

```verilog
// if-else
generate
  if (adder_width < 8)
    ripple_carry # (adder_width) u1 (a, b, sum);
  else
    carry_look_ahead # (adder_width) u1 (a, b, sum);
endgenerate
// case
parameter WIDTH=1;
generate
case (WIDTH)
  1: adder1 x1 (c0, sum, a, b, ci);
  2: adder2 x1 (c0, sum, a, b, ci);
  default: adder # width (c0, sum, a, b, ci);
endcase
endgenerate
```
Chapter 8: Verilog Language Support

Verilog Synthesis Guidelines

- Avoid asynchronous state machines – To use the synthesis tool for asynchronous state machines, make a netlist of technology primitives from your target library.
- Level-sensitive latches – For modeling level-sensitive latches, use continuous assignment statements.

Verilog Language Guidelines

always Blocks

An always block can have more than one event control argument, provided they are all edge-triggered events or all signals; these two kinds of arguments cannot be mixed in the same always block.

Examples

```verilog
// OK: Both arguments are edge-triggered events
always @(posedge clk or posedge rst)
// OK: Both arguments are signals
always @(A or B)
// No good: One edge-triggered event, one signal
always @(posedge clk or rst)
```

An always block represents either sequential logic or combinational logic. The one exception is that you can have an always block that specifies level-sensitive latches and combinational logic. This is not a recommended style because it is easy to have user errors or unwanted level-sensitive latches.

An event expression with `posedge` / `negedge` keywords implies edge-triggered sequential logic; and without `posedge` / `negedge` keywords implies combinational logic, a level-sensitive latch, or both.

Each sequential always block is triggered from exactly one clock (and optional sets and resets).

You must declare every signal assigned a value inside an always block as a reg or integer. An integer is a 32-bit quantity by default, and is used with the Verilog operators to do two’s complement arithmetic. The Synplify synthesis tool also supports range specifications for integers, allowing you to create an integer size other than 32 bits.
Verilog Module Template

Syntax:

    integer [msb:lsb] identifier;

Avoid combinational loops in `always` blocks. Make sure all signals assigned in a combinational `always` block are explicitly assigned values every time the `always` block executes, otherwise the synthesis tool needs to insert level-sensitive latches in your design to hold the last value for the paths that do not assign values. This is a common source of errors, so the tool issues a warning message that latches are being inserted into your design. You will get an error message if you have combinational loops in your design that are not recognized as level-sensitive latches by the synthesis tool (for example if you have an asynchronous state machine).

It is illegal to have a given bit of the same `reg` or `integer` variable assigned in more than one `always` block.

Assigning a 'bx to a signal is interpreted as a “don't care” (there is no 'bx value in hardware); the synthesis tool then creates the hardware with the most efficient design.

Verilog Module Template

Hardware designs can include combinational logic, sequential logic, state machines, and memory. These elements are described in the Verilog module. You also can create hardware by directly instantiating built-in gates into your design (in addition to instantiating your own modules).

Within a Verilog module you can describe hardware with one or more continuous assignments, `always` blocks, module instantiations, and gate instantiations. The order of these statements within the module is irrelevant, and all execute concurrently. The following is the Verilog module template:

    module <top_module_name>(<port list>);

    /* Port declarations. followed by wire,
       reg, integer, task and function declarations */

    /* Describe hardware with one or more continuous assignments,
       always blocks, module instantiations and gate instantiations */
// Continuous assignment
wire <result_signal_name>;
assign <result_signal_name> = <expression>;

// always block
always @(event expression)
begin
    // Procedural assignments
    // if statements
    // case, casex, and casez statements
    // while, repeat and for loops
    // user task and user function calls
end

// Module instantiation
<module_name> <instance_name> (<port list>);

// Instantiation of built-in gate primitive
gate_type_keyword (<port list>);

endmodule

The statements between the begin and end statements in an always block execute sequentially from top to bottom. If you have a fork-join statement in an always block, the statements within the fork-join execute concurrently.

You can add comments in Verilog by preceding your comment text with // (two forward slashes). Any text from the slashes to the end of the line is treated as a comment, and is ignored by the Synplify synthesis tool. To create a block comment, which can span any number of lines, start the comment with /* (forward slash followed by asterisk) and end the comment with */ (asterisk followed by forward slash). Block comments cannot be nested.
Scalable Modules

Creating a Scalable Module

You can create a Verilog module that is scalable, so that it can be stretched or shrunk to handle a user-specified number of bits in the port list buses.

Declare parameters with default parameter values. The parameters can be used to represent bus sizes inside a module.

Syntax

```
parameter parameter_name = value;
```

You can define more than one parameter per declaration by using comma-separated `parameter_name = value` pairs.

Example

```
parameter size = 1;
parameter word_size = 16, byte_size = 8;
```

Using Scalable Modules

To use scalable modules, instantiate the scalable module and then override the default parameter value with the `defparam` keyword. Give the instance name of the module you are overriding, the parameter name, and the new value.

Syntax

```
defparam instance_name.parameter_name = new_value;
```

Example

```
big_register my_register (q, data, clk, rst);
defparam my_register.size = 64;
```
Combine the instantiation and the override in one statement. Use a # (hash mark) immediately after the module name in the instantiation, and give the new parameter value. To override more than one parameter value, use a comma-separated list of new values.

**Syntax**

```
module_name # (new_values_list) instance_name (port_list);
```

**Example**

```
big_register #(64) my_register (q, data, clk, rst);
```

**Creating a Scalable Adder**

```
module adder(cout, sum, a, b, cin);

    /* Declare a parameter, and give a default value */
    parameter size = 1;
    output cout;

    /* Notice that sum, a, and b use the value of the size parameter */
    output [size-1:0] sum;
    input [size-1:0] a, b;
    input cin;
    assign {cout, sum} = a + b + cin;
endmodule
```

**Scaling by Overriding a Parameter Value with defparam**

```
module adder8(cout, sum, a, b, cin);

    output cout;
    output [7:0] sum;
    input [7:0] a, b;
    input cin;
    adder my_adder (cout, sum, a, b, cin);

    // Creates my_adder as an eight bit adder
    defparam my_adder.size = 8;
endmodule
```
Scaling by Overriding the Parameter Value with #

module adder16(cout, sum, a, b, cin);
output cout;

/* You can define a parameter at this level of hierarchy, and
pass that value down to the lower level instance. In this
example, we declare a parameter called my_size. Note: you
can declare a parameter with the same name as the lower
level name (size) because this level of hierarchy has a
different name scope than the lower level and there is no
conflict -- but there is no correspondence between the two
names either, so you still have to explicitly pass the
parameter value down through the hierarchy. */

parameter my_size = 16;    // I want a sixteen bit adder
output [my_size-1:0] sum;
input [my_size-1:0] a, b;
input cin;

/* my_size overrides size inside instance my_adder of adder */
// Creates my_adder as a sixteen bit adder
adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
Built-in Gate Primitives

You can create hardware by directly instantiating built-in gates into your design (in addition to instantiating your own modules.) The built-in Verilog gates are called primitives.

**Syntax**

\[
gate\_type\_keyword\ [\ instance\_name\ ] (\ port\_list\ ) ;
\]

The gate type keywords for simple and tristate gates are listed in the following tables. The `instance_name` is a unique instance name, and is optional. The signal names in the `port_list` can be given in any order with the restriction that all outputs must come before the inputs. For tristate gates, outputs come first, then inputs, and then enable.

The following tables list the available keywords.

**Table 8-2: Simple gates**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buf</code></td>
<td>buffer</td>
</tr>
<tr>
<td><code>not</code></td>
<td>inverter</td>
</tr>
<tr>
<td><code>and</code></td>
<td>and gate</td>
</tr>
<tr>
<td><code>nand</code></td>
<td>nand gate</td>
</tr>
<tr>
<td><code>or</code></td>
<td>or gate</td>
</tr>
<tr>
<td><code>nor</code></td>
<td>nor gate</td>
</tr>
<tr>
<td><code>xor</code></td>
<td>exclusive or gate</td>
</tr>
<tr>
<td><code>xnor</code></td>
<td>exclusive nor gate</td>
</tr>
</tbody>
</table>

**Table 8-3: Tristate gates**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bufif1</code></td>
<td>tristate buffer with logic one enable</td>
</tr>
</tbody>
</table>
Combinational Logic

Combinational logic is hardware in which the output values are based on some function of the current input values. There is no clock, and no saved states. Most hardware is a mixture of combinational and sequential logic.

You create combinational logic with an always block and/or continuous assignments.

Combinational Logic Examples

The following combinational logic synthesis examples are included in the synplicity_install_dir/examples/verilog/combinat directory:

- Adders
- ALU
- Bus Sorter
- 3-to-8 Decoder
- 8-to-3 Priority Encoders
- Comparator
- Multiplexors (concurrent signal assignments, case statements, or if-then-else statements can be used to create multiplexors; the Synplify synthesis tool automatically creates parallel multiplexers when the conditions in the branches are mutually exclusive)
- Parity Generator
- Tristate Drivers

Table 8-3: Tristate gates (Continued)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufif0</td>
<td>tristate buffer with logic zero enable</td>
</tr>
<tr>
<td>notif1</td>
<td>tristate inverter with logic one enable</td>
</tr>
<tr>
<td>notif0</td>
<td>tristate inverter with logic zero enable</td>
</tr>
</tbody>
</table>
always Blocks for Combinational Logic

Use the Verilog always blocks to model combinational logic as shown in the following template.

```verbatim
always @(event_expression)
begin
  // Procedural assignment statements,
  // if, case, casex, and casez statements
  // while, repeat, and for loops
  // task and function calls
end
```

When modeling combinational logic with always blocks, keep the following in mind:

- The always block must have exactly one event control (@(event_expression)) in it, located immediately after the always keyword.

- You should list all signals feeding into the combinational logic in the event expression. This includes all signals that affect signals that are assigned inside the always block. All signals on the right side of an assignment inside an always block should be listed. The Synplify synthesis tool assumes that the sensitivity list is complete, and generates the desired hardware. However, it will issue a warning message if any signals on the right side of an assignment inside an always block are not listed, because your pre- and post-synthesis simulation results might not match.

- You must explicitly declare as reg or integer all signals you assign in the always block.

**Note:** Make sure all signals assigned in a combinational always block are explicitly assigned values every time the always block executes. Otherwise, the synthesis tool must insert level-sensitive latches in your design to hold the last value for the paths that do not assign values. This will occur, for instance, if there are combinational loops in your design. This often represents a coding error. The synthesis tool issues a warning message that latches are being inserted into your design because of combinational loops. You will get an error.
message if you have combinational loops in your design that are not recognized as level-sensitive latches by the synthesis tool.

### Event Expression

Every `always` block must have one event control `@(event_expression)`, that specifies the signal transitions that trigger the `always` block to execute. This is analogous to specifying the inputs to logic on a schematic by drawing wires to gate inputs. If there is more than one signal, separate the names with the `or` keyword.

**Syntax**

```
always @(signal1 or signal2 ...)
```

**Example**

```verbatim
/* The first line of an always block for a mux that triggers when 'a', 'b' or 'sel' changes */
always @(a or b or sel)
```

Locate the event control immediately after the `always` keyword. Do not use the `posedge` or `negedge` keywords in the event expression; they imply edge-sensitive sequential logic.

**Example: Multiplexor**

See also *Example: Comma-separated Sensitivity List on page 8-5.*

```verbatim
module mux (out, a, b, sel);
output out;
input a, b, sel;
reg out;
always @(a or b or sel)
begin
  if (sel)
    out = a;
  else
    out = b;
end
endmodule
```
Continuous Assignments for Combinational Logic

Use continuous assignments to model combinational logic. To create a continuous assignment:

1. Declare the assigned signal as a wire using the syntax:

   ```verilog
   wire [msb:lsb] result_signal;
   ```

2. Specify your assignment with the `assign` keyword, and give the expression (value) to assign.

   ```verilog
   assign result_signal = expression;
   ```

   or ...

   Combine the wire declaration and assignment into one statement:

   ```verilog
   wire [msb:lsb] result_signal = expression;
   ```

Each time a signal on the right side of the equal sign (=) changes value, the expression re-evaluates, and the result is assigned to the signal on the left side of the equal sign. You can use any of the built-in operators to create the expression.

The bus range `[msb:lsb]` is only necessary if your signal is a bus (more than one bit wide).

All outputs and inouts to modules default to wires; therefore the wire declaration is redundant for outputs and inouts and “assign result_signal = expression;” is sufficient.

Example: Bit-wise AND

```verilog
module bitand (out, a, b);
output [3:0] out;
input [3:0] a, b;
// This wire declaration is not required because "out" is an output in the port list */
wire [3:0] out;
assign out = a & b;
endmodule
```
Example: 8-bit Adder

```verilog
module adder_8 (cout, sum, a, b, cin);
  output cout;
  output [7:0] sum;
  input cin;
  input [7:0] a, b;
  assign {cout, sum} = a + b + cin;
endmodule
```

Sequential Logic

Sequential logic is hardware that has an internal state or memory. The state elements are either flip-flops that update on the active edge of a clock signal or level-sensitive latches that update during the active level of a clock signal.

Because of the internal state, the output values might depend not only on the current input values, but also on input values at previous times. Sequential logic in which the updated state values depend on the previous state values is a state machine. There are standard ways of modeling state machines in Verilog. Most hardware is a mixture of combinational and sequential logic.

You create sequential logic with `always` blocks and/or continuous assignments.

Sequential Logic Examples

The following sequential logic synthesis examples are included in the `synplicity_install_dir/examples/verilog/sequential` directory:

- Flip-flops and level-sensitive latches
- Counters (up, down, and up/down)
- Register file
• Shift registers
• State machines

For additional information on synthesizing flip-flops and latches, see these topics:

• Flip-flops Using always Blocks on page 8-20
• Level-sensitive Latches on page 8-21
• Sets and Resets on page 8-24

Flip-flops Using always Blocks

The easiest way to create flip-flops/registers is to assign values to the signals in an always block, and specify the active clock edge in the event expression.

always Block Template

```
always @(event_expression)
begin
  // Procedural statements
end
```

The always block must have one event control (@(event_expression)) immediately after the always keyword that specifies the clock signal transitions that trigger the always block to execute.

Syntax

```
always @(edge_keyword clock_name)
```

where edge_keyword is posedge (for positive-edge triggered) or negedge (for negative-edge triggered).

Example

```
always @(posedge clk)
```
Assignments to Signals in always Blocks

- You must explicitly declare as reg or integer any signal you assign inside an always block.

- Any signal assigned within an edge-triggered always block will be implemented as a register; for instance, signal q in the following example.

Example

```verilog
module dff_or (q, a, b, clk);
  output q;
  input a, b, clk;
  reg q; // Declared as reg, since assigned in always block

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

In this example, the result of a | b connects to the data input of a flip-flop, and the q signal connects to the q output of the flip-flop.

Level-sensitive Latches

The preferred method of modeling level-sensitive latches in Verilog is to use continuous assignment statements.

Example

```verilog
module latchor1 (q, a, b, clk);
  output q;
  input a, b, clk;

  assign q = clk ? (a | b) : q;
endmodule
```

Whenever clk, a, or b change, the expression on the right side re-evaluates. If your clk becomes true (active, logic 1), a | b is assigned to the q output. When the clk changes and becomes false (deactivated), q is assigned to q (holds the last value of q). If a or b changes and clk is already active, the new value a | b is assigned to q.
Although it is simpler to specify level-sensitive latches using continuous assignment statements, you can create level-sensitive latches from \texttt{always} blocks. Use an \texttt{always} block and follow these guidelines for event expression and assignments.

**always Block Template**

\begin{verbatim}
always@ (event_expression)
begin  // Procedural statements
end
\end{verbatim}

Whenever the assignment to a signal is incompletely defined, the event expression specifies the clock signal and the signals that feed into the data input of the level-sensitive latch.

**Syntax**

\begin{verbatim}
always @ (clock_name or signal1 or signal2 ...)
\end{verbatim}

**Example**

\begin{verbatim}
always @(clk or data)
begin
  if (clk)
    q <= data;
end
\end{verbatim}

The \texttt{always} block must have exactly one event control (@(event_expression)) in it, and must be located immediately after the \texttt{always} keyword.

**Assignments to Signals in always Blocks**

- You must explicitly declare as \texttt{reg} or \texttt{integer} any signal you assign inside an \texttt{always} block.

- Any incompletely-defined signal that is assigned within a level-triggered \texttt{always} block will be implemented as a register.

**Note:** Whenever level-sensitive latches are generated from an \texttt{always} block, the Synplify synthesis tool issues a warning message, so that you can verify if a given level-sensitive latch is really
what you intended. (If you model a level-sensitive latch using continuous assignment then no warning message is issued.)

Example: Creating Level-sensitive Latches You Want

```verilog
module latchor2 (q, a, b, clk);
output q;
input a, b, clk;
reg q;
always @(clk or a or b)
begin
  if (clk)
    q <= a | b;
end
endmodule
```

If \( \text{clk} \), \( a \), or \( b \) change, and \( \text{clk} \) is a logic 1, then set \( q \) equal to \( a|b \).

Notice that we did not specify what to do when \( \text{clk} \) is a logic zero (there is no \textit{else} in the \textit{if} statement), so when \( \text{clk} \) is a logic 0, the last value assigned is maintained (there is an implicit \( q=q \)). The Synplify synthesis tool correctly recognizes this as a level-sensitive latch, and creates a level-sensitive latch in your design. The tool issues a warning message when you compile this module (after examination, you may choose to ignore this message).

Example: Creating Unwanted Level-sensitive Latches

```verilog
module mux4to1 (out, a, b, c, d, sel);
output out;
input a, b, c, d;
input [1:0] sel;
reg out;
always @(sel or a or b or c or d)
begin
  case (sel)
    2'd0: out = a;
    2'd1: out = b;
    2'd3: out = d;
  endcase
end
endmodule
```
In the above example, the `sel` case value `2'd2` was intentionally omitted. Accordingly, `out` is not updated when the select line has the value `2'd2`, and a level-sensitive latch must be added to hold the last value of `out` under this condition. The Synplify synthesis tool issues a warning message when you compile this module, and there can be mismatches between RTL simulation and post-synthesis simulation. You can avoid generating level-sensitive latches by adding the missing case in the `case` statement; using a “default” case in the `case` statement; or using the Verilog `full_case` directive.

## Sets and Resets

A set signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic one. Asynchronous sets take place independent of the clock, whereas synchronous sets only occur on an active clock edge.

A reset signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic zero. Asynchronous resets take place independent of the clock, whereas synchronous resets take place only at an active clock edge.

### Asynchronous Sets and Resets

Asynchronous sets and resets are independent of the clock. When active, they set flip-flop outputs to one or zero (respectively), without requiring an active clock edge. Therefore, list them in the event control of the `always` block, so that they trigger the `always` block to execute, and so that you can take the appropriate action when they become active.

**Event Control Syntax**

```
always @ ( edge_keyword clock_signal or edge_keyword reset_signal or
          edge_keyword set_signal)
```

*Edge_keyword* is *posedge* for active-high set or reset (or positive-edge triggered clock) or *negedge* for active-low set or reset (or negative-edge triggered clock)

You can list the signals in any order.
Example: Event Control

// Asynchronous, active-high set (rising-edge clock)
always @(posedge clk or posedge set)

// Asynchronous, active-low reset (rising-edge clock)
always @(posedge clk or negedge reset)

// Asynchronous, active-low set and active-high reset
// (rising-edge clock)
always @(posedge clk or negedge set or posedge reset)

Example: always Block Template with Asynchronous, Active-high reset, set
always @(posedge clk or posedge set or posedge reset)
begin
  if (reset) begin
    /* Set the outputs to zero */
    end
  else if (set) begin
    /* Set the outputs to one */
    end
  else begin
    /* Clocked logic */
    end
end
Example: flip-flop with Asynchronous, Active-high reset and set

```verilog
dff1 #1 (q, qb, d, clk, set, reset);
input d, clk, set, reset;
output q, qb;
// Declare q and qb to be reg because assigned inside always
reg q, qb;
always @(posedge clk or posedge set or posedge reset)
begin
  if (reset) begin
    q <= 0;
    qb <= 1;
  end else if (set) begin
    q <= 1;
    qb <= 0;
  end else begin
    q <= d;
    qb <= ~d;
  end
end
endmodule
```

For simple, single variable flip-flops, the following template can be used.

```verilog
always @(posedge clk or posedge set or posedge reset)
q = reset ? 1'b0 : set ? 1'b1 : d;
```

**Synchronous Sets and Resets**

Synchronous sets and resets set flip-flop outputs to logic 1 or 0 (respectively) on an active clock edge.

Do not list the set and reset signal names in the event expression of an always block because they should not trigger the always block to execute upon changing. Instead, trigger the always block on the active clock edge, and check the reset and set inside the always block first.

**RTL View Primitives**

The Verilog compiler can detect and extract the following flip-flops with synchronous sets and resets and display them in the RTL schematic view:

- `sdffr`—flip-flop with synchronous reset
- `sdffs`—flip-flop with synchronous set
• sdffrs—flip-flop with both synchronous set and reset
• sdffpat—vectored flip-flop with synchronous set/reset pattern
• sdffre—enabled flip-flop with synchronous reset
• sdffse—enabled flip-flop with synchronous set
• sdffpate—enabled, vectored flip-flop with synchronous set/reset pattern

**Note:** You can check the name (type) of any primitive by placing your mouse pointer over it in the RTL view; a tooltip displays the name.

![Flip-flops with synchronous sets and resets](image.png)

**Figure 8-1:** Flip-flops with synchronous sets and resets

**Event Control Syntax**

```verilog
code
always @ (edge_keyword clock_name )
```

In the syntax line, *edge_keyword* is posedge for a positive-edge triggered clock or negedge for a negative-edge triggered clock.
Example: Event Control

    // Positive edge triggered
    always @(posedge clk)

    // Negative edge triggered
    always @ (negedge clk)

Example: always Block Template with Synchronous, Active-high reset, set

    always @(posedge clk)
    begin
      if (reset) begin
        /* Set the outputs to zero */
      end else if (set) begin
        /* Set the outputs to one */
      end else begin
        /* Clocked logic */
      end
    end

Example: D Flip-flop with Synchronous, Active-high set, reset

    module dff2 (q, qb, d, clk, set, reset);
    input d, clk, set, reset;
    output q, qb;
    reg q, qb;

    always @(posedge clk)
    begin
      if (reset) begin
        q <= 0;
        qb <= 1;
      end else if (set) begin
        q <= 1;
        qb <= 0;
      end else begin
        q <= d;
        qb <= ~d;
      end
    end
endmodule
Verilog State Machines

A finite state machine is hardware that advances from state to state at a clock edge.

State Machine Guidelines

The Synplify synthesis tool works best with synchronous state machines. The recommended practice is to write a fully synchronous design, and to avoid asynchronous paths, such as a path through the asynchronous reset of a register. See Asynchronous State Machines on page 8-32 for information about asynchronous state machines.

- The state machine must have a synchronous or asynchronous reset to be inferred. State machines must have an asynchronous or synchronous reset to set the hardware to a valid state after power-up, and to reset your hardware during operation (asynchronous resets are available freely in most FPGA architectures).

- You can define state machines using multiple event controls in an always block only if the event control expressions are identical (for example, @(posedge clk)). These state machines are known as implicit state machines. However it is better to use the explicit style described here and shown in Example: FSM Coding Style on page 8-30.

- Separate the sequential from the combinational always block statements. Besides making it easier to read, it makes what is being registered very obvious. It also gives better control over the type of register element used.

- Represent states with defined labels or enumerated types.

- It is easiest to use a case statement in an always block to check the current state at the clock edge, advance to the next state, then set the output values. You could also use if statements in an always block, but generally you should stay with case statements for coding-style consistency.

- Always use a default assignment as the last assignment in your case statement and set the state variable to 'bx. See Example: default Assignment on page 8-31.
• Set encoding style with the `syn_encoding` directive. This attribute overrides the default encoding assigned during compilation. The default encoding is determined by the number of states. If you have a `syn_encoding` attribute, its value is used during the mapping stage to determine encoding style.

```verilog
attribute syn_encoding : string;
attribute syn_encoding of <typename> : type is "sequential";
```

See Chapter 7, *Synthesis Attributes and Directives*, for details about the syntax and values.

One-hot implementations are not always the best choice for state machines, even in FPGAs and CPLDs. For example, one-hot state machines might result in larger implementations, which can cause fitting problems. An example in an FPGA in which one-hot implementation can be detrimental is when the state machine drives a large decoder, generating many output signals. In a 16-state state machine, for instance, the output decoder logic might reference sixteen signals in a one-hot implementation, but only four signals in a sequential representation.

**Example: FSM Coding Style**

```verilog
module FSM1 (clk, rst, enable, data_in, data_out, state0, state1, state2);
input clk, rst, enable;
input [2:0] data_in;
output data_out, state0, state1, state2;

/* Defined state labels; this style preferred over `defines`*/
parameter deflt=2'bxx;
parameter idle=2'b00;
parameter read=2'b01;
parameter write=2'b10;

reg data_out, state0, state1, state2;
reg [1:0] state, next_state;

/* always block with sequential logic*/
always @(posedge clk or negedge rst)
  if (!rst) state <= idle;
  else state <= next_state;
```

/* always block with combinatorial logic*/
always @(state or enable or data_in) begin
/* Default values for FSM outputs*/
    state0 <= 1'b0;
    state1 <= 1'b0;
    state2 <= 1'b0;
    data_out <= 1'b0;
    data_in [0];
    data_in [1];
    data_in [2];
    state = 'bx;

    case (state)
        idle : if (enable) begin
            state0 <= 1'b1;
            data_out <= data_in[0];
            next_state <= read;
        end
        else begin
            next_state <= idle;
        end
        read : if (enable) begin
            state1 <= 1'b1;
            data_out <= data_in[1];
            next_state <= write;
        end
        else begin
            next_state <= read;
        end
        write : if (enable) begin
            state2 <= 1'b1;
            data_out <= data_in[2];
            next_state <= idle;
        end
        else begin
            next_state <= write;
        end
    /* Default assignment for simulation */
    default : next_state <= deflt;
    endcase
end
endmodule

Example: default Assignment

default: state = 'bx;

Assigning 'bx to the state variable (which is a “don't care” for synthesis) tells the Synplify synthesis tool that you have specified all the used states in your case statement. Any remaining states are not used, and the
synthesis tool can remove unnecessary decoding and gates associated with the unused states. You do not have to add any special, non-Verilog directives.

If you set the state to a used state for the default case (for example, default state = state1), the tool generates the same logic as if you assign 'bx, but there will be pre- and post-synthesis simulation mismatches until you reset the state machine. These mismatches occur because all inputs are unknown at start up on the simulator. You therefore go immediately into the default case, which sets the state variable to state1. When you power up the hardware, it can be in a used state, such as state2, and then advance to a state other than state1. Post-synthesis simulation behaves more like hardware with respect to initialization.

State Values

In Verilog, you must give explicit state values for states. This is usually done using parameters or `defines. The preferred method is to use parameters. The state names are shown in the FSM Viewer only if you use parameters.

Example 1: Using Parameters for State Values (Recommended)

```verilog
parameter state1 = 2'h1, state2 = 2'h2;
...
current_state = state2; // Setting current state to 2'h2
```

Example 2: Using `define for State Values

```verilog`
define state1 2'h1
`define state2 2'h2
...
current_state = `state2; // Setting current state to 2'h2
```

Asynchronous State Machines

Avoid defining asynchronous state machines in Verilog. An asynchronous state machine has states, but no clearly defined clock, and has combinational loops.
Do not use Synplify synthesis tools to design asynchronous state machines; the synthesis tool might remove your hazard-suppressing logic when it performs logic optimization, causing your asynchronous state machines to work incorrectly.

The synthesis tool displays a “Found combinational loop” warning message for an asynchronous state machine when it detects combinational loops in continuous assignment statements, always blocks, and built-in gate-primitive logic.

To create asynchronous state machines, do one of the following:

- To use Verilog, make a netlist of technology primitives from your target library. Any instantiated technology primitives are left in the netlist, and not removed during optimization.
- Use a schematic editor (and not Verilog) for the asynchronous state machine part of your design.

The following asynchronous state machine examples generate warning messages.

**Example 1: Asynchronous FSM with Continuous Assignment**

```verilog
module async1 (out, g, d);
  output out;
  input g, d;
  assign out = g & d | !g & out | d & out;
endmodule
```

**Example 2: Asynchronous FSM with an always Block**

```verilog
module async2 (out, g, d);
  output out;
  input g, d;
  reg out;

  always @(g or d or out)
  begin
    out = g & d | !g & out | d & out;
  end
endmodule
```
RAM Inference

The Synplify synthesis tool can infer synchronous and synchronously
resettable RAMs from your Verilog source code and, where appropriate,
generate technology-specific single or dual-port RAMs. No special input
such as attributes or directives in your source code is needed. The primitive
generated for an inferred RAM has an asynchronous READ, except as
noted in Synchronous READ RAMs on page 8-37.

The following table lists the supported technology-specific RAMs that the
Synplify synthesis tool can generate.

Table 8-4: Generated technology-specific RAMs

<table>
<thead>
<tr>
<th>Technology Vendor</th>
<th>Family</th>
<th>RAM Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera</td>
<td>FLEX10K/10KE</td>
<td>Single port</td>
</tr>
<tr>
<td>Altera</td>
<td>APEX20K/20KE/20KC</td>
<td>Single and dual port</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Virtex, Virtex-E, Virtex2, Virtex2p</td>
<td>Single and dual port. Dual port block RAMs have only one write port.</td>
</tr>
</tbody>
</table>

Example

```verilog
module ram_test(q, a, d, we, clk);
output [7:0] q;
input  [7:0] d;
input  [6:0] a;
input  clk, we;
reg [7:0] mem [127:0];
always @(posedge clk) begin
  if(we)
    mem[a] <= d;
end
assign q = mem[a];
endmodule
```
RAMs with Special Write Enables

The Synplify synthesis tool can infer RAMs when the Write Enable is tied to Vcc or when the Write Enable is nested within IF statements.

always-enabled Write Enable

The RAM extraction code supports the inference of RAMs with their Write Enable tied permanently to Vcc rather than implementing the logic in flip-flops.

Example

```verilog
module xyz (q, d, clk, addr);
  input clk;
  input [7:0] addr;
  input [3:0] d;
  output [3:0] q;
  reg [255:0] mem [3:0];
  wire we;
  assign we = 1'b1;
  always @(posedge clk)
  begin
    if(we)
      begin
        mem[addr] = d;
      end
    end
  assign q = mem[addr];
endmodule
```

Nested Write Enable

The RAM extraction code can infer RAMs when the Write Enable is more complex as found in nested if statements. The compilers extract common terms from the feedback MUX Enables to derive the common Write Enable signal.
Example

module xyz (q, d, we1, we2, clk, addr);
input clk, we1, we2;
input [7:0] addr;
input [3:0] d;
output [3:0] q;
reg [255:0] mem [3:0];

always @(posedge clk)
begin
  if(we1)
  begin
    if(we2)
    begin
      mem[addr] = d;
    end
  end
end

assign q = mem[addr];
endmodule

Limited RAM Resources

If your RAM resources are limited, you can designate additional instances of inferred RAMs as flip-flops and logic using the syn_ramstyle attribute. This attribute takes the string argument of registers – place the attribute on either the output signal driven by the RAM or on the instance name of the RAM. Use the Attributes panel of the SCOPE spreadsheet to assign this and all other attributes.

Additional Components Generated

After inferring a RAM for some technologies, you might notice that the Synplify synthesis tool has generated a few additional components adjacent to the RAM. This glue logic ensures accuracy in your post place-and-route simulation results.
Synchronous READ RAMs

All RAM primitives that the tool generates for inferred RAMs have asynchronous READs, except for single port Altera 10K RAMs and single port Xilinx Virtex block RAMs. For these exceptions, the Synplify synthesis tool generates a synchronous RAM. The following example shows how the address of the RAM must be registered to generate a synchronous READ RAM.

Example

```
module ram_test(q, a, d, we, clk);
output [7:0] q;
input  [7:0] d;
input  [6:0] a;
input  clk, we;

reg [6:0] read_add;
reg [7:0] mem [127:0];

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

assign q = mem[read_add];
endmodule
```
ROM Inference

ROM Tables

As part of B.E.S.T. (Behavioral Extraction Synthesis Technology), the Synplify synthesis tool infers ROMs (read-only memories) from your HDL source code, and generates block components in the RTL view for them.

The data contents of the ROMs are stored in a text file named `rom.info`. To quickly view `rom.info` in read-only mode, synthesize your HDL source code, open an RTL view, and then push down into the ROM component.

Generally, the Synplify synthesis tool infers ROMs from HDL source code that uses `case` statements, or equivalent `if` statements, to make 16 or more signal assignments using constant values (words). The constants must all be the same width.

Another requirement for ROM inference is that values must be specified for at least half of the address space. For example, if the ROM has 5 address bits, then the address space is 32 and at least 16 of the different addresses must be specified.

Example

```verilog
module rom(z,a);
    output [3:0] z;
    input [4:0] a;
    reg [3:0] z;
    always @(a) begin
        case (a)
            5'b00000 : z = 4'b0001;
            5'b00001 : z = 4'b0010;
            5'b00010 : z = 4'b0110;
            5'b00011 : z = 4'b1010;
            5'b00100 : z = 4'b1000;
            5'b00101 : z = 4'b1001;
            5'b00110 : z = 4'b0000;
            5'b00111 : z = 4'b1110;
            5'b01000 : z = 4'b1111;
            5'b01001 : z = 4'b1110;
            5'b01010 : z = 4'b0001;
            5'b01011 : z = 4'b1000;
        endcase
    end
```

5'b01100 : z = 4'b1110;
5'b01101 : z = 4'b0011;
5'b01110 : z = 4'b1111;
5'b01111 : z = 4'b1100;
5'b10000 : z = 4'b1000;
5'b10001 : z = 4'b0000;
5'b10010 : z = 4'b0011;
default : z = 4'b0111;
endcase
endmodule

ROM Table Data (rom.info File)

Note: This data is for viewing only

ROM work.rom4(behave)-z_1[3:0]
address width: 5
data width: 4
inputs:
0: a[0]
1: a[1]
2: a[2]
3: a[3]
4: a[4]
outputs:
0: z_1[0]
1: z_1[1]
2: z_1[2]
3: z_1[3]
data:
00000 -> 0001
00001 -> 0010
00010 -> 0110
00011 -> 1010
00100 -> 1000
00101 -> 1001
00110 -> 0000
00111 -> 1110
01000 -> 1111
01001 -> 1110
01010 -> 0001
01011 -> 1000
01100 -> 1110
01101 -> 0011
Instantiating Black Boxes in Verilog

Black boxes are modules in which just the interface is specified; internal information is ignored by the Synplify synthesis tool. Black boxes can be used to directly instantiate:

- Technology-vendor primitives and macros (including I/Os).
- User-designed macros whose functionality was defined in a schematic editor, or another input source. (When the place-and-route tool can merge design netlists from different sources.)

Black boxes are specified with the `syn_black_box` synthesis directive. If the macro is an I/O, use `black_box_pad_pin=1` on the external pad pin. See below for details, and Verilog Examples on page B-39.

Note: For most of the technology-vendor architectures, macro libraries are provided (in `installation_dir/lib/technology/family.v`) that predefine the black boxes for their primitives and macros (including I/Os).

Although the synthesis tool ignores the internals of black-box modules, you need to define their functionality if you plan to simulate with a Verilog simulator. The contents are simply ignored during synthesis.

If the black box has tristate outputs, you must define these outputs with a `black_box_tri_pins` directive (see `black_box_tri_pins` on page 7-142).
Instantiating a Black Box

To instantiate a black box:

1. Create an interface for the macro with no contents (you can have contents, but they are ignored during synthesis.)

   The interface only declares the ports and the port directions. You tell the synthesis tool to ignore the contents by putting `syn_black_box` just before the semicolon (;) in the module declaration. Here is an example:

   ```
   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
   ```

2. Make an instance of the stub in your design (just as you would any other module).

3. Compile the stub, along with the module containing the instantiation of the stub. After synthesis, the synthesis result file will contain a reference to `myram`.

Instantiating a Technology Vendor I/O

If you want to instantiate a pad from a vendor library and you are creating a black box for a pad, you must also specify which pin is the external pad pin with the `black_box_pad_pin=1` directive.

Example: Input Pad Black Box

   ```
   module vendors_input_pad(out, in) /* synthesis syn_black_box */;
   input in /* synthesis black_box_pad_pin=1 */;
   output out ;
   ...
   ```
Example: Output Pad Black Box

```verilog
module vendors_output_pad(out, in) /* synthesis syn_black_box */;
    input in;
    output out /* synthesis black_box_pad_pin=1 */;
    ...
```

Before placement and routing, you typically merge the synthesis result file and the netlist representing the black box. Technology vendors have different methods of merging netlists. Refer to the section for your vendor under FPGA and CPLD Support, for more information.

PREP Verilog Benchmarks

PREP (Programmable Electronics Performance) Corporation distributes benchmark results that show how FPGA vendors compare with each other in terms of device performance and area. The following PREP benchmarks are included in the `synplicity_install_dir/examples/verilog/prep` directory:

- PREP VHDL Benchmark 1. Data Path
- PREP VHDL Benchmark 2. Timer/Counter
- PREP VHDL Benchmark 3. Small State Machine
- PREP VHDL Benchmark 4. Large State Machine
- PREP VHDL Benchmark 5. Arithmetic Circuit
- PREP VHDL Benchmark 6. 16-Bit Accumulator
- PREP VHDL Benchmark 7. 16-Bit Counter
- PREP VHDL Benchmark 8. 16-Bit Pre-scaled Counter
- PREP VHDL Benchmark 9. Memory Map

The source code for the benchmarks can be used for design examples for synthesis or for doing your own FPGA vendor comparisons.

PREP Corp. has disbanded, but you can still obtain information from their Web site: www.prep.org.
Hierarchy: Structural Verilog

The Synplify synthesis tool accepts and processes hierarchical Verilog designs. You create hierarchy by instantiating a module or a built-in gate primitive within another module.

The signals connect across the hierarchical boundaries through the port list, and can either be listed by position (the same order that you declare them in the lower-level module), or by name (where you specify the name of the lower-level signals to connect to).

Connecting by name minimizes errors, and is recommended when the instantiated module has many ports.

Creating a Hierarchical Design

To create a hierarchical design:

1. Create modules.

2. Instantiate the modules within other modules. (When you instantiate modules inside of others, the ones that you have instantiated are sometimes called “lower-level modules” to distinguish them from the “top-level” module that is not inside of another module.)

3. Connect signals in the port list together across the hierarchy either “by position” or “by name” (see the examples, below).

Example: Creating Modules (Interfaces Shown)

```verilog
module mux(out, a, b, sel);
// mux
output [7:0] out;
input [7:0] a, b;
input sel;

// <mux functionality>
endmodule
```
module reg8(q, data, clk, rst); // Eight-bit register
output [7:0] q;
input [7:0] data;
input clk, rst;
// <Eight-bit register functionality>
endmodule

module rotate(q, data, clk, r_l, rst); // Rotates bits or loads
output [7:0] q;
input [7:0] data;
input clk, r_l, rst;
// When r_l is high, it rotates; if low, it loads data
// <Rotate functionality>
endmodule

Example: Top-level Module with Ports Connected by Position

module top1(q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;
// The order of the listed signals here will match
// the order of the signals in the mux module declaration.
mux mux_1 (mux_out, a, b, sel);
reg8 reg8_1 (reg_out, mux_out, clk, rst);
rotate rotate_1 (q, reg_out, clk, r_l, rst);
endmodule

Example: Top-level Module with Ports Connected by Name

module top2(q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;
/* The syntax to connect a signal "by name" is:
   .<lower_level_signal_name>(<local_signal_name>) */
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));

/* Ports connected "by name" can be in any order */
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
rotate rotate_1 (.q(q), .data(reg_out), .clk(clk), .r_l(r_l),
                 .rst(rst));

endmodule

**synthesis Macro**

Use this text macro along with the Verilog `ifdef compiler directive to conditionally exclude part of your Verilog code from being synthesized. The most common use of the synthesis macro is to avoid synthesizing stimulus that only has meaning for logic simulation.

The Synplify synthesis tool defines the synthesis macro so that the statement `ifdef synthesis is true. The tool compiles the statements in the `ifdef branch (usually there are none), and ignores the stimulus statements in the `else branch.

---

**Note:** Because Verilog simulators do not recognize a synthesis macro, the compiler for your simulator will use the stimulus in the `else branch.

---

The following example is implemented as an AND gate, because the Synplify synthesis tool uses the assign c = a & b branch:

```
`define DOAND
module top (a,b,c);
   input a,b;
   output c;
`ifdef synthesis
   assign c = a & b;
`else
   assign c = a | b;
`endif
endmodule
```
To implement the same example as an OR gate, add an `undef DOAND statement after the output c line, as follows:

```verilog
`define DOAND
module top (a,b,c);
    input a,b;
    output c;
`undef DOAND
`ifdef DOAND
    assign c = a & b;
`else
    assign c = a | b;
`endif
endmodule
```

**Note:** A macro in Verilog has a nonzero value only if it is defined.
Synthesis Attributes and Directives

Verilog synthesis attributes and directives allow you to associate information with your design to control the way it is analyzed, compiled, and mapped.

- **Attributes** direct the way your design is optimized and mapped during synthesis. Although you can place synthesis attributes directly in your source code, you should use the Attributes panel of the SCOPE spreadsheet to specify them in a constraint file. That way, changes can be made (via the constraint file), without requiring you to recompile.

- **Directives** control the way your design is analyzed prior to mapping. They must therefore be included directly in your source code; they cannot be specified in a constraint file.

Because Verilog does not have predefined attributes or directives for synthesis, you attach attributes and directives to source code items as comments.

In addition to the general attributes and directives described in Chapter 7, *Synthesis Attributes and Directives*, the Synplify synthesis tool also supports vendor-specific directives and attributes that apply to the specific programmable logic vendor you are targeting for your design.
CHAPTER 9

VHDL Language Support

This chapter discusses how you can use the VHDL language to create HDL source code for the Synplify synthesis tool:

- Language Constructs on page 9-2
- VHDL Language Constructs on page 9-5
- VHDL Synthesis Guidelines on page 9-33
- Sets and Resets on page 9-44
- State Machines on page 9-49
- Hierarchical Designs on page 9-56
- Configuration Specification and Declaration on page 9-60
- Scalable Designs on page 9-71
- RAM Inference on page 9-77
- Synthesis Attributes and Directives on page 9-85
- Instantiating Black Boxes in VHDL on page 9-86
- VHDL Synthesis Examples on page 9-89
- PREP VHDL Benchmarks on page 9-91
Language Constructs

The following sections define the supported, unsupported, and ignored language constructs for VHDL.

**Supported VHDL Language Constructs**

The following is a compact list of language constructs that are supported.

- Entity, architecture and package design units.
- Function and procedure subprograms.
- All IEEE library packages, including:
  - std_logic_1164
  - std_logic_unsigned
  - std_logic_signed
  - std_logic_arith
  - numeric_std and numeric_bit
  - standard library package (std)
- In, out, inout, buffer, linkage ports.
- Signals, constants, and variables.
- Aliases.
- Integer, physical and enumeration data types; subtypes of these.
- Arrays of scalars and records.
- Record data types.
- All operators (+, -, *, /, **, mod, rem, abs, not, =, /=, <, <=, >, >=, and, or, nand, nor, xor, xnor, sll, srl, sla, sra, rol, ror, &).
Note: Operators \(/\), \(\text{mod}\), and \(\text{rem}\) are supported for compile-time constants or when the right argument is a power of 2. Operator \(\text{**}\) is supported when the arguments are compile-time constants. The left operand must always be a power of 2 (or be zero). When the left operand is 2, the right operand can be a variable. Operators \text{sla}, \text{rol}, \text{and rot} are supported when the shift distance is a compile-time constant).

- Sequential statements: signal and variable assignment, wait, if, case, loop, for, while, return, null, function, and procedure call.
- Concurrent statements: signal assignment, process, block, generate (for and if), component instantiation, function, and procedure call.
- Component declarations and four methods of component instantiations.
- Configuration specification and declaration
- Generics; attributes; overloading.
- Next and exit looping control constructs.
- Predefined attributes: \(\text{t'base}\), \(\text{t'left}\), \(\text{t'right}\), \(\text{t'high}\), \(\text{t'low}\), \(\text{t'succ}\), \(\text{t'pred}\), \(\text{t'vel}\), \(\text{t'pos}\), \(\text{t'leftof}\), \(\text{t'rightof}\), \(\text{integer'image}\), \(\text{a'left}\), \(\text{a'right}\), \(\text{a'high}\), \(\text{a'low}\), \(\text{a'range}\), \(\text{a'reverse\_range}\), \(\text{a'length}\), \(\text{a'ascending}\), \(\text{s'stable}\), \(\text{s'event}\).

Miscellaneous VHDL Language Constructs

- Unconstrained ports in entities.
- Variable indexing of arrays; aggregates (and others clauses).
- Block ports and generics.
- Initializing signals and variables (except for initializing a variable declared in a process).
- Ranges in case statements.
- Nesting of functions.
Unsupported VHDL Language Constructs

If any of these constructs are found, an error message is reported and the synthesis run is cancelled.

- Access, and file types
- Register and bus kind signals
- Guarded blocks
- Expanded (hierarchical) names
- Global signals declared in packages
- User-defined resolution functions. The Synplify synthesis tool only supports the resolution functions for std_logic and std_logic_vector.

Ignored VHDL Language Constructs

The Synplify synthesis tool ignores the following constructs in your design. If found, the tool parses and ignores them and continues with the synthesis run.

- disconnect
- assert and report
VHDL Language Constructs

This section describes the synthesis language support that the Synplify synthesis tool provides for each VHDL construct. The language information is taken from the most recent VHDL Language Reference Manual (Revision ANSI/IEEE Std 1076-1993). The section names match those from the LRM for easy reference.

- Data Types
- Declaring and Assigning Objects in VHDL
- Signals and Ports
- Variables
- VHDL Constants
- Libraries and Packages
- Operators
- VHDL Process
- Common Sequential Statements
- Concurrent Signal Assignments
- Resource Sharing
- Combinational Logic
- Sequential Logic
- Component Instantiation in VHDL
- Creating a Scalable Design Using Generate Statements

Data Types

Predefined Enumeration Types

Enumeration types have a fixed set of unique values. The two predefined data types – bit and Boolean, as well as the std_logic type defined in the std_logic_1164 package are the types that represent hardware values. You
can declare signals and variables (and constants) that are vectors (arrays) of these types by using the types bit_vector, and std_logic_vector. You are recommended to use std_logic and std_logic_vector because they are highly flexible for synthesis and simulation.

Table 9-1: Type std_logic

<table>
<thead>
<tr>
<th>Values</th>
<th>Treated by the Synplify synthesis tool as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U' (uninitialized)</td>
<td>don’t care</td>
</tr>
<tr>
<td>'X' (forcing unknown)</td>
<td>don’t care</td>
</tr>
<tr>
<td>'0' (forcing logic 0)</td>
<td>logic 0</td>
</tr>
<tr>
<td>'1' (forcing logic 1)</td>
<td>logic 1</td>
</tr>
<tr>
<td>'Z' (high impedance)</td>
<td>high impedance</td>
</tr>
<tr>
<td>'W' (weak unknown)</td>
<td>don’t care</td>
</tr>
<tr>
<td>'L' (weak logic 0)</td>
<td>logic 0</td>
</tr>
<tr>
<td>'H' (weak logic 1)</td>
<td>logic 1</td>
</tr>
<tr>
<td>'-' (don’t care)</td>
<td>don’t care</td>
</tr>
</tbody>
</table>

Table 9-2: Type bit

<table>
<thead>
<tr>
<th>Values</th>
<th>Treated by the Synplify synthesis tool as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0'</td>
<td>logic 0</td>
</tr>
<tr>
<td>'1'</td>
<td>logic 1</td>
</tr>
</tbody>
</table>

Table 9-3: Type boolean

<table>
<thead>
<tr>
<th>Values</th>
<th>Treated by the Synplify synthesis tool as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>logic 0</td>
</tr>
<tr>
<td>true</td>
<td>logic 1</td>
</tr>
</tbody>
</table>
User-defined Enumeration Types

You can create your own enumerated types. This is common for state machines because it allows you to work with named values rather than individual bits or bit vectors.

Syntax

\[
\text{type type\_name is (value\_list);}
\]

Examples

\[
\begin{align*}
\text{type states is ( state0, state1, state2, state3);} \\
\text{type traffic\_light\_state is ( red, yellow, green);} \\
\end{align*}
\]

Integers

An integer is a predefined VHDL type that has 32 bits. When you declare an object as an integer, you should generally restrict the range of values to those you are using. This results in a minimum number of bits for implementation and on ports.

Data Types for Signed and Unsigned Arithmetic

For signed arithmetic, you have three choices:

- Use the std_logic_vector data type defined in the std_logic_1164 package, and the package std_logic_signed.
- Use the signed data type, and signed arithmetic defined in the package std_logic_arith.
- Use an integer subrange (for example: signal mysig: integer range -8 to 7). If the range includes negative numbers, the Synplify synthesis tool uses a twos-complement bit vector of minimum width to represent it (four bits in this example). Using integers limits you to a 32-bit range of values, and is only recommended to represent small buses. Integers are most commonly used for indexes.
- Use the signed data type from the numeric_std or numeric_bit packages.
For unsigned arithmetic, you have three choices:

- Use the std_logic_vector data type defined in the std_logic_1164 package and the package std_logic_unsigned.
- Use the unsigned data type and unsigned arithmetic defined in the package std_logic_arith.
- Use an integer subrange (for example: signal mysig: integer range 0 to 15;). If the integers are restricted to positive values, the Synplify synthesis tool uses an unsigned bit vector of minimum width to represent it (four bits in this example). Using integers limits you to a 32-bit range of values, and is only recommend to represent small buses (integers are most commonly used for indexes).
- Use the unsigned data type from the numeric_std or numeric_bit packages.

**Declaring and Assigning Objects in VHDL**

VHDL objects (object classes) include signals (and ports), variables, and constants. The Synplify synthesis tool does not support the file object class.

**Naming Objects**

VHDL is case insensitive. A VHDL name (identifier) must start with a letter and can be followed by any number of letters, numbers, or underscores (_). Underscores cannot be the first or last character in a name and cannot be used twice in a row. No special characters such as '$', '?', '“', '‘', or '!', can be used as part of a VHDL identifier.

**Syntax**

\[ \text{object_class object_name : data_type [ := initial_value ] ;} \]

Where:

- object_class is signal, variable, or constant.
- object_name is the name (the identifier) of the object.
data_type can be any predefined data type (such as bit or std_logic_vector) or user-defined data type.

Assignment Operators

<= Signal assignment operator.

:= Variable assignment and initial value operator.

Signals and Ports

In VHDL, the port list of the entity lists the I/O signals for the design. Ports of mode in can be read from, but not assigned (written) to. Ports of mode out can be assigned to, but not read from. Ports of mode inout are bidirectional and can be read from and assigned to. Ports of mode buffer are like inout ports but can have only one internal driver on them.

Internal signals are declared in the architecture declarative area and can be read from or assigned to anywhere within the architecture.

Examples

signal my_sig1 : std_logic; -- Holds a single std_logic bit
begin
    my_sig1 <= '1'; -- An architecture statement area
end

-- My_sig2 is a 4-bit integer
signal my_sig2 : integer range 0 to 15;
begin -- An architecture statement area
    my_sig2 <= 12;
end

-- My_sig_vec1 holds 8 bits of std_logic, indexed from 0 to 7
signal my_sig_vec1 : std_logic_vector (0 to 7);
begin -- An architecture statement area
    -- Simple signal assignment with a literal value
    my_sig_vec1 <= "01001000";
end

-- 16 bits of std_logic, indexed from 15 down to 0
signal my_sig_vec2 : std_logic_vector (15 downto 0);
begin -- An architecture statement area
    -- Simple signal assignment with a vector value
    my_sig_vec2 <= "011111001000101";
Variables

VHDL variables are declared within a process or subprogram and then used internally. Generally, variables are not visible outside the process or subprogram they are declared unless passed as a parameter to another subprogram.

Example

process (clk)  -- What follows is the process declaration area
  variable my_var1 : std_logic := '0';  -- Initial value '0'
begin  -- What follows is the process statement area
  my_var1 := '1';
end process;
Example

```vhdl
process (clk, reset)
-- Set the initial value of the variable to hex FF
    variable my_var2 : std_logic_vector (1 to 8) := X"FF";
begin
-- my_var2 is assigned the octal value 44
    my_var2 := O"44";
end process;
```

VHDL Constants

VHDL constants are declared in any declarative region and can be used within that region. The value of a constant cannot be changed.

Example

```vhdl
package my_constants is
    constant num_bits : integer := 8;
-- Other package declarations
end my_constants;
```

Libraries and Packages

When you want to synthesize a design in VHDL, include the HDL files in the source files list of your Synplify synthesis tool project. Often your VHDL design will have more than one source file. List all the source files in the order you want them compiled; the files at the top of the list are compiled first.

Compiling Design Units into Libraries

All design units in VHDL, including your entities and packages are compiled into libraries. A library is a special directory of entities, architectures and/or packages. You compile source files into libraries by adding them to the source file list. In VHDL, the library you are compiling has the default name `work`. All entities and packages in your source files are
automatically compiled into work. You can keep source files anywhere on your disk, even though you add them to libraries. You can have as many libraries as are needed.

1. To add a file to a library, select the file in the Project view.

   The library structure is maintained in the Project view. The name of the library to which a file belongs is displayed on the same line as the filename, and directly in front of it.

2. Choose Project -> Set Library from the menu bar, then type the library name. You can add any number of files to a library.

3. If you want to use a design unit that you compiled into a library (one that is no longer in the default work library), you must use a library clause in the VHDL source code to reference it.

   For example, if you add a source file for the entity ram16x8 to library my_rams, and instantiate the 16x8 RAM in the design called top_level, you need to add library my_rams; just before defining top_level.

### Predefined Packages

The Synplify synthesis tool supports the two standard libraries std and ieee that contain packages (most are now standardized) containing commonly used definitions of data types, functions, and procedures. These libraries and their packages are built in to the synthesis tool, so you never need to compile them. The packages are described in the following table.

Table 9-4: Predefined VHDL libraries and packages

<table>
<thead>
<tr>
<th>Library</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>std</td>
<td>standard</td>
<td>Defines the basic VHDL types including bit and bit_vector</td>
</tr>
<tr>
<td>ieee</td>
<td>std_logic_1164</td>
<td>Defines the 9-value std_logic and std_logic_vector types</td>
</tr>
<tr>
<td>ieee</td>
<td>numeric_std</td>
<td>Defines arithmetic operations on types defined in std_logic_1164</td>
</tr>
</tbody>
</table>
Accessing Packages

To gain access to a package include a `library` clause in your VHDL source code to specify the library the package is contained in, and a `use` clause to specify the name of the package. The `library` and `use` clauses must be included immediately before the design unit (entity or architecture) that uses the package definitions.

**Syntax**

```vhdl
library library_name;
use library_name.package_name.all;
```

To access the data types, functions and procedures declared in `std_logic_1164`, `std_logic_arith`, `std_logic_signed`, or `std_logic_unsigned`, you need a `library ieee` clause and a `use` clause for each of the packages you want to use.

**Example**

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

-- Other code
```
Library and Package Rules

To access the standard package, no library or use clause is required. The standard package is predefined (built-in) in VHDL, and contains the basic data types of bit, bit_vector, Boolean, integer, real, character, string, and others along with the operators and functions that work on them.

If you create your own package, and compile it into the work library to access its definitions, you still need a use clause before the entity using them, but not a library clause (because work is the default library.)

To access packages other than those in work and std, you need to provide a library and use clause for each package as shown in the following example of creating a resource library.

```vhdl
-- Compile this in library mylib
library ieee;
use ieee.std_logic_1164.all;

package my_constants is
constant max: std_logic_vector(3 downto 0):="1111";
  .
  .
end package;

-- Compile this in library work
library ieee, mylib;
use ieee.std_logic_1164.all;
use mylib.my_constants.all;

entity compare is
port(a: in std_logic_vector(3 downto 0);
  z: out std_logic);
end compare;

architecture rtl of compare is
begin
  z <= '1' when (a = max) else '0';
end rtl;
```

Note: The rising_edge and falling_edge functions are defined in the std_logic_1164 package. If you use these functions, your clock signal must to be declared as type std_logic.
Instantiating Components in a Design

No library or use clause is required to instantiate components (entities and their associated architectures) compiled in the default work library. The files containing the components must be listed in the source files list before the file(s) that instantiates them.

To instantiate components from the built-in technology-vendor macro libraries, you need to include the appropriate use and library clauses in your source code. Refer to the section for your vendor for more information.

To create a separate resource library to hold your components, put all the entities and architectures in one source file, and assign that source file the library components in the Synplify synthesis tool Project view. To access the components from your source code, put the clause `library components;` before the designs that instantiates them. There is no need for a use clause.

Operators

The Synplify synthesis tool supports creating expressions using all predefined VHDL operators:

Table 9-5: VHDL arithmetic operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division (supported for compile-time constants and when divisor is a power of 2)</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation (supported for compile-time constants and when left operand is 2; right operand can be a variable)</td>
</tr>
<tr>
<td>mod</td>
<td>modulus (supported for compile-time constants and when right operand is a power of 2)</td>
</tr>
<tr>
<td>rem</td>
<td>remainder (supported for compile-time constants and when right operand is a power of 2)</td>
</tr>
</tbody>
</table>
### Table 9-6: VHDL relational operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal (if either operand has a bit with an ‘X’ or ‘Z’ value, the result is ‘X’)</td>
</tr>
<tr>
<td>/=</td>
<td>not equal (if either operand has a bit with an ‘X’ or ‘Z’ value, the result is ‘X’)</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value ‘X’)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value ‘X’)</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value ‘X’)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value ‘X’)</td>
</tr>
</tbody>
</table>

### Table 9-7: VHDL logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>and</td>
</tr>
<tr>
<td>or</td>
<td>or</td>
</tr>
<tr>
<td>nand</td>
<td>nand</td>
</tr>
<tr>
<td>nor</td>
<td>nor</td>
</tr>
<tr>
<td>xor</td>
<td>xor</td>
</tr>
<tr>
<td>xnor</td>
<td>xnor</td>
</tr>
<tr>
<td>not</td>
<td>not (takes only one operand)</td>
</tr>
</tbody>
</table>
Table 9-8: VHDL shift operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sll</td>
<td>shift left logical – logically shifted left by R index positions</td>
</tr>
<tr>
<td>srl</td>
<td>shift right logical – logically shifted right by R index positions</td>
</tr>
<tr>
<td>sla</td>
<td>shift left arithmetic – arithmetically shifted left by R index positions</td>
</tr>
<tr>
<td>sra</td>
<td>shift right arithmetic – arithmetically shifted right by R index positions</td>
</tr>
<tr>
<td>rol</td>
<td>rotate left logical – rotated left by R index positions</td>
</tr>
<tr>
<td>ror</td>
<td>rotate right logical – rotated right by R index positions</td>
</tr>
</tbody>
</table>

Table 9-9: Other VHDL operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>identity</td>
</tr>
<tr>
<td>-</td>
<td>negation</td>
</tr>
<tr>
<td>&amp;</td>
<td>concatenation</td>
</tr>
</tbody>
</table>

Note: Initially, X’s are treated as don’t cares, but are eventually converted to 0’s or 1’s, whichever minimizes hardware.
VHDL Process

The VHDL keyword process introduces a block of logic that is triggered to execute when one or more signals change value. Use processes to model combinational and sequential logic.

process Template to Model Combinational Logic

```
<optional_label> : process (<sensitivity_list>)

-- Declare local variables, data types,
-- and other local declarations here

begin

-- Sequential statements go here, including:
-- signal and variable assignments
-- if and case statements
-- while and for loops
-- function and procedure calls

end process  <optional_label>;
```

Sensitivity List

The sensitivity list specifies the signal transitions that trigger the process to execute. This is analogous to specifying the inputs to logic on a schematic by drawing wires to gate inputs. If there is more than one signal, separate the names with commas.

A warning is issued when a signal is not in the sensitivity list but is used in the process, or when the signal is in the sensitivity list but not used by the process.

Syntax

```
process (signal1, signal2, ...) ;
```

A process can have only one sensitivity list, located immediately after the keyword process, or one or more wait statements. If there are one or more wait statements, one of these wait statements must be either the first or last statement in the process.

List all signals feeding into the combinational logic (all signals that affect signals assigned inside the process) in the sensitivity list. If you forget to list all signals, the Synplify synthesis tool generates the desired hardware,
and reports a warning message that you are not triggering the process every time the hardware is changing its outputs, and therefore your pre- and post-synthesis simulation results might not match.

Any signals assigned in the process must either be outputs specified in the port list of the entity or declared as signals in the architecture declarative area.

Any variables assigned in the process are local and must be declared in the process declarative area.

**Note:** Make sure all signals assigned in a combinational process are explicitly assigned values every time the process executes,. Otherwise, the synthesis tool must insert level-sensitive latches in your design, in order to hold the last value for the paths that don’t assign values (if, for example, you have combinational loops in your design). This usually represents coding error, so the synthesis tool issues a warning message that level-sensitive latches are being inserted into the design because of combinational loops. You will get an error message if you have combinational loops in your design that are not recognized as level-sensitive latches.

### Common Sequential Statements

**if-then-else Statement**

**Syntax**

```
if condition1 then
  sequential_statement(s)
[elsif condition2 then
  sequential_statement(s)]
[else
  sequential_statement(s)]
end if ;
```

**Note:** The else and elsif clauses are optional.
Example

library ieee;
use ieee.std_logic_1164.all;
entity mux is
    port (output_signal : out std_logic;
        a, b, sel : in std_logic);
end mux;

architecture if_mux of mux is
begin
    process (sel, a, b)
    begin
        if sel = '1' then
            output_signal <= a;
        elsif sel = '0' then
            output_signal <= b;
        else
            output_signal <= 'X';
        end if;
    end process;
end if_mux;

case Statement

Syntax

    case expression is
        when choice1 => sequential_statement(s)
        when choice2 => sequential_statement(s)

        -- Other case choices
        when choiceN => sequential_statement(s)
    end case;

Note: VHDL requires that the expression match one of the given choices. To create a default, have the final choice be when others => sequential_statement(s) or null. (Null means not to do anything.)
Example

library ieee;
use ieee.std_logic_1164.all;
entity mux is
    port (output_signal : out std_logic;
          a, b, sel : in std_logic);
end mux;

architecture case_mux of mux is
begin
    process (sel, a, b)
    begin
        case sel is
            when '1' =>
                output_signal <= a;
            when '0' =>
                output_signal <= b;
            when others =>
                output_signal <= 'X';
        end case;
    end process;
end case_mux;

Concurrent Signal Assignments

There are three types of concurrent signal assignments in VHDL.

- Simple
- Selected (with-select-when)
- Conditional (when-else)

Use the concurrent signal assignment to model combinational logic. Put the concurrent signal assignment in the architecture body. You can have as many statements as you need to describe the implementation of your hardware. Because all statements are concurrently active, the order you place them in the architecture body is not important.
Re-evaluation of Signal Assignments

Every time any signal on the right side of the assignment operator (\(<=\)) changes value (including signals used in the expressions, values, choices, or conditions), the assignment statement is re-evaluated, and the result is assigned to the signal on the left side of the assignment operator. You can use any of the predefined operators to create the assigned value.

Simple Signal Assignments

Syntax

```
signal <= expression;
```

Example

```
architecture simple_example of simple is
begin
  c <= a nand b;
end simple_example;
```

Selected Signal Assignments

Syntax

```
with expression select
  signal <= value1 when choice1,
           value2 when choice2,
           .
           .
           .
           valueN when choiceN;
```

Example

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (output_signal : out std_logic;
       a, b, sel : in std_logic);
end mux;
```
architecture with_select_when of mux is
begin
  with sel select
    output_signal <= a when '1',
                 b when '0',
                 'X' when others;
end with_select_when;

Conditional Signal Assignments

Syntax

\[
\text{signal} <= \text{value1 when condition1 else value2 when condition2 else valueN-1 when conditionN-1 else valueN;}
\]

Example

library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (output_signal: out std_logic;
     a, b, sel: in std_logic);
end mux;

architecture when_else_mux of mux is
begin
  output_signal <= a when sel = '1' else
                   b when sel = '0' else
                   'X';
end when_else_mux;

Resource Sharing

When you have mutually exclusive operators in a case statement, the Synplify synthesis tool shares resources for the operators in the case statements. For example, automatic sharing of operator resources includes adders, subtractors, incrementors, decrementors, and multipliers.
Combinational Logic

Combinational logic is hardware in which the output values are based on some function of the current input values. There is no clock and no saved states. Most hardware is a mixture of combinational and sequential logic.

Create combinational logic with concurrent signal assignments and/or processes.

Sequential Logic

Sequential logic is hardware that has an internal state or memory. The state elements are either flip-flops that update on the active edge of a clock signal, or level-sensitive latches, that update during the active level of a clock signal.

Because of the internal state, the output values might depend not only on the current input values, but also on input values at previous times. State machines are made of sequential logic in which the updated state values depend on the previous state values. There are standard ways of modeling state machines in VHDL. Most hardware is a mixture of combinational and sequential logic.

Create sequential logic with processes and/or concurrent signal assignments.

Component Instantiation in VHDL

A structural description of a design is made up of component instantiations that describe the subsystems of the design and their signal interconnects. The Synplify synthesis tool supports four major methods of component instantiation:

- Simple component instantiation (described below)
- Selected component instantiation
- Direct entity instantiation
- Configurations described in Configuration Specification on page 9-60
Simple Component Instantiation

In this method, a component is first declared either in the declaration region of the architecture, or in a package of (typically) component declarations, and then instantiated in the architecture’s statement region. By default, the synthesis process binds a named entity (and architecture) in the working library to all component instances that specify a component declaration with the same name.

Syntax

\[
\text{label: } \begin{cases}
\text{component} & \text{declaration_name} \\
\text{generic map} & (\text{actual\_generic1}, \text{actual\_generic2}, \ldots) \\
\text{port map} & (\text{port1}, \text{port2}, \ldots)
\end{cases};
\]

The use of the reserved word component is optional in component instantiations.

Example: VHDL 1987

```
architecture struct of hier_add is
  component add
    generic (size : natural);
    port (a : in bit_vector(3 downto 0);
         b : in bit_vector(3 downto 0);
         result : out bit_vector(3 downto 0)
    );
  end component;

begin
  -- Simple component instantiation
  add1 : add
    generic map(size => 4)
    port map(a => ain,
              b => bin,
              result => q);
  -- Other code
```
Example: VHDL 1993

```vhdl
architecture struct of hier_add is
component add
    generic (size : natural);
    port (a : in bit_vector(3 downto 0);
         b : in bit_vector(3 downto 0);
         result : out bit_vector(3 downto 0))
end component;
begin
    -- Simple component instantiation
    add1: component add -- Component keyword new in 1993
        generic map(size => 4)
        port map(a => ain,
                  b => bin,
                  result => q);
    -- Other code
```

**Note:** If no entity is found in the working library named the same as the declared component, all instances of the declared component are mapped to a black box and the error message “Unbound component mapped to black box” is issued.

---

**VHDL Selected Name Support**

Selected Name Support (SNS) is provided in VHDL for constants, operators, and functions in library packages. SNS eliminates ambiguity in a design referencing elements with the same names, but that have unique functionality when the design uses the elements with the same name defined in multiple packages. By specifying the library, package, and specific element (constant, operator, or function), SNS designates the specific constant, operator, or function used. This section discusses all facets of SNS. Complete VHDL examples are included to assist you in understanding how to use SNS effectively.
Constants

SNS lets you designate which constant to use from multiple library packages. To incorporate a constant into a design, specify the library, package, and constant. Using this feature eliminates ambiguity when multiple library packages have identical names for constants and are used in an entity-architecture pair.

The following example has two library packages available to the design CONSTANTS. Each library package has a constant defined by the name C1 and each has a different value. To distinguish between the constants and determine which one is used, SNS is used to specify the constant (see WORK.PACKAGE.C1 in the constants example below).

```vhdl
-- CONSTANTS PACKAGE1
library IEEE;
use IEEE.std_logic_1164.all;
package PACKAGE1 is
  constant C1: std_logic_vector := "10010101";
end PACKAGE1;

-- CONSTANTS PACKAGE2
library IEEE;
use IEEE.std_logic_1164.all;
package PACKAGE2 is
  constant C1: std_logic_vector := "10110110";
end PACKAGE2;

-- CONSTANTS EXAMPLE
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity CONSTANTS is
  generic (num_bits : integer := 8) ;
  port (a,b: in std_logic_vector (num_bits -1 downto 0);
        out1, out2: out std_logic_vector (num_bits -1 downto 0) )
end CONSTANTS;
architecture RTL of CONSTANTS is
begin
  out1 <= a + work PACKAGE1.C1; --Example of specifying SNS
  out2 <= b + work PACKAGE2.C1; --Example of specifying SNS
end RTL;
```
In the above design, outputs out1 and out2 use two C1 constants from two different packages. Although each output uses a constant named C1, the constants are not equivalent. For out1, the constant C1 is from PACKAGE1. For out2, the constant C1 is from PACKAGE2. For example:

\[
\text{out1} \leftarrow a + \text{work.PACKAGE1.C1}; \text{ is equivalent to } \text{out1} \leftarrow a + "10001010";
\]

whereas:

\[
\text{out2} \leftarrow b + \text{work.PACKAGE2.C1}; \text{ is equivalent to } \text{out2} \leftarrow b + "10110110";
\]

The constants have different values in different packages. SNS specifies the package and eliminates ambiguity within the design.

### Functions and Operators

Functions and operators in VHDL library packages customarily have overlapping naming conventions. For example, the add operator in the IEEE standard library exists in both the std_logic_signed and std_logic_unsigned packages. Depending upon which add operator is used, different values result. Defining only one of the IEEE library packages is a straightforward solution to eliminate ambiguity, but applying this solution is not always possible. A design requiring both std_logic_signed and std_logic_unsigned package elements must use SNS to eliminate ambiguity.

### Functions

In the following example, multiple IEEE packages are declared in a 256x8 RAM design. Both std_logic_signed and std_logic_unsigned packages are included. In the RAM definition, the signal address_in is converted from type std_logic_vector to type integer using the CONV_INTEGER function, but which CONV_INTEGER function will be called? SNS determines which function the design uses. The RAM definition clearly declares the std_logic_unsigned package as the source for the CONV_INTEGER function.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;
```
entity FUNCTIONS is
  port( address : in std_logic_vector(7 downto 0);
       data_in : in std_logic_vector(7 downto 0);
       data_out : out std_logic_vector(7 downto 0);
       we : in std_logic;
       clk : in std_logic
  );
end FUNCTIONS;

architecture RTL of FUNCTIONS is

  type mem_type is array (255 downto 0) of std_logic_vector (7 downto 0);
  signal mem: mem_type;
  signal address_in: std_logic_vector(7 downto 0);

begin
  data_out <= mem(IEEE.std_logic_unsigned.CONV_INTEGER(address_in));
  process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        mem(IEEE.std_logic_unsigned.CONV_INTEGER(address_in)) <= data_in;
      end if;
      address_in <= address;
    end if;
  end process;
end RTL;

Operators

In this example, comparator functions from the IEEE std_logic_signed and
std_logic_unsigned library packages are used. Depending upon which
comparator is called, a signed or an unsigned comparison results. In the
assigned outputs below, the op1 and op2 functions show the valid SNS
syntax for operator implementation.

  library IEEE;
  use IEEE.std_logic_1164.std_logic_vector;
  use IEEE.std_logic_signed.">";
  use IEEE.std_logic_unsigned.">";

entity OPERATORS is
  port (
    in1 : std_logic_vector(1 to 4);
    in2 : std_logic_vector(1 to 4);
    in3 : std_logic_vector(1 to 4);
    in4 : std_logic_vector(1 to 4);
    op1, op2 : out boolean
  );
end OPERATORS;

architecture RTL of OPERATORS is
begin
  process(in1, in2, in3, in4)
  begin

    -- Example of specifying SNS
    op1 <= IEEE.std_logic_signed."="(in1, in2);

    -- Example of specifying SNS
    op2 <= IEEE.std_logic_unsigned."="(in3, in4);
  end process;
end RTL;

User-defined Function Support

SNS is not limited to predefined standard IEEE packages and packages supported by the Synplify synthesis tool; SNS also supports user-defined packages. You can create library packages that access constants, operators, and functions in the same manner as the packages supported by IEEE or the synthesis tool.

The following example incorporates two user-defined packages in the design. Each package includes a function named func. In PACKAGE1, func is an XOR gate, whereas in PACKAGE2, func is an AND gate. Depending on which package is called, func results in either an XOR or an AND gate. The function call uses SNS to distinguish which function is called.

-- USER DEFINED PACKAGE1
library IEEE;
use IEEE.std_logic_1164.all;
package PACKAGE1 is
  function func(a, b: in std_logic) return std_logic;
end PACKAGE1;

Can you explain how SNS works with user-defined packages and how it helps in distinguishing between different functions?
package body PACKAGE1 is
  function func(a,b: in std_logic) return std_logic is
    begin
    return(a xor b);
    end func;
  end PACKAGE1;

-- USER DEFINED PACKAGE2
library IEEE;
use IEEE.std_logic_1164.all;

package PACKAGE2 is
  function func(a,b: in std_logic) return std_logic;
end PACKAGE2;

package body PACKAGE2 is
  function func(a,b: in std_logic) return std_logic is
    begin
    return(a and b);
    end func;
  end PACKAGE2;

-- USER DEFINED FUNCTION EXAMPLE
library IEEE;
use IEEE.std_logic_1164.all;

entity USER_DEFINED_FUNCTION is
  port (
    in0: in  std_logic;
    in1: in  std_logic;
    out0: out std_logic;
    out1: out std_logic
  );
end USER_DEFINED_FUNCTION;

architecture RTL of USER_DEFINED_FUNCTION is
begin
  out0 <= work.PACKAGE1.func(in0, in1);
  out1 <= work.PACKAGE2.func(in0, in1);
end RTL;
Demand Loading

In the previous section, the user-defined function example successfully uses SNS to determine which func function to implement. However, neither PACKAGE1 nor PACKAGE2 was declared as a use package clause (for example, work.PACKAGE1.all). How could func have been executed without a use package declaration? A feature of SNS is demand loading, which loads the necessary package without explicit use declarations. Demand loading lets you create designs using SNS without use package declarations, which supports all necessary constants, operators, and functions.
VHDL Synthesis Guidelines

General Synthesis Guidelines

Some general guidelines are presented here to help you synthesize your VHDL design.

- Top-level entity and architecture. The Synplify synthesis tool chooses the top-level entity and architecture – the last architecture for the last entity in the last file compiled. Entity selection can be overridden from the VHDL panel of the Options for Implementation dialog box. Files are compiled in the order in which they appear – from top to bottom in the Project view source files list.

- Simulate your design before synthesis because it exposes logic errors. Logic errors that are not caught are passed through the synthesis tool, and the synthesized results will contain the same logic errors.

- Simulate your design after placement and routing. Have the place-and-route tool generate a post placement and routing (timing-accurate) simulation netlist, and do a final simulation before programming your devices.

- Avoid asynchronous state machines. To use the synthesis tool for asynchronous state machines, make a netlist of technology primitives from your target library.

- For modeling level-sensitive latches, it is simplest to use concurrent signal assignments.

VHDL Language Guidelines

This section discusses VHDL language guidelines.

Processes

- A process must have either a sensitivity list or one wait statement.

- Each sequential process can be triggered from exactly one clock and only one edge of clock (and optional sets and resets).
• Avoid combinational loops in processes. Make sure all signals assigned in a combinational process are explicitly assigned values every time the process executes; otherwise, the Synplify synthesis tool needs to insert level-sensitive latches in your design to hold the last value for the paths that don’t assign values. This might represent a mistake on your part, so the Synplify synthesis tool issues a warning message that level-sensitive latches are being inserted into your design. You will get an warning message if you have combinational loops in your design that are not recognized as level-sensitive latches (for example, if you have an asynchronous state machine).

Assignments

• Assigning an 'X' or '-' to a signal is interpreted as a “don’t care”, so the Synplify synthesis tool creates the hardware that is the most efficient design.

Data Types

• Integers are 32-bit quantities. If you declare a port to be an “integer” data type, you should specify a range (for example, my_input: in integer range 0 to 7). Otherwise, your synthesis result file will contain a 32-bit port.

• Enumeration types are represented as a vector of bits. The encoding can be sequential, gray, or one hot. You might need to manually choose the encoding for ports with an enumeration type.

Model Template

You can place any number of concurrent statements (signal assignments, processes, component instantiations, and generate statements) in your architecture body as shown in the following example. The order of these statements within the architecture is not important as all can execute concurrently.

• The statements between the begin and the end in a process execute sequentially, in the order you type them from top to bottom.

• You can add comments in VHDL by proceeding your comment text with two dashes “--”. Any text from the dashes to the end of the line is treated as a comment, and ignored by the Synplify synthesis tool.
-- List libraries/packages that contain definitions you use
library <library_name> ;
use <library_name>.<package_name>.all ;

-- The entity describes the interface for your design.
entity <entity_name> is
  generic ( <define_interface_constants_here> ) ;
  port ( <port_list_information_goes_here> ) ;
end <entity_name> ;

-- The architecture describes the functionality (implementation)
-- of your design
architecture <architecture_name> of <entity_name> is

-- Architecture declaration region.
-- Declare internal signals, data types, and subprograms here

-- If you will create hierarchy by instantiating a
-- component (which is just another architecture), then
-- declare its interface here with a component declaration;
component <entity_name_instantiated_below>
  port ( <port_list_information_as_defined_in_the_entity> ) ;
end component ;

begin  -- Architecture body, describes functionality

-- Use concurrent statements here to describe the functionality
-- of your design. The most common concurrent statements are the
-- concurrent signal assignment, process, and component
-- instantiation.

-- Concurrent signal assignment (simple form):
<result_signal_name> <= <expression> ;

-- Process:
process <sensitivity list>)
-- Declare local variables, data types,
-- and other local declarations here
begin

-- Sequential statements go here, including:
  -- signal and variable assignments
  -- if and case statements
  -- while and for loops
  -- function and procedure calls
end process;


-- Component instantiation
<instance_name> : <entity_name>
    generic map (<override values here >)
    port map (<port list>);
end <architecture_name> ;

Creating Flip-flops and Registers Using VHDL Processes

It is easy to create flip-flops and registers using a process in your VHDL design.

process Template

    process (<sensitivity list>)
    begin
        <sequential statement(s)>
    end;

To create a flip-flop:

1. List your clock signal in the sensitivity list. Recall that if the value of any signal listed in the sensitivity list changes, the process is triggered, and executes. For example,

    process (clk)

2. Check for rising_edge or falling_edge as the first statement inside the process. For example,

    process (clk)
    begin
    if rising_edge(clk) then
        <sequential statement(s)>
    or

    process (clk)
    begin
    if falling_edge(clk) then
        <sequential statement(s)>
**Note:** Alternatively you could use an if clk'event and clk = '1' then statement to test for a rising edge (or if clk'event and clk = '0' then for a falling edge). Although these statements work, for clarity and consistency, use the rising_edge and falling_edge functions from the VHDL 1993 standard.

3. Set your flip-flop output to a value, with no delay, if the clock edge occurred. For example, q <= d;

**Complete Example**

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

entity dff_or is
    port (a, b, clk: in std_logic;
         q: out std_logic);
end dff_or;

architecture sensitivity_list of dff_or is
begin
    process (clk) -- Clock name is in sensitivity list
    begin
        if rising_edge(clk) then
            q <= a or b;
        end if;
    end process;
end sensitivity_list;
```

In this example, if clk has an event on it, the process is triggered and starts executing. The first statement (the if statement) then checks to see if a rising edge has occurred for clk. If the if statement evaluates to true, there was a rising edge on clk and the q output is set to the value of a or b. If the clk changes from 1 to 0, the process is triggered and the if statement executes, but it evaluates to false and the q output is not updated. This is the functionality of a flip-flop, and synthesis correctly recognizes it as such and connects the result of the a or b expression to the data input of a D-type flip-flop and the q signal to the q output of the flip-flop.

**Note:** The signals you set inside the process will drive the data inputs of D-type flip-flops.
Clock Edges

There are many ways to correctly represent clock edges within a process including those shown here.

The recommended rising clock edge representation is:

```
rising_edge(clk) -- Easiest
```

Other supported rising clock edge representations are:

```
clk = '1' and clk'event
clk'last_value = '0' and clk'event
clk'event and clk /= '0'
```

The recommended falling clock edge representation is:

```
falling_edge(clk) -- Easiest
```

Other supported falling clock edge representations are:

```
clk = '0' and clk'event
clk'last_value = '1' and clk'event
clk'event and clk /= '1'
```

Incorrect or Unsupported Representations for Clock Edges

Rising clock edge:

```
clk = '1'
clk and clk'event -- Because clk is not a Boolean
```

Falling clock edge:

```
clk = '0'
not clk and clk'event -- Because clk is not a Boolean
```
Defining an Event Outside a Process

The 'event attribute can be used outside of a process block. For example, the process block

```vhdl
process (clk,d)
begin
  if (clk='1' and clk'event) then
    q <= d;
  end if;
end process;
```

can be replaced by including the following line outside of the process statement:

```vhdl
q <= d when (clk='1' and clk'event);
```

Using a WAIT Statement Inside a Process

The Synplify synthesis tool supports a wait statement inside a process to create flip-flops, instead of using a sensitivity list.

Example

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity dff_or is
  port (a, b, clk: in std_logic;
        q: out std_logic);
end dff_or;
architecture wait_statement of dff_or is
begin
  process
  -- Note the absence of a sensitivity list.
  begin
    -- The process waits here until the condition becomes true
    wait until rising_edge(clk);
    q <= a or b;
  end process;
end wait_statement;
```
Rules for Using wait Statements Inside a Process

- It is illegal in VHDL to have a process with a wait statement and a sensitivity list.
- The wait statement must either be the first or the last statement of the process.

Clock Edge Representation in wait Statements

The recommended rising clock edge representation is:

```vhdl
wait until rising_edge(clk);
```

Other supported rising clock edge representations are:

```vhdl
wait until clk = '1' and clk'event
wait until clk'last_value = '0' and clk'event
wait until clk'event and clk /= '0'
```

The recommended falling clock edge representation is:

```vhdl
wait until falling_edge(clk)
```

Other supported falling clock edge representations are:

```vhdl
wait until clk = '0' and clk'event
wait until clk'last_value = '1' and clk'event
wait until clk'event and clk /= '1'
```

Level-sensitive Latches Using Concurrent Signal Assignments

To model level-sensitive latches in VHDL, we recommend using a concurrent signal assignment statement. You would use the conditional signal assignment form (also known as `when-else`).

Syntax

```vhdl
signal <= value1 when condition1 else
        value2 when condition2 else
        valueN-1 when conditionN-1 else
        valueN;
```
Example

In VHDL, you are not allowed to read the value of ports of mode out inside of an architecture that it was declared for. Ports of mode buffer can be read from and written to, but must have no more than one driver for the port in the architecture. In the following port statement example, q is defined to be of mode buffer.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity latchor1 is
  port (a, b, clk : in std_logic;
    -- q has mode buffer so it can be read inside architecture
        q: buffer std_logic);
end latchor1;

architecture behave of latchor1 is
begin
  q <= a or b when clk = '1' else q;
end behave;
```

Whenever clk, a, or b changes, the expression on the right side re-evaluates. If clk becomes true (active, logic 1), the value of a or b is assigned to the q output. When the clk changes and becomes false (deactivated), q is assigned to q (holds the last value of q). If a or b changes, and clk is already active, the new value of a or b is assigned to q.

Level-sensitive Latches Using VHDL Processes

Although it is simpler to specify level-sensitive latches using concurrent signal assignment statements, you can create level-sensitive latches with VHDL processes. Follow the guidelines given here for the sensitivity list and assignments.

process Template

```vhdl
process (<sensitivity list>)
begin
  <sequential statement(s)>
end process;
```
Sensitivity List

The sensitivity list specifies the clock signal, and the signals that feed into the data input of the level-sensitive latch. The sensitivity list must be located immediately after the process keyword.

Syntax

\[
\text{process} (\text{clock\_name, signal1, signal2, ...})
\]

Example

process (clk, data)

process Template for a Level-sensitive Latch

process (<clock, data\_signals ... > ...) 
begin
  if (<clock> = <active\_value>)
    <signals> <= <expression\_involving\_data\_signals> ;
  end if;
end process ;

All data signals assigned in this manner become logic into data inputs of level-sensitive latches.

Note: Whenever level-sensitive latches are generated from a process, the Synplify synthesis tool issues a warning message so that you can verify if level-sensitive latches are really what you intended. Often a thorough simulation of your architecture will reveal mistakes in coding style that would cause level-sensitive latches to be created during synthesis.

Example: Creating Level-sensitive Latches that You Want

library ieee;
use ieee.std_logic_1164.all;
entity latchor2 is
  port (a, b, clk : in std_logic ;
        q: out std_logic );
end latchor2;
architecture behave of latchor2 is
begin
process (clk, a, b)
begin
  if clk = '1' then
    q <= a or b;
  end if;
end process;
end behave;

If there is an event (change in value) on either clk, a or b, and clk is a logic 1, set q to a or b.

Notice that we did not specify what to do when clk is a logic 0 (there is no else) so when clk is a logic zero, the last value assigned is maintained (there is an implicit q=q). The Synplify synthesis tool correctly recognizes this as a level-sensitive latch, and creates a level-sensitive latch in your design. It will issue a warning message when you compile this architecture, but after examination, this warning message can safely be ignored.

Example: Creating Unwanted Level-sensitive Latches
This design demonstrates the level-sensitive latch warning caused by a missed assignment in the when two => case. The message generated is:

"Latch generated from process for signal odd, probably caused by a missing assignment in an if or case statement".

This information will help you find a functional error even before simulation.

library ieee;
use ieee.std_logic_1164.all;
entity mistake is
  port (inp: in std_logic_vector (1 downto 0);
       outp: out std_logic_vector (3 downto 0);
       even, odd: out std_logic);
end mistake;

architecture behave of mistake is
constant zero: std_logic_vector (1 downto 0):= "00";
constant one: std_logic_vector (1 downto 0):= "01";
constant two: std_logic_vector (1 downto 0):= "10";
constant three: std_logic_vector (1 downto 0):= "11";
begin
process (inp)
Sets and Resets

A set signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic one.

A reset signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic zero.

Asynchronous Sets and Resets

By definition, asynchronous sets and resets are independent of the clock and do not require an active clock edge. Therefore, you must include the set and reset signals in the sensitivity list of your process so they trigger the process to execute.
Sets and Resets

Chapter 9: VHDL Language Support

Sensitivity List

The sensitivity list is a list of signals (including ports) that, when there is an event (change in value), triggers the process to execute.

Syntax

```
process (clk_name, set_signal_name, reset_signal_name)
```

The signals are listed in any order, separated by commas.

Example: process Template with Asynchronous, Active-high reset, set

```vhdl
process (clk, reset, set)
begin
  if reset = '1' then
    -- Reset the outputs to zero.
    elsif set = '1' then
      -- Set the outputs to one.
      elsif rising_edge(clk) then
        -- Rising clock edge clock
        -- Clocked logic goes here.
        end if;
  end if;
end process;
```

Example: D Flip-flop with Asynchronous, Active-high reset, set

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity dff1 is
  port (data, clk, reset, set : in std_logic;
        qrs: out std_logic);
end dff1;
architecture async_set_reset of dff1 is
begin
  setreset: process(clk, reset, set)
  begin
    if reset = '1' then
      qrs <= '0';
    elsif set = '1' then
      qrs <= '1';
    elsif rising_edge(clk) then
      qrs <= data;
    end if;
  end process setreset;
end async_set_reset;
```
Synchronous Sets and Resets

Synchronous sets and resets set flip-flop outputs to logic ‘1’ or ‘0’ respectively on an active clock edge.

Do not list the set and reset signal names in the sensitivity list of a process because they should not trigger the process to execute upon changing. Instead, trigger the process when the clock signal changes, and then check the reset and set as the first statements.

RTL View Primitives

The VHDL compiler can detect and extract the following flip-flops with synchronous sets and resets and display them in the RTL schematic view:

- `sdffr`—flip-flop with synchronous reset
- `sdffs`—flip-flop with synchronous set
- `sdffrs`—flip-flop with both synchronous set and reset
- `sdffpat`—vectored flip-flop with synchronous set/reset pattern
- `sdffre`—enabled flip-flop with synchronous reset
- `sdffse`—enabled flip-flop with synchronous set
- `sdffpate`—enabled, vectored flip-flop with synchronous set/reset pattern

Note: You can check the name (type) of any primitive by placing your mouse pointer over it in the RTL view; a tooltip displays the name.
Sensitivity List

The sensitivity list is a list of signals (including ports) that, when there is an event (change in value), triggers the process to execute.

Syntax

```
process (clk_signal_name)
```

Example: process Template with Synchronous, Active-high reset, set

```
process(clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      -- Set the outputs to '0'.
    elsif set = '1' then
      -- Set the outputs to '1'.
    else
      -- Clocked logic goes here.
    end if;
  end if;
end process;
```
Example: D Flip-flop with Synchronous, Active-high reset, set

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity dff2 is
    port (data, clk, reset, set : in std_logic;
            qrs : out std_logic);
end dff2;
architecture sync_set_reset of dff2 is
begin
    setreset: process (clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                qrs <= '0';
            elsif set = '1' then
                qrs <= '1';
            else
                qrs <= data;
            end if;
        end if;
    end process setreset;
end sync_set_reset;
```
State Machines

State Machine Guidelines

A finite state machine (FSM) is hardware that advances from state to state at a clock edge.

The Synplify synthesis tool works best with synchronous state machines. The recommended practice is to write a fully synchronous design, avoiding asynchronous paths such as those through the asynchronous reset of a register. See *Asynchronous State Machines in VHDL* on page 9-54 for information about asynchronous state machines.

The following are guidelines for coding FSMs:

- The state machine must have a synchronous or asynchronous reset to be inferred. State machines must have an asynchronous or synchronous reset to set the hardware to a valid state after power-up, and to reset your hardware during operation (asynchronous resets are available freely in most FPGA architectures).

- The synthesis tool does not infer implicit state machines that are created using multiple `wait` statements in a `process`.

- Separate the sequential `process` statements from the combinational ones. Besides making it easier to read, it makes what is being registered very obvious. It also gives better control over the type of register element used.

- Represent states with defined labels or enumerated types.

- It is easiest to use a `case` statement in a `process` to check the current state at the clock edge, advance to the next state, and set the output values. You can also use `if-then-else` statements.

- Assign default values to outputs derived from the FSM before the `case` statement. This helps prevent the generation of unwanted latches and makes it easier to read because there is less clutter from rarely used signals.

- If you do not have `case` statements for all possible combinations of the selector, use a `when others` assignment as the last assignment in your `case` statement and set the state vector to some valid state. If your state vector is not an enumerated type, set the value to `X`. Assign the
state to X in the default clause of the case statement, to avoid mismatches between pre- and post-synthesis simulations. See Example: Default Assignment on page 9-52.

- Override the default encoding style with the syn_encoding attribute. The default encoding is determined by the number of states. However, if you have a syn_encoding attribute, its value is used during the mapping stage to determine encoding style.

```vhdl
attribute syn_encoding : string;
attribute syn_encoding of <typename> : type is "sequential";
```

See Chapter 7, Synthesis Attributes and Directives, for details about the syntax and values.

One-hot implementations are not always the best choice for state machines, even in FPGAs and CPLDs. For example, one-hot state machines might result in higher speeds in CPLDs, which can cause fitting problems because of the larger number of global signals. An example in an FPGA in which one-hot implementation is most ineffective is when the state machine drives a large decoder, generating many output signals. In a 16-state state machine, for example, the output decoder logic might reference sixteen signals in a one-hot implementation, but only four signals in an encoded representation.

In general, do not use syn_enum_encoding to set the encoding style. Use syn_encoding instead. The value of syn_enum_encoding is used by the compiler to interpret the enumerated data types but is ignored by the mapper when the state machine is actually implemented. syn_enum_encoding affects the final circuit only when you have turned off the FSM Compiler. Therefore, if you are not using FSM Compiler or the syn_state_machine attribute, both of which use syn_encoding, you can use syn_enum_encoding to set the encoding style. See Chapter 7, Synthesis Attributes and Directives, for details about the syntax and values.
Example: FSM Coding Style

```vhdl
architecture behave of test is
  type state_value is (deflt, idle, read, write);
  signal state, next_state: state_value;
begin
  -- Figure out the next state
  process (clk, rst)
  begin
    if rst = '0' then
      state <= idle;
    elsif rising_edge(clk) then
      state <= next_state;
    end if;
  end process;

  process (state, enable, data_in)
  begin
    data_out <= '0';
    -- Catch missing assignments to next_state
    next_state <= idle;
    state0 <= '0';
    state1 <= '0';
    state2 <= '0';
    case state is
      when idle =>
        if enable = '1' then
          state0 <= '1'; data_out <= data_in(0);
          next_state <= read;
        else
          next_state <= idle;
        end if;
      when read =>
        if enable = '1' then
          state1 <= '1'; data_out <= data_in(1);
          next_state <= write;
        else
          next_state <= read;
        end if;
      when deflt =>
        if enable = '1' then
          state2 <= '1'; data_out <= data_in(2);
          next_state <= idle;
        else
          next_state <= write;
    end case;
  end process;
end
```

Example: Default Assignment

The second others keyword in the following example pads (covers) all the bits. In this way, you do not need to remember the exact number of X's needed for the state variable or output signal.

```vhdl
when others =>
  state := (others => 'X');
```

Assigning X to the state variable (which is a “don’t-care” for synthesis) tells the Synplify synthesis tool that you have specified all the used states in your case statement, and any unnecessary decoding and gates related to other cases can therefore be removed. You do not have to add any special, non-VHDL directives.

If you set the state to a used state for the when others case (for example: `when others => state <= deflt`), the synthesis tool generates the same logic as if you assign X, but there will be pre- and post-synthesis simulation mismatches until you reset the state machine. These mismatches occur because all inputs are unknown at start up on the simulator. You therefore go immediately into the when others case, which sets the state variable to state1. When you power up the hardware, it can be in a used state, such as state2, and then advance to a state other than state1. Post-synthesis simulation behaves more like hardware with respect to initialization.

Using Enumerated Types for State Values

Generally, you represent states in VHDL with a user-defined enumerated type.

**Syntax**

```vhdl
type type_name is (state1_name, state2_name, ..., stateN_name);
```
Example

type states is (st1, st2, st3, st4, st5, st6, st7, st8);
begnin
-- The statement region of a process or subprogram.
next_state := st2 ;
-- Setting the next state to st2

Simulation Tips When Using Enumerated Types

Initialization in simulation should mimic the behavior of hardware when it powers up. Therefore, you should not initialize your state machine to a known state during simulation, because the hardware will not be in a known state when it powers up.

Creating an Extra Initialization State

If you use an enumerated type for your state vector, create an extra initialization state in your type definition (for example, stateX), and place it first in the list, as shown in the example below.

type state is (stateX, state1, state2, state3, state4);

In VHDL, the default initial value for an enumerated type is the leftmost value in the type definition (in this example, stateX). When you begin the simulation, you will be in this initial (simulation only) state.

Detecting Reset Problems

In your state machine case statement, create an entry for staying in stateX when you get in stateX. For example:

when stateX => next_state := stateX;

Look for your design entering stateX. This means that your design is not resetting properly.

Note: The Synplify synthesis tool does not create hardware to represent this initialization state (stateX). It is removed during optimization.
Detecting Forgotten Assignment to the Next State

Assign your next state value to stateX right before your state machine case statement.

Example

```vhdl
next_state := stateX;
case (current_state) is
  ... when state3 =>
    if (foo = '1') then
      next_state := state2;
    end if;
  ...
end case;
```

Asynchronous State Machines in VHDL

Avoid defining asynchronous state machines in VHDL. An asynchronous state machine has states, but no clearly defined clock, and has combinational loops. However, if you must use asynchronous state machines, you can do one of the following.

- Create a netlist of the technology primitives from your technology-vendor's target library. Any instantiated primitives that are left in the netlist are not removed during optimization.
- Use a schematic editor for the asynchronous state machine part of your design.

Do not use the Synplify synthesis tool to design asynchronous state machines; the tool might remove your hazard-suppressing logic when it performs logic optimization, causing your asynchronous state machine to work incorrectly.

The synthesis tool displays a “found combinational loop” warning message for an asynchronous FSM when it detects combinational loops in continuous assignment statements, processes and built-in gate-primitive logic.
Asynchronous State Machines that Generate Error Messages

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity async is
-- output is a buffer mode so that it can be read
  port (output : buffer std_logic ;
       g, d : in std_logic);
end async;
```

Asynchronous FSM from Concurrent Assignment Statement

```vhdl
architecture async1 of async is
begin
  output <= (((((g and d) or (not g)) and output) or d) and
              output);
end async1;
```

Asynchronous FSM Created with a process

```vhdl
architecture async2 of async is
begin
  process(g, d, output)
  begin
    output <= (((((g and d) or (not g)) and output) or d) and
                output);
  end process;
end async2;
```
Hierarchical Designs

Creating a Hierarchical Design in VHDL

Creating hierarchy is similar to creating a schematic. You place available parts from a library onto a schematic sheet and connect them.

To create a hierarchical design in VHDL, you instantiate one design unit inside of another. In VHDL, the design units you instantiate are called components. Before you can instantiate a component, you must declare it (step 2, below).

The basic steps for creating a hierarchical VHDL design are:

1. Write the design units (entities and architectures) for the parts you wish to instantiate.

2. Declare the components (entity interfaces) you will instantiate.

3. Instantiate the components, and connect (map) the signals (including top-level ports) to the formal ports of the components to wire them up. For examples, see VHDL Examples on page B-44 of the Synplify Reference Manual.

Step 1 – Write Entities and Architectures

Write entities and architectures for the design units to be instantiated.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity muxhier is
  port (outvec: out std_logic_vector (7 downto 0);
        a_vec, b_vec: in std_logic_vector (7 downto 0);
        sel: in std_logic);
end muxhier;
architecture mux_design of muxhier is
begin
  -- <mux functionality>
end mux_design;
```
library ieee;
use ieee.std_logic_1164.all;
entity reg8 is
  port (q: buffer std_logic_vector (7 downto 0);
        data: in std_logic_vector (7 downto 0);
        clk, rst: in std_logic);
end reg8;
architecture reg8_design of reg8 is -- Eight bit register
begin
  -- <Eight bit register functionality>
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;
entity rotate is
  port (q: buffer std_logic_vector (7 downto 0);
        data: in std_logic_vector (7 downto 0);
        clk, rst, r_l: in std_logic);
end rotate;
architecture rotate_design of rotate is
begin
  -- Rotates bits or loads
  -- When r_l is high, it rotates; if low, it loads data
  -- <Rotation functionality>
end rotate_design;

Step 2 – Declare the Components

Components are declared in the declarative region of the architecture with a component declaration statement.

The component declaration syntax is:

    component entity_name
      port (port_list);
    end component;

The entity_name and port_list of the component must match exactly that of the entity you will be instantiating.
Example

architecture structural of top_level_design is
  -- Component declarations are placed here in the
  -- declarative region of the architecture.

component muxhier -- Component declaration for mux
  port (outvec: out std_logic_vector (7 downto 0);
        a_vec, b_vec: in std_logic_vector (7 downto 0);
        sel: in std_logic);
end component;

component reg8  -- Component declaration for reg8
  port (q: out std_logic_vector (7 downto 0);
        data: in std_logic_vector (7 downto 0);
        clk, rst: in std_logic);
end component;

component rotate -- Component declaration for rotate
  port (q: buffer std_logic_vector (7 downto 0);
        data: in std_logic_vector (7 downto 0);
        clk, rst, r_l: in std_logic);
end component;

begin
  -- The structural description goes here.
end structural;

Step 3 – Instantiate the Components

Use the following syntax to instantiate your components:

    unique_instance_name : component_name
      [generic map (override_generic_values )]
      port map (port_connections);

You can connect signals either with positional mapping (the same order
declared in the entity) or with named mapping (in which you specify the
name of the lower-level signals to which connections will be made).
Connecting by name minimizes errors, and is recommended when the
component has many ports. To use configuration specification and decla-
ration, refer to Configuration Specification and Declaration on page 9-60.
Example

library ieee;
use ieee.std_logic_1164.all;
entity top_level is
  port (q: buffer std_logic_vector (7 downto 0);
        a, b: in std_logic_vector (7 downto 0);
        sel, r_l, clk, rst: in std_logic);
end top_level;

architecture structural of top_level is
  -- The component declarations shown in Step 2 go here.
  -- Declare the internal signals here
  signal mux_out, reg_out: std_logic_vector (7 downto 0);
begin
  -- The structural description goes here.
  -- Instantiate a mux, name it inst1, and wire it up.
  -- Map (connect) the ports of the mux using positional association.
  inst1:  muxhier port map (mux_out, a, b, sel);

  -- Instantiate a rotate, name it inst2, and map its ports.
  inst2: rotate port map (q, reg_out, clk, r_l, rst);

  -- Instantiate a reg8, name it inst3, and wire it up.
  -- reg8 is connected with named association.
  -- The port connections can be given in any order.
  -- Note that the actual (local) signal names are on
  -- the right of the '=' mapping operators, and the
  -- formal signal names from the component
  -- declaration are on the left.
  inst3: reg8 port map (couk => clk,
                        data => mux_out,
                        q => reg_out,
                        rst => rst);
end structural;
Configuration Specification and Declaration

A configuration declaration or specification can be used to define binding information of component instantiations to design entities (entity-architecture pairs) in a hierarchical design. After the structure of one level of a design has been fully described using components and component instantiations, a designer must describe the hierarchical implementation of each component.

A configuration declaration or specification can also be used to define binding information of design entities (entity-architecture pairs) that are compiled in different libraries.

This section discusses usage models of the configuration declaration statement supported by the Synplify synthesis tool. The following topics are covered:

- Configuration Specification
- Configuration Declaration

Component declarations and component specifications are not required for a component instantiation where the component name is the same as the entity name. In this case, the entity and its last architecture denote the default binding. In direct-entity instantiations, the binding information is available as the entity is specified, and the architecture is optionally specified. Configuration declaration and/or configuration specification are required when the component name does not match the entity name. If configurations are not used in this case, VHDL simulators give error messages, and the synthesis tool creates a black box and continues synthesis.

Configuration Specification

A configuration specification associates binding information with component labels that represent instances of a given component declaration. A configuration specification is used to bind a component instance to a design entity, and to specify the mapping between the local generics and ports of the component instance and the formal generics and ports of the entity. Optionally, a configuration specification can bind an entity to one
of its architectures. The Synplify synthesis tool supports a subset of configuration specification commonly used in RTL synthesis; this section discusses that support.

The following Backus-Naur Form (BNF) grammar is supported (VHDL-93 LRM pp.73-79):

```
configuration_specification ::= 
    for component_specification binding_indication ;

component_specification ::= 
    instantiation_list : component_name

instantiation_list ::= 
    instantiation_label {, instantiation_label } | others | all

binding_indication ::= [ use entity_aspect ]

entity_aspect ::= 
    entity entity_name [( architecture_identifier )] | 
    configuration configuration_name
```

```
for L1: XOR_GATE use entity work.XOR_GATE(behavior);  
for others: AND_GATE use entity work.AND_GATE(structure);  
for all: XOR_GATE use entity work.XOR_GATE;
```

**Example: Configuration Specification**

In the following example, two architectures (RTL and structural) are defined for an adder. There are two instantiations of an adder in design top. A configuration statement defines which adder architecture to use for each instantiation.
library IEEE;
use IEEE.std_logic_1164.all;
entity adder is
  port ( a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         s : out std_logic;
         cout : out std_logic);
end adder;

library IEEE;
use IEEE.std_logic_unsigned.all;
arquitectura rtl of adder is
  signal tmp : std_logic_vector(1 downto 0);
  begin
    tmp <= ('0' & a) + b + cin;
    s <= tmp(0);
    cout <= tmp(1);
  end rtl;

arquitectura structural of adder is
  begin
    s <= a xor b xor cin;
    cout <= ((not a) and b and cin) or ( a and (not b) and cin)
            or (a and b and (not cin)) or ( a and b and cin);
  end structural;

library IEEE;
use IEEE.std_logic_1164.all;
entity top is
  port ( a : in std_logic_vector(1 downto 0);
         b : in std_logic_vector(1 downto 0);
         c : in std_logic;
         cout : out std_logic;
         sum : out std_logic_vector(1 downto 0));
end top;

arquitectura top_a of top is
  component myadder
    port ( a : in std_logic;
           b : in std_logic;
           cin : in std_logic;
           s : out std_logic;
           cout : out std_logic);
  end component;
signal carry : std_logic;
for s1 : myadder use entity work.adder(structural);
for r1 : myadder use entity work.adder(rtl);
begin
  s1 : myadder port map ( a(0), b(0), c, sum(0), carry);
  r1 : myadder port map ( a(1), b(1), carry, sum(1), cout);
end top_a;

Results

for s1 : myadder use entity work.adder(structural); end for;

for r1 : myadder use entity work.adder(rtl); end for;

Unsupported Constructs for Configuration Specification

The following are the configuration specification constructs that are not supported by the Synplify synthesis tool. Appropriate messages are issued in the log file when these constructs are used.

1. The VHDL-93 LRM defines the binding indication in the configuration statement as:

   binding_indication ::=  
     [use entity_aspect]  
     [generic_map_aspect]  
     [port_map_aspect]
The synthesis tool supports the entity_aspect of a binding indication. It does not yet support the generic_map_aspect and port_map_aspect. If used, a warning is issued, and the tool ignores the generic and port maps.

2. The VHDL-93 LRM defines entity_aspect in the binding indication as:

   entity_aspect ::= 
   
   entity entity_name [ ( architecture_identifier) ] | 
   configuration configuration_name | open 

   The synthesis tool supports entity_name and configuration_name in the entity_aspect of a binding indication. The tool does not yet support the open construct.

Configuration Declaration

Configuration declaration specifies binding information of component instantiations to design entities (entity-architecture pairs) in a hierarchical design. Configuration declaration can bind component instantiations in an architecture, in either a block statement, a for...generate statement or an if...generate statement. It is also possible to bind different entity-architecture pairs to different indices of a for...generate statement.

The Synplify synthesis tool supports a subset of configuration declaration commonly used in RTL synthesis. The following Backus-Naur Form (BNF) grammar is supported (VHDL-93 LRM pp.11-17):

   configuration_declaration ::= 
   
   configuration identifier of entity_name is 
   
   block_configuration 
   
   end [ configuration ] [configuration_simple_name ] ; 

   block_configuration ::= 
   
   for block_specification 
   
   { configuration_item } 
   
   end for ;
The BNF grammar for component_specification and binding_indication is described in Configuration Specification on page 9-60.

**Example 1: Configuration Declaration**

The following example shows a configuration declaration describing the binding in a 3-level hierarchy, for...generate statement labeled label1, within block statement blk1 in architecture arch_gen3. Each architecture implementation of an instance of my_and1 is determined in the configuration declaration and depends on the index value of the instance in the for...generate statement.

```vhdl
entity and1 is
  port(a,b: in bit ; o: out bit);
end;

architecture and_arch1 of and1 is
begin
  o <= a and b;
end;

architecture and_arch2 of and1 is
begin
  o <= a and b;
end;
```

architecture and_arch3 of and1 is
begin
  o <= a and b;
end;

library WORK; use WORK.all;
entity gen3_config is
  port(a,b: in bit_vector(0 to 7);
       res: out bit_vector(0 to 7));
end;

library WORK; use WORK.all;
architecture arch_gen3 of gen3_config is
  component my_and1 port(a,b: in bit; o: out bit); end component;
begin
  label1: for i in 0 to 7 generate
    blk1: block
    begin
      a1: my_and1 port map(a(i),b(i),res(i));
    end block;
  end generate;
end;

library work;
configuration config_gen3_config of gen3_config is
  for arch_gen3 -- ARCHITECTURE block_configuration "for
      block_specification"
    for label1 (0 to 3) --GENERATE block_config "for
      block_specification"
      for blk1 -- BLOCK block_configuration "for
        block_specification"
        -- {configuration_item}
        for a1 : my_and1 -- component_configuration
          -- Component_specification "for idList : compName"
          use entity work.and1(and_arch1); --
          binding_indication
          end for; -- a1 component_configuration
    end for; -- blk1 BLOCK block_configuration
end for; -- label1 GENERATE block_configuration
for label1 (4) -- GENERATE block_configuration "for
      block_specification"
  for blk1
    for a1 : my_and1
      use entity work.and1(and_arch3);
    end for;
  end for;
end;
Example 2: Configuration Declaration

In the following example, two architectures (RTL and structural) are defined for an adder. There are two instantiations of an adder in design top. A configuration declaration defines which adder architecture to use for each instantiation. This example is similar to the configuration specification example.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity adder is
  port ( a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         s : out std_logic;
         cout : out std_logic);
end adder;

library IEEE;
use IEEE.std_logic_unsigned.all;
architecture rtl of adder is
  signal tmp : std_logic_vector(1 downto 0);
begin
  tmp <= ('0' & a) + b + cin;
  s <= tmp(0);
  cout <= tmp(1);
end rtl;

architecture structural of adder is
begin
  s <= a xor b xor cin;
  cout <= ((not a) and b and cin) or ( a and (not b) and cin) or
           (a and b and (not cin)) or ( a and b and cin);
end structural;
```
library IEEE;
use IEEE.std_logic_1164.all;
entity top is
    port ( a : in std_logic_vector(1 downto 0);
           b : in std_logic_vector(1 downto 0);
           c : in std_logic;
           cout : out std_logic;
           sum : out std_logic_vector(1 downto 0));
end top;
architecture top_a of top is
component myadder
    port ( a : in std_logic;
           b : in std_logic;
           cin : in std_logic;
           s : out std_logic;
           cout : out std_logic);
end component;
signal carry : std_logic;
bEGIN
    s1 : myadder port map ( a(0), b(0), c, sum(0), carry);
    r1 : myadder port map ( a(1), b(1), carry, sum(1), cout);
end top_a;
library work;
configuration config_top of top is  -- configuration_declaration
    for top_a  -- block_configuration "for block_specification"
        -- component_configuration
        for s1: myadder -- component_specification
            use entity work.adder (structural);  -- binding_indication
        end for;  -- component_configuration
        -- component_configuration
        for r1: myadder -- component_specification
            use entity work.adder (rtl);  -- binding_indication
        end for;  -- component_configuration
    end for;  -- block_configuration
end config_top;
Results

for s1 : myadder use entity work.adder (structural); end for;

for r1 : myadder use entity work.adder (rtl); end for;

Unsupported Constructs for Configuration Declaration

The following are the configuration declaration constructs that are not supported by the Synplify synthesis tool. Appropriate messages are displayed in the log file if these constructs are used.

1. The VHDL-93 LRM defines the configuration declaration as:

   configuration_declaration ::= 
     configuration identifier of entity_name is 
     configuration_declarative_part 
     block_configuration 
   end [ configuration ] [configuration_simple_name ] ; 

configuration_declarative_part ::= \{ configuration_declarative_item \} 

configuration_declarative_item ::= 
  use_clause | attribute_specification | group_declaration 

The synthesis tool does not support the configuration_declarative_part. It parses the use_clause and attribute_specification without any warning message. The group_declaration is not supported and an error message is issued.
2. VHDL-93 LRM defines entity aspect in the binding indication as:

\[
\text{entity_aspect ::= entity entity_name [ ( architecture_identifier) ] | configuration configuration_name | open}
\]

\[
\text{block_configuration ::= for block_specification { use_clause } { configuration_item } end for ;}
\]

The synthesis tool does not support use_clause in block_configuration. This construct is parsed and ignored.
Scalable Designs

Creating Scalable Designs

You can create a VHDL design that is scalable, meaning that it can handle a user-specified number of bits or components. Any of these methods can be used to create scalable designs:

- Unconstrained vector ports
- VHDL generics
- VHDL generates statements.

Creating a Scalable Design Using Unconstrained Vector Ports

Do not size (constrain) the ports until you need them. This first example is coding the adder using the + operator, and gives much better synthesized results than the second and third scalable design examples, which code the adders as random logic.

Example

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity addn is
  -- Note: the a, b, and result ports are not constrained.
  -- In VHDL, they automatically size to whatever is connected
  -- to them.
  port(result : out std_logic_vector;
       cout : out std_logic;
       a, b : in std_logic_vector;
       cin : in std_logic);
end addn;

architecture stretch of addn is
  signal tmp : std_logic_vector (a'length downto 0);
begin
  -- Note: this next line works because "+" sizes to
  -- the largest operand (also, you just need to pad one
  -- argument).
```

```vhdl
  -- Note: this next line works because "+" sizes to
  -- the largest operand (also, you just need to pad one
  -- argument).
```
tmp <= ('0' & a) + b + cin;
result <= tmp(a'length - 1 downto 0);
cout <= tmp(a'length);
assert result'length = a'length;
assert result'length = b'length;
end stretch;

-- Top level design
-- Here is where you specify the size for a, b,
-- and result. It is illegal to leave your top
-- level design ports unconstrained.

library ieee;
use ieee.std_logic_1164.all;
entity addntest is
  port (result : out std_logic_vector (7 downto 0);
      cout : out std_logic;
      a, b : in std_logic_vector (7 downto 0);
      cin : in std_logic);
end addntest;

architecture top of addntest is
component addn
  port (result : std_logic_vector;
        cout : std_logic;
        a, b : std_logic_vector;
        cin : std_logic);
end component;
begin
  test : addn port map (result => result,
                         cout => cout,
                         a => a,
                         b => b,
                         cin => cin);
end;

Creating a Scalable Design Using VHDL Generics

Create a VHDL generic with default value. The generic is used to represent
bus sizes inside a architecture, or a number of components. You can
define more than one generic per declaration by separating the generic
definitions with semicolons (;).
Scalable Designs

Chapter 9: VHDL Language Support

Syntax

    generic (generic_1_name : type [:= default_value]) ;

Examples

    generic (num : integer := 8) ;
    generic (top : integer := 16; num_bits : integer := 32);

Using a Scalable Architecture

Instantiate the scalable architecture, and override the default generic value with the generic map statement.

Syntax

    generic map (list_of_overriding_values)

Examples

Generic map construct

    generic map (16)
    -- These values will get mapped in order given.
    generic map (8, 16)

Creating a scalable adder

    library ieee;
    use ieee.std_logic_1164.all;
    entity adder is
        generic(num_bits : integer := 4); -- Default adder
        -- Size is 4 bits
        port (a : in std_logic_vector (num_bits downto 1);
             b : in std_logic_vector (num_bits downto 1);
             cin : in std_logic;
             sum : out std_logic_vector (num_bits downto 1);
             cout : out std_logic);
    end adder;
architecture behave of adder is
begin
    process (a, b, cin)
        variable vsum : std_logic_vector (num_bits downto 1);
        variable carry : std_logic;
    begin
        carry := cin;
        for i in 1 to num_bits loop
            vsum(i) := (a(i) xor b(i)) xor carry;
            carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));
        end loop;
    sum <= vsum;
    cout <= carry;
    end process;
end behave;

Scaling the Adder by Overriding the generic Statement
library ieee;
use ieee.std_logic_1164.all;
entity adder16 is
    port (a : in std_logic_vector (16 downto 1);
    b : in std_logic_vector (16 downto 1);
    cin : in std_logic;
    sum : out std_logic_vector (16 downto 1);
    cout : out std_logic);
end adder16;

architecture behave of adder16 is
begin
    for i in 1 to num_bits loop
        vsum(i) := (a(i) xor b(i)) xor carry;
    end loop;

library ieee;
use ieee.std_logic_1164.all;
entity adder16 is
    port (a : in std_logic_vector (16 downto 1);
    b : in std_logic_vector (16 downto 1);
    cin : in std_logic;
    sum : out std_logic_vector (16 downto 1);
    cout : out std_logic);
end adder16;

architecture behave of adder16 is
begin
    for i in 1 to num_bits loop
        vsum(i) := (a(i) xor b(i)) xor carry;
    end loop;

component adder
    -- The default adder size is 4 bits.
    generic(num_bits : integer := 4);
    port (a : in std_logic_vector;
    b : in std_logic_vector;
    cin : in std_logic;
    sum : out std_logic vector;
    cout : out std_logic);
end component;
begin
my_adder : adder
  generic map (16) -- Use a 16 bit adder
  port map(a, b, cin, sum, cout);
end behave;

Creating a Scalable Design Using Generate Statements

A VHDL generate statement allows you to repeat logic blocks in your design without having to write the code to instantiate each one individually.

Creating a 1-bit Adder

library ieee;  
use ieee.std_logic_1164.all;  
entity adder is  
  port (a, b, cin : in std_logic;
        sum, cout : out std_logic);
  end adder;
architecture behave of adder is
begin
  sum <= (a xor b) xor cin;
  cout <= (a and b) or (cin and a) or (cin and b);
end behave;

Instantiating the 1-bit Adder Many Times with a generate Statement

library ieee;  
use ieee.std_logic_1164.all;  
entity addern is  
  generic(n : integer := 8);
  port (a, b : in std_logic_vector (n downto 1);
        cin : in std_logic;
        sum : out std_logic_vector (n downto 1);
        cout : out std_logic);
  end addern;
architecture structural of addern is
-- The adder component declaration goes here.
component adder
  port (a, b, cin : in std_logic;
       sum, cout : out std_logic);
end component;
signal carry : std_logic_vector (0 to n);
begins -- Generate instances of the single-bit adder n times.
-- You don't need to declare the index 'i' because
-- indices are implicitly declared for all FOR
-- generate statements.
gen: for i in 1 to n generate
  add: adder port map(
    a => a(i),
    b => b(i),
    cin => carry(i - 1),
    sum => sum(i),
    cout => carry(i));
end generate;
carry(0) <= cin;
cout <= carry(n);
end structural;
RAM Inference

The Synplify synthesis tool can infer synchronous and synchronously resettable RAMs from your VHDL code. When appropriate, it can also generate technology-specific single or dual port RAMs. No special input is needed, such as attributes or directives in your source code. The primitive generated for an inferred RAM has an asynchronous READ, except where noted below, in *Synchronous READ RAMs on page 9-81*.

The following table lists the supported technology-specific RAMs that can be generated by the Synplify synthesis tool.

Table 9-10: Generated technology-specific RAMs

<table>
<thead>
<tr>
<th>Technology vendor</th>
<th>Family</th>
<th>RAM Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera</td>
<td>APEX20K/20KE/20KC</td>
<td>Single and dual port</td>
</tr>
<tr>
<td>Altera</td>
<td>FLEX10K/10KE</td>
<td>Single port</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Virtex, Virtex-E, Virtex2, Virtex2p</td>
<td>Single and dual port. Dual port block RAMs have only one write port.</td>
</tr>
</tbody>
</table>

**Example**

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

entity ram_test is
  port (
    d : in std_logic_vector(7 downto 0);
    a : in std_logic_vector(6 downto 0);
    we : in std_logic;
    clk : in std_logic;
    q : out std_logic_vector(7 downto 0));
end ram_test;

architecture rtl of ram_test is
  type mem_type is array (127 downto 0) of
    std_logic_vector (7 downto 0);
begin
```

Synplify Reference Manual, April 2002
signal mem: mem_type;
begin
process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        mem(conv_integer (a)) <= d;
      end if;
    end if;
  end process;
q <= mem(conv_integer (a));
end rtl;

RAMs with Special Write Enables

The Synplify synthesis tool can infer RAMs when the Write Enable is tied to Vcc or when the Write Enable is nested within if statements.

always-enabled Write Enable

The RAM extraction code supports the inference of RAMs with their Write Enable tied permanently to Vcc rather than implementing the logic in flip-flops.
Example

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

entity WEtrue is
    generic (ram_depth : integer := 1024;
             addr_width : integer := 10;
             data_width : integer := 8);
    port (clk : in std_logic;
          addr : in std_logic_vector(addr_width-1 downto 0);
          data_in : in std_logic_vector(data_width-1 downto 0);
          data_out : out std_logic_vector(data_width-1 downto 0));
end WEtrue;

architecture rtl of WEtrue is
    type ram_type is array(ram_depth-1 downto 0)of
        std_logic_vector(data_width-1 downto 0);
    signal ram : ram_type;
    signal we : std_logic;

    begin
    we <= '1';
    process(clk)
    begin
        if(clk'event and clk='1') then
            if(we = '1') then
                ram(CONV_INTEGER(addr)) <= data_in;
            end if;
        end if;
    end process;
    data_out <= ram(CONV_INTEGER(addr));
end rtl;
```

**Nested Write Enable**

The RAM extraction code can infer RAMs when the Write Enable is more complex as found in nested IF statements. The compilers extract common terms from the feedback MUX Enables to derive the common Write Enable signal.
Example

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

entity WEnestedif is
  generic (ram_depth : integer := 1024;
            addr_width : integer := 10;
            data_width : integer := 8);
  port( addr : in std_logic_vector(addr_width-1 downto 0);
        data_in : in std_logic_vector(data_width-1 downto 0);
        we1, we2, clk: in std_logic;
        data_out : out std_logic_vector(data_width-1 downto 0));

end WEnestedif;

architecture rtl of WEnestedif is
  type ram_type is array(ram_depth-1 downto 0) of
                     std_logic_vector(data_width-1 downto 0);
  signal ram : ram_type;

begin
  data_out <= ram(conv_integer(addr));
  process(clk, addr, we1, we2)
  begin
    if(clk'event and clk='1') then
      if(we1 = '1') then
        if (we2 = '1') then
          ram(conv_integer(addr)) <= data_in;
        end if;
      end if;
    end if;
  end process;
end rtl;
```

Limited RAM Resources

If your RAM resources are limited, designate additional instances of inferred RAMs as flip-flops and logic using the `syn_ramstyle` attribute. This attribute takes the string argument of `registers`, and places it on either the output signal driven by the RAM or on the instance name of the RAM. Alternatively, you can use the Attributes panel of the SCOPE spreadsheet to assign this and all other attributes. Using the spreadsheet lets you modify attributes without requiring you to recompile the source code.
Additional Components Generated

After inferring a RAM for some technologies, you might notice that the Synplify synthesis tool has generated a few additional components adjacent to the RAM. These components assure accuracy in your post placement and routing simulation results.

Synchronous READ RAMs

All RAM primitives that the Synplify synthesis tool generates for inferred RAMs have asynchronous READs, except for single port Altera 10K RAMs and Xilinx Virtex block SelectRAM+ devices. For these exceptions, the tool generates a synchronous RAM. The following example shows how the address of the RAM must be registered to generate a synchronous READ RAM.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_test is
  port (  
    d : in std_logic_vector(7 downto 0);  
    a : in std_logic_vector(6 downto 0);  
    we : in std_logic;  
    clk : in std_logic;  
    q : out std_logic_vector(7 downto 0));
end ram_test;
architecture rtl of ram_test is
  type mem_type is array (127 downto 0) of    
    std_logic_vector (7 downto 0);
  signal mem: mem_type;
  signal read_add : std_logic_vector(6 downto 0);
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        mem(conv_integer(a)) <= d;
      end if;
      read_add <= a;
    end if;
  end process;
  q <= mem(conv_integer(read_add));
end rtl;
```
Chapter 9: VHDL Language Support

ROM Inference

As part of B.E.S.T. (Behavioral Extraction Synthesis Technology), the Synplify synthesis tool infers ROMs (read-only memories) from your HDL source code, and generates block components in the RTL view for them.

The data contents of the ROMs are stored in a text file named `rom.info`. To quickly view `rom.info` in read-only mode, synthesize your HDL source code, open an RTL view, and then push down into the ROM component. For an example of the ROM data, refer to `ROM Table Data (rom.info File)` on page 8-39.

Generally, the Synplify synthesis tool infers ROMs from HDL source code that uses `case` statements, or equivalent `if` statements, to make 16 or more signal assignments using constant values (words). The constants must all be the same width.

Another requirement for ROM inference is that values must be specified for at least half of the address space. For example, if the ROM has 5 address bits, then the address space is 32 and at least 16 of the different addresses must be specified.

Example

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity rom4 is
  port (a : in std_logic_vector(4 downto 0);
        z : out std_logic_vector(3 downto 0))
end rom4;
architecture behave of rom4 is
begin
  process(a)
  begin
    if a = "00000" then
      z <= "0001";
    elsif a = "00001" then
      z <= "0010";
    elsif a = "00010" then
      z <= "0110";
    elsif a = "00011" then
      z <= "1010";
```
elsif a = "00100" then
  z <= "1000";
elsif a = "00101" then
  z <= "1001";
elsif a = "00110" then
  z <= "0000";
elsif a = "00111" then
  z <= "1110";
elsif a = "01000" then
  z <= "1111";
elsif a = "01001" then
  z <= "1110";
elsif a = "01010" then
  z <= "0001";
elsif a = "01011" then
  z <= "1000";
elsif a = "01100" then
  z <= "1110";
elsif a = "01101" then
  z <= "0011";
elsif a = "01110" then
  z <= "1111";
elsif a = "01111" then
  z <= "1100";
elsif a = "10000" then
  z <= "1000";
elsif a = "10001" then
  z <= "0000";
elsif a = "10010" then
  z <= "0011";
elsif a = "10011" then
  z <= "1111";
else
  z <= "0111";
end if;
end process;
end behave;

ROM Table Data (rom.info file)

Note: This data is for viewing only

ROM work.rom4(behave)-z_1[3:0]
address width: 5
data width: 4
inputs:
0: a[0]
1: a[1]
Chapter 9: VHDL Language Support

ROM Inference

2: a[2]
3: a[3]
4: a[4]
outputs:
0: z_1[0]
1: z_1[1]
2: z_1[2]
3: z_1[3]
data:
00000 -> 0001
00001 -> 0010
00010 -> 0110
00011 -> 1010
00100 -> 1000
00101 -> 1001
00110 -> 0000
00111 -> 1110
01000 -> 1111
01001 -> 1110
01010 -> 0001
01011 -> 1000
01100 -> 1110
01101 -> 0011
01110 -> 0010
01111 -> 0010
10000 -> 0010
10001 -> 0010
10010 -> 0010
default -> 0111
Synthesis Attributes and Directives

VHDL synthesis attributes and directives allow you to associate information with your design to control the way it is analyzed, compiled, and mapped.

- **Attributes** direct the way your design is optimized and mapped during synthesis. Although you can place synthesis attributes directly in your source code, you should use the Attributes panel of the SCOPE spreadsheet to specify them in a constraint file. That way, changes can be made (via the constraint file), without requiring you to recompile.

- **Directives** control the way your design is analyzed prior to synthesis. Because of this, they must be included directly in your VHDL source code and cannot be specified in a constraint file.

Only directives are required to be specified in your source code (attributes can be specified in a constraint file).

In addition to the general attributes and directives described in Chapter 7, *Synthesis Attributes and Directives*, the Synplify synthesis tool also supports vendor-specific directives and attributes that apply to the specific programmable logic vendor you are targeting for your design.

The VHDL attributes and directives are predefined in a package provided in the Synplify synthesis tool library (installation_dir/lib/vhd/synattr.vhd). You can either use this package (recommended) or redefine the attributes and directives each time you want to include them in source code. The library contains the built-in attributes, along with declarations for timing constraints (including black-box timing constraints) and vendor-specific attributes. To access the library, add the following to the beginning of each of the VHDL design units that uses the attributes:

```vhdl
library synplify;
use synplify.attributes.all;
```
Instantiating Black Boxes in VHDL

Black boxes are design units in which just the interface is specified; internal information is ignored by the Synplify synthesis tool. Black boxes can be used to directly instantiate:

- Technology-vendor primitives and macros (including I/Os).
- User-defined macros whose functionality was defined in a schematic editor, or another input source (when the place-and-route tool can merge design netlists from different sources).

Black boxes are specified with the `syn_black_box` synthesis directive, in conjunction with other directives. If the black box is a technology-vendor I/O pad, use the `black_box_pad_pin` directive instead. Here is a list of the directives that you can use to specify modules as black boxes, and to define design objects on the black box for consideration during synthesis:

- `syn_black_box`
- `black_box_pad_pin`
- `black_box_tri_pins`
- `syn_isclock`
- `syn_tco<n>`
- `syn_tpd<n>`
- `syn_tsu<n>`

For descriptions of the black-box attributes and directives, see Chapter 7, *Synthesis Attributes and Directives*.

Black-Box Timing Constraints

You can provide timing information for your individual black box instances. The following are the three predefined timing constraints available for black boxes.

- `syn_tpd<n>` – Timing propagation for combinational delay through the black box.
• syn_tsu<\text{n}> – Timing setup delay required for input pins (relative to the clock).

• syn_tco<\text{n}>– Timing clock to output delay through the black box.

Here, \text{n} is an integer from 1 through 10, inclusive. See Black-box Source Code Directives on page 6-36, for details about constraint syntax.

**Instantiating a Black Box**

Before instantiating a black box, you must define its functionality using a predefined macro or by designing it. Although the Synplify synthesis tool ignores the internals of black-box design units, you must define the functionality if you plan to simulate with a VHDL simulator.

Most programmable logic architectures provide macro libraries with predefined black boxes for primitives and macros (including I/Os). Refer to the section for your vendor for more information about predefined macros.

To instantiate a black box:

1. Create a component declaration for the black-box user macro or technology-vendor cell.

2. Declare the syn_black_box attribute to be a Boolean attribute.

3. Define the syn_black_box attribute on the black-box component to be TRUE.

**Instantiating a Technology-vendor I/O**

To instantiate a technology-vendor I/O:

1. Create a component declaration for your technology-vendor pad.

2. Declare the black_box_pad_pin directive to be a string attribute.

3. Define the black_box_pad_pin directive on the component to be the external pin name for the pad.
Example

```vhdl
library synplify;
entity top is
    port (clk, rst, en, data: in bit; q: out bit);
end top;

architecture structural of top is
-- In this example, GIZMO is a user-defined macro
-- that was created in a schematic editor, and that
-- you want to directly instantiate in your design
-- as a black box.
component GIZMO
    port(Q: out bit; D, C, CLR: in bit);
end component;
-- Set the syn_black_box attribute on GIZMO.
attribute syn_black_box : boolean;
attribute syn_black_box of GIZMO:  component is true;

-- In this example, MYBUF is a user I/O macro that
-- was created in a schematic editor, and that
-- you want to directly instantiate in your
-- design as a black box.
component MYBUF
    port(O: out bit; I: in bit);
end component;
-- Set the black_box_pad_pin attribute on MYBUF
-- to the pin, "I", that interfaces with the
-- external world.
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of MYBUF:  component is "I";
signal data_core: bit;

begin
-- Instantiate an MYBUF. Map (connect) data to I and
-- data_core to O.
data_pad: MYBUF port map (
    O => data_core,
    I => data);
-- Instantiate a GIZMO and map (connect) the ports.
my_gizmo: GIZMO port map (
    Q => q,
    D => data_core,
    C => clk,
    CLR => rst);
end structural;
```
VHDL Synthesis Examples

This section lists the VHDL synthesis examples included with the tool.

Combinational Logic Examples

The following combinational logic synthesis examples are included in the Synplify_install_dir/examples/vhdl/combinat directory:

- Adders
- ALU
- Bus Sorter (illustrates using procedures in VHDL)
- 3-to-8 Decoders
- 8-to-3 Priority Encoders
- Comparator
- Interrupt Handler (coded with an if-then-else statement for the desired priority encoding)
- Multiplexors (concurrent signal assignments, case statements, or if-then-else statements can be used to create multiplexors; the Synplify synthesis tool automatically creates parallel multiplexors when the conditions in the branches are mutually exclusive)
- Parity Generator
- Tristate Drivers
Sequential Logic Examples

The following sequential logic synthesis examples are included in the `synplify_install_dir/examples/vhdl/sequential` directory:

- Flip-flops and level-sensitive latches
- Counters (up, down, and up/down)
- Register file
- Shift register
- State machines

For additional information on synthesizing flip-flops and latches, see:

- Creating Flip-flops and Registers Using VHDL Processes on page 9-36
- Level-sensitive Latches Using Concurrent Signal Assignments on page 9-40
- Level-sensitive Latches Using VHDL Processes on page 9-41
- Asynchronous Sets and Resets on page 9-44
- Synchronous Sets and Resets on page 9-46
PREP VHDL Benchmarks

PREP (Programmable Electronics Performance) Corporation distributes benchmark results that show how FPGA vendors compare with each other in terms of device performance and area.

The following PREP benchmarks are included in the synplify_install_dir/examples/vhdl/prep directory:

- PREP VHDL Benchmark 1. Data Path
- PREP VHDL Benchmark 2. Timer/Counter
- PREP VHDL Benchmark 3. Small State Machine
- PREP VHDL Benchmark 4. Large State Machine
- PREP VHDL Benchmark 5. Arithmetic Circuit
- PREP VHDL Benchmark 6. 16-Bit Accumulator
- PREP VHDL Benchmark 7. 16-Bit Counter
- PREP VHDL Benchmark 8. 16-Bit Pre-scaled Counter
- PREP VHDL Benchmark 9. Memory Map

The source code for the benchmarks can be used for design examples for synthesis or for doing your own FPGA vendor comparisons.

PREP Corp. has disbanded, but you can still obtain information from their Web site: www.prep.org.
APPENDIX A

Designing with Actel

This section discusses the following vendor-specific topics:

- Using Actel on page A-2
- Actel-specific Tcl Command Options on page A-8
- Actel Attribute and Directive Summary on page A-10
Using Actel

The Synplify synthesis tool creates technology-specific netlists for a number of Actel families of FPGAs. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

After synthesis, the synthesis tool generates EDIF netlists ready for the Actel Designer Series place-and-route tool.

The rest of this chapter discusses the following topics:

- Actel Features on page A-2
- Constraint Files on page A-2
- Synthesis Reports on page A-3
- Actel Device Mapping Options on page A-3
- Instantiating Macros and Black Boxes in Actel Designs on page A-5

Actel Features

The Synplify synthesis tool contains the following Actel-specific features:

- Direct mapping to Actel’s c-modules and s-modules
- Timing-driven mapping, replication, and buffering
- Inference of counters, adders, subtractors, etc., and module generation
- Automatic use of clock buffers for clocks and reset signals
- Automatic I/O insertion

Constraint Files

Optimize your design by using constraint files. Constraint files can contain timing constraints, general attributes, and Actel-specific attributes. Use the SCOPE spreadsheet to create and manage the files.
Add the constraint files to the project list along with your HDL source files. For more information about constraint files, refer to the chapter on timing constraints.

**Synthesis Reports**

The Synplify synthesis tool generates a resource usage report, a timing report, and a net buffering report for the Actel designs that you synthesize. To view the synthesis reports, click View Log.

**Actel Device Mapping Options**

You select device mapping options for Actel technologies from the Device tab in the Options for Implementation dialog box (click Impl Options in the Project window or select Project -> Implementation Options to display the dialog box). You can set the following:

- Fanout Limits
- I/O Insertion
- Maximum Number of Critical Paths (Actel 500K and PA only)

**Fanout Limits**

Large fanouts can cause large delays and routability problems. During technology mapping, the Synplify synthesis tool automatically maintains reasonable fanout limits, keeping the fanout under the limit you specify (a guideline limit is a limit that the synthesis tool tries to stay under, but which it might exceed if there are excessive speed or area costs).

**Setting a Fanout Guideline**

To set a guideline other than the default for the fanout limit:

1. Open the Device tab in the Options for Implementation dialog box.

2. Type in an integer for Fanout Guide. The value entered is a guideline for the number of fanouts for a given driver.

The fanout limit is only a guideline for the synthesis tool unless you specify it as a hard limit (see *Setting a Hard Fanout Limit on page A-4*).
The synthesis tool first reduces fanout by replicating the driver of the high fanout net and splitting the net into segments. Replication can affect the number of register bits in your design. If replication is not possible, the algorithm buffers the signals. Buffering is more expensive in terms of intrinsic delay and resource consumption, and is therefore not used unless the fanout exceeds the guideline. Timing also affects the fanout limit for a net. Critical nets can be replicated or buffered more aggressively.

The log file contains the net buffering report that shows how many nets were buffered or had their sources replicated and the number of segments created for each net.

**Setting a Hard Fanout Limit**

The Fanout Guide value is only a guideline. When you specify a hard limit, the Synplify synthesis tool will never exceed the Fanout Guide value. You set a hard fanout limit as follows:

1. Set the Fanout Guide value as previously described (see *Fanout Limits* on page A-3).

2. On the Device tab of the Options for Implementation dialog box, check the Hard limit to Fanout box.

3. Following synthesis, click the View Log button to display the net buffering report.

The Hard limit to Fanout option is not available for the PA family.

**I/O Insertion**

The Synplify synthesis tool inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist unless you disable I/O insertion. You can override which I/O pads are used by instantiating the Actel I/O pads directly. If you manually insert I/O pads, you only insert them for the pins that require them.

1. If you do not want to automatically insert any I/O pads, check the Disable I/O Insertion box on the Device tab of the Options for Implementation dialog box.

   This is useful to see how much area your blocks of logic take up, before synthesizing an entire FPGA. Note that if you disable
automatic I/O insertion, you will not get any I/O pads in your design unless you manually instantiate them yourself.

2. Manually instantiate I/O pads for specific pins, if you need them.

**Maximum Number of Critical Paths**

This feature, available for Actel 500K and PA only, allows you to set the maximum number of critical paths in a forward-annotated constraint file (.sdf). To set a maximum number of critical paths:

1. Open the Device tab in the Options for Implementation dialog box and make sure that the Actel 500K or PA technology is selected.

2. Enter a value for Max number of critical paths in SDF.

The default value is 4000. Due to varying design characteristics, experiment with a range of values to achieve the best circuit performance possible.

The .sdf file displays a prioritized list of the worst-case paths in a design. Actel Designer prioritizes routing to ensure that the worst-case paths are routed efficiently.

**Instantiating Macros and Black Boxes in Actel Designs**

You can instantiate ACTgen macros or other Actel macros like gates, counters, flip-flips, or I/Os by using the supplied Actel macro libraries to pre-define the Actel macro black boxes.

For general information on instantiating black boxes, see *Instantiating Black Boxes in VHDL* on page 9-86, and *Instantiating Black Boxes in Verilog* on page 8-40. For Actel-specific Verilog and VHDL details, see the following:

- Actel Macro Libraries (Verilog)
- Using ACTgen Macros with Verilog Designs
- Actel Macro Libraries (VHDL)
- Using ACTgen Macros with VHDL Designs
Actel Macro Libraries (Verilog)

The Actel macro libraries contain pre-defined black boxes for the Actel macros so that you can manually instantiate them in your design. To instantiate a macro, add the appropriate Actel macro library filename at the top of the source files list. The files for the Actel macro libraries are located in the `synplify_install_dir/lib/actel` and `synplify_install_dir/lib/proasic` directories; the latter is reserved for the 500K and PA families. Use the macro library file that corresponds to your target architecture. If you are targeting the 1200XL architecture, use the `act2.v` macro library.

Using ACTgen Macros with Verilog Designs

To use ACTgen macros with Verilog designs, do the following:

1. In ACTgen, generate the function you want to include.

2. Use the Actel netlist translation utility to convert the resulting EDIF netlist to Verilog.

3. Include the appropriate Actel macro library file for your target architecture in your the source files list for your project.

   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.

4. Include the Verilog version of the ACTgen result in your source files list.

Actel Macro Libraries (VHDL)

The Actel macro libraries supplied with the Synplify synthesis tool contain pre-defined black boxes for the Actel macros, so that you can manually instantiate them into your design. Simply add the appropriate `library` and `use` clauses to the top of the files that instantiate the macros.

Syntax

```
library family;
use family.components.all;
```
Using ACTgen Macros with VHDL Designs

To use ACTgen macros with VHDL designs:

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.

3. Edit the ACTgen converted VHDL file, and add the appropriate library clause at the top of the file:

   ```vhdl
   library family;
   use family.components.all
   ```

4. Include the VHDL version of the ACTgen result in your source files list.
Appendix A: Designing with Actel

Actel-specific Tcl Command Options

This section describes the Actel-specific set_option Tcl command options. These are the options you set for synthesis such as the target technology, device architecture, and synthesis styles.

set_option Command for Actel

The set_option command lets you specify the same device mapping options as the dialog box offers (Project -> Implementation Options).

This table provides information on specific options for Actel architectures. For a complete list of options for this command, refer to set_option on page 4-15.

Table A-1: Actel set_option command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Sets the target technology for the implementation. Keyword must be one of the following Actel architecture names: 3200DX, 40MX, 42MX, 54SX, 54SXA, 54SXS, ACT1, ACT2, ACT3, EX, 500K, PA For the 1200XL architecture, use ACT2</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Check the Device tab (Project -&gt; Implementation Options) for available part choices.</td>
</tr>
<tr>
<td>-speed_grade value</td>
<td>Sets the speed grade for the implementation. Check the Device tab (Project -&gt; Implementation Options) for available speed grade choices. This option is not supported by the Actel 500K and PA architectures.</td>
</tr>
<tr>
<td>-fanout_guide value</td>
<td>Sets the fanout limit guideline for the current project. If you want to set a hard limit, you must also set the -maxfan_hard option to true. For more information about fanout limits, see Fanout Limits on page A-3.</td>
</tr>
</tbody>
</table>
### Table A-1: Actel set_option command (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-maxfan_hard 1</code></td>
<td>Specifies that the specified <code>-fanout_guide</code> value is a hard fanout limit that the Synplify synthesis tool must not exceed. To set a guideline limit, see the <code>-fanout_guide</code> option. For more information about fanout limits, see <a href="#">Fanout Limits on page A-3</a>.</td>
</tr>
<tr>
<td>`-disable_io_insertion 1</td>
<td>0</td>
</tr>
<tr>
<td><code>-report_path value</code></td>
<td>Sets the maximum number of critical paths in a forward-annotated SDF constraint file. This option applies only to the Actel 500K and PA architectures. For information about setting critical paths, see <a href="#">Maximum Number of Critical Paths on page A-5</a>.</td>
</tr>
</tbody>
</table>

**Note:** You cannot specify a package `-package` option in the Synplify synthesis tool environment. You must use Actel Designer for this.
# Actel Attribute and Directive Summary

The following table summarizes the synthesis and Actel-specific attributes and directives available with the Actel technology. Complete descriptions and examples are in Chapter 7, *Synthesis Attributes and Directives*.

Table A-2: Synthesis and Actel-specific attributes and directives

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alsloc</td>
<td>Forward-annotates the relative placements of macros and IP blocks to Actel Designer. This attribute does not apply to Actel 500K and PA designs.</td>
</tr>
<tr>
<td>alsspin</td>
<td>Assigns scalar or bus ports to Actel I/O pin numbers. This attribute does not apply to Actel 500K and PA designs.</td>
</tr>
<tr>
<td>alspreserve</td>
<td>Specifies that a net be preserved, and prevents it from being removed during place-and-route optimization. This attribute does not apply to Actel 500K and PA designs.</td>
</tr>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>parallel_case (D)</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>syn_black_box (D)</td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_enum_encoding (D)</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
Table A-2: Synthesis and Actel-specific attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_isclock (D)</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td>syn_keep (D)</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>Sets a fanout guideline for an individual input port or register output.</td>
</tr>
<tr>
<td>syn_netlist_hierarchy</td>
<td>Determines whether the EDIF output netlist is flat or hierarchical.</td>
</tr>
<tr>
<td>syn_noarrayports</td>
<td>Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Turns off the automatic insertion of clock buffers.</td>
</tr>
<tr>
<td>syn_noprune (D)</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td>syn_preserve (D)</td>
<td>Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.</td>
</tr>
<tr>
<td>syn_preserve_sr_priority</td>
<td>Forces set/reset flip-flops to honor the coded priority for the set or reset.</td>
</tr>
<tr>
<td>syn_radhardlevel</td>
<td>Specifies the radiation-resistant design technique to apply to a module, architecture, or register.</td>
</tr>
<tr>
<td>syn_sharing (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
<tr>
<td>syn_state_machine (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td>syn_tco&lt;n&gt; (D)</td>
<td>Defines timing clock to output delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_tpd&lt;n&gt; (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
### Table A-2: Synthesis and Actel-specific attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_tristate (D)</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td>syn_tsu&lt;n&gt; (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The ( n ) indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_useenables</td>
<td>Prevents generation of registers with clock enable pins for 54SX designs.</td>
</tr>
<tr>
<td>translate_off/translate_on (D)</td>
<td>Specifies sections of code to exclude from synthesis, such as simulation-specific code.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
APPENDIX B

Designing with Altera

The following topics describe how to best design and synthesize with the Altera technology:

- Altera Device-specific Support on page B-2
- General Guidelines on page B-2
- Maximizing Results with APEX, APEX II, Mercury, Excalibur on page B-14
- Maximizing Results with Stratix on page B-20
- Maximizing Results with FLEX and ACEX on page B-26
- Maximizing Results with MAX on page B-32
- Altera Attribute and Directive Summary on page B-36
- Verilog Examples on page B-39
- VHDL Examples on page B-44
- VHDL Prepared Components on page B-51
Altera Device-specific Support

The Synplify synthesis tool creates technology-specific netlists for a number of Altera programmable logic devices. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

The synthesis tool outputs FLEX, MAX, and ACEX netlists as input to the MAX+PLUS II place-and-route tool. It generates APEX20K/E/C, APEX II, Excalibur, Stratix and Mercury netlists (.vqm) as input for the Quartus II place-and-route tool. Because the accepted netlists are updated periodically, check the most current version of the synthesis tool.

General Guidelines

These general guidelines will help you achieve the best design and implementation in Altera technologies:

- Specifying Constraints and Attributes to Obtain Quality Results on page B-3
- Instantiating Verilog and VHDL Macros/Black Boxes on page B-3
- Instantiating LPMs in the Source Code on page B-3
- Inferring General-purpose RAMs on page B-4
- Inferring General-purpose ROMs on page B-6
- Place-and-route Tools and EDIF / VQM Filenames on page B-9
- Inferring Registers with the Compiler on page B-9
- Mapping to ATOM Primitives on page B-10
- Selecting Implementation Options on page B-10
- Viewing Synthesis Reports on page B-11
Specifying Constraints and Attributes to Obtain Quality Results

Specify timing constraints, general HDL attributes, and Altera-specific attributes to improve your design. Manage these files with the SCOPE constraints and attributes editor.


Instantiating Verilog and VHDL Macros/Black Boxes

Select from Altera’s library of predefined macros—for example, soft, global, carry, and cascade—and instantiate them as black boxes. Instantiate global buffers for clocks, resets, sets, and other heavily loaded signals. The number of global buffers available varies with the part.

For Verilog designs, make sure the macro library file is the first file in your list of source files. The path name is:

```
synplify_install_dir/lib/altera/altera.v
```

For VHDL designs, instantiate Altera VHDL macro library elements as predefined black boxes into your design. Add the library and use clauses to the top of the source files in which you instantiate the macros, using the syntax below:

```
library altera ;
use altera.maxplus2.all ;
```

See Verilog Examples on page B-39 and VHDL Examples on page B-44.

Refer to your Altera documentation for supported macros.

Instantiating LPMs in the Source Code

Altera’s Library of Parameterized Modules (LPMs) contains technology-independent logic functions parameterized for scalability and adaptability. The library is derived from LPM version 2.2.0, which offers architecture-independent design entry for all MAX+PLUS II- and Quartus II-supported devices.
**Appendix B: Designing with Altera**

### General Guidelines

#### Instantiating LPMs in Verilog

In Verilog, define LPMs as black boxes, specify attributes, and then instantiate them into a higher-level module. See *Instantiating LPMs* on page B-41 and *Prepared-components Method* on page B-48.

#### Instantiating LPMs in VHDL

In VHDL, instantiate LPMs as either black boxes or as prepared components (predefined, generic LPMs). See *VHDL Examples* on page B-44, and *VHDL Prepared Components* on page B-51 for examples.

See Altera documentation for supported LPMs.

**Black-box Method**

Instantiate a black-box component from the LPM; then assign LPM-specific attributes. This method works for all Altera LPMs. However, this method requires more coding than the prepared-components method.

See *LPM_RAM_DQ Defined as a Black Box* on page B-42, and *LPM_RAM_DQ Instantiated in a Higher-level Module* on page B-42 for examples.

For further details on attributes, see *VHDL Attribute and Directive Syntax* on page 7-16.

**Prepared-components Method**

Consult the list of prepared components, before choosing the black-box method. Prepared components are available as component declarations in LPM_COMPONENTS. See *VHDL Prepared Components* on page B-51.

Specify the appropriate library and use clauses. Then instantiate the prepared components and map the ports and values. See *Prepared-components Method* on page B-48 for details.

#### Inferring General-purpose RAMs

From the HDL source code, the Synplify synthesis tool automatically infers synchronous RAMs and generates general-purpose, single- or dual-port RAMS. This section describes RAM inference for all Altera families except Stratix; for Stratix, see *RAM Inference* on page B-22.
Choosing a RAM Implementation

Inferred RAMs are mapped to LPM RAM primitives. In FLEX architectures, they are implemented in EAB blocks. For APEX20K/E/C, Mercury, ACEX, APEX II, and Excalibur architectures, they are implemented in ESBs.

If the RAM size or address width meets the minimum specifications shown below, the RAM is automatically implemented in EAB/ESB. If the specifications are not met, the RAM is implemented in registers.

Table B-1: Minimum specifications for implementing inferred RAM (Altera)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Minimum RAM Size for Inference (bits)</th>
<th>Address Width (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEX10K/FLEX10KE/ACEX</td>
<td>128</td>
<td>5</td>
</tr>
<tr>
<td>APEX20K/E/C, APEX II, Excalibur, and Mercury</td>
<td>64</td>
<td>4</td>
</tr>
</tbody>
</table>

If ESB/EAB resources are limited, you should specify that individual RAMs be inferred as registers and logic. Use the `syn_ramstyle` attribute (Verilog or VHDL) with a value of `registers` to designate these instances.

Example formats for specifying RAMs to be inferred as logic include the following.

.sdc File Example

```text
define_attribute {ram_dout[7:0]} syn_ramstyle {registers};
```

Verilog Example

```verilog
reg[127:0] ram_dout[7:0] /* synthesis syn_ramstyle="registers" */;
```

VHDL Example

```vhdl
type ram_memtype is array (127 downto 0) of std_logic_vector (7 downto 0);
signal ram_dout : ram_memtype;
attribute syn_ramstyle of ram_dout : signal is "registers";
```

See *Altera Attribute and Directive Summary on page B-36.*
Inferring General-purpose ROMs

From your HDL source code, the Synplify synthesis tool automatically infers ROMs and maps them to ESBs/EABs. This is described below.

You can, however, use the `syn_romstyle` attribute (Verilog or VHDL) with a value of `logic` to designate individual ROM instances that are to be mapped to logic instead of ESBs/EABs. See `syn_romstyle (Altera)` on page 7-97.

**FLEX, ACEX, Mercury and APEX II**

For FLEX10K/FLEX10KE, ACEX, APEX20K/E/C, Mercury and APEX II architectures, the Synplify synthesis tool maps inferred ROM to `lpm_rom` megafunctions; Quartus II then implements the `lpm_rom` in ESBs/EABs. Only asynchronous ROM are mapped for these architectures.

If the ROM size or address width meets the minimum specifications shown below, the ROM is automatically implemented in EAB/ESB. If the specifications are not met, the ROM is implemented in registers.

Table B-2: Minimum specifications for implementing inferred ROM(Altera)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Size in Bits</th>
<th>Address Width in Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEX10K/FLEX10KE/ACEX</td>
<td>128</td>
<td>7</td>
</tr>
<tr>
<td>Mercury, APEX II</td>
<td>64</td>
<td>4</td>
</tr>
</tbody>
</table>

**APEX20K/E/C, Stratix and Excalibur**

For APEX20K/E/C, Stratix and Excalibur architectures, the synthesis tool maps inferred ROM directly to device RAM primitives whenever the ROM function is complex enough:

- `apex20k_ram_slice` for APEX20K
- `apex20ke_ram_slice` for APEX20KE and Excalibur
- `apex20kc_ram_slice` for APEX20KC
- `stratix_ram_block` for Stratix

Otherwise, it maps the ROM to logic. For APEX20K/E/C and Excalibur, both synchronous and asynchronous ROM are mapped; for Stratix, only synchronous ROM are mapped.
Mapping Synchronous ROM in APEX20K/E/C
You can register either the read address or the data output, or both, in order to make the ROM synchronous. The register itself can optionally have an asynchronous clear (aclr) and/or clock enable (clken) control signal.

Mapping Synchronous ROM in Stratix
You must register the read address, in order to make the ROM synchronous; you may also register the data output. The read address register cannot have a clear control signal; it can optionally have a clock enable signal. The data output register can optionally have an asynchronous clear (aclr) and/or clock enable (clken) control signal.

Examples: Usage of syn_romstyle
See syn_romstyle (Altera) on page 7-97, for more information on this directive.

.sdc File Example
The inferred ROM here would normally be mapped to ESBs/EABs. Explicitly specifying logic as the value of syn_romstyle forces the ROM to be mapped to logic.

    define_attribute {rom_dout[7:0]} syn_romstyle {logic};

Verilog Example
The inferred ROM here would normally be mapped to logic. Explicitly specifying block_rom forces the ROM to be mapped to ESBs/EABs.

    // Map the inferred ROM to ESBs/EABs, not to logic.
    reg [7:0] rom_dout [1:0] /* synthesis syn_romstyle="block_rom" */;

VHDL Example

    type rom_memtype is array (127 downto 0) of std_logic_vector (7 downto 0);
    signal rom_dout : rom_memtype;

    -- Map the inferred ROM to logic, not to ESBs/EABs.
    attribute syn_romstyle : string;
    attribute syn_romstyle of rom_dout : signal is "logic";
Automatic Resource Management for ESBs/EABs

The Synplify synthesis tool automatically infers both RAM and ROM, mapping them to ESBs/EABs. As the available ESBs/EABs are limited, the tool maps inferred RAM and ROM to them in the following order: design entities that are explicitly attributed to ESBs/EABs are mapped first, then unattributed inferred RAM is mapped, followed by unattributed ROM.

Explicit attribution to ESBs/EABs includes all of the following. These are always mapped to ESBs/EABs.

- Black boxes are mapped to ESBs/EABs. You determine the number of ESBs/EABs to be used for a black box by assigning it the syn_resources attribute (in addition to syn_black_box). For example, syn_resources = "block_ram=2" tells the synthesis tool that two ESBs/EABs are to be used for this particular black box.

- RTL-based design entities that are assigned the attribute altera_implement_in_esb or altera_implement_in_eab are mapped to ESBs or EABs, respectively. The number of ESBs/EABs used is determined automatically by the synthesis tool.

- Inferred RAMs with a syn_ramstyle attribute value of block_ram are mapped to ESBs/EABs. The number of ESBs/EABs used is determined automatically by the synthesis tool.

- Inferred ROMs with a syn_romstyle attribute value of block_rom are mapped to ESBs/EABs. The number of ESBs/EABs used is determined automatically by the synthesis tool.

After mapping design entities that are explicitly attributed to ESBs/EABs, inferred RAM, are mapped to any remaining ESBs/EABs, followed by inferred ROM:

1. Inferred RAMs without any syn_ramstyle attribute are mapped to remaining ESBs/EABs, provided the inferred RAMs meet the minimum conditions noted in the above table (Minimum specifications for implementing inferred RAM (Altera) on page B-5).

2. Inferred ROM without any syn_romstyle attribute are mapped to remaining ESBs/EABs, provided the inferred ROMs meet the minimum conditions noted in the above table (Minimum specifications for implementing inferred ROM(Altera) on page B-6).
Within each of these categories of mapping, larger inferred RAM or ROM are mapped before smaller ones.

If, at any time during this mapping process, the remaining ESB/EAB count is insufficient for further mapping of inferred RAMs/ROMs, the synthesis tool will try to use registers or logic to implement them, instead.

## Place-and-route Tools and EDIF / VQM Filenames

Altera requires the name (sans extension) of an EDIF or VQM file to be identical to the top-level design name inside the file. For this reason, be sure to give your synthesis result file the same name as your top-level Verilog module or VHDL entity.

Because the synthesis result filename defaults to your HDL source filename (but with the “edf” extension: .edf), give the source file and the top-level module/entity the same name (sans extension). If you fail to give identical names, an Altera error message such as the following will appear:

```
"Can't find top level cell <filename>"
```

To keep track of the files generated by MAX+PLUS II or Quartus II, it is a good idea to create a subdirectory for the EDIF or VQM result files before executing the place-and-route tool.

## Outputting Port Names for the Place-and-route Tools

MAX+PLUS II cannot handle port names that are more than 30 characters in length, but Quartus II can. When you generate an output synthesis file for MAX+PLUS II, check that the syn_edif_name_length attribute is set to restricted, so that port names are generated correctly.

## Inferring Registers with the Compiler

The compiler can infer additional register primitive types so that the mapper can implement a better solution, when it targets Altera technologies:

- Registers with clock enables—The compiler identifies and extracts clock-enable logic for registers. The RTL view displays registers with
enables as special primitives with an extra pin for the enable signal. The special primitives are DFFE, DFFRE, DFFSE, DFFRSE, DFFPATRE and DFFPATRSE. Set syn_direct_enable in the HDL source file (not in the SCOPE spreadsheet or a constraint file) to direct the compiler to infer desirable clock enables. For further syntax details see syn_direct_enable on page 7-53. By default, registers without clock enables will be inferred.

- Registers with synchronous sets/resets—the compiler identifies and extracts registers with synchronous sets and resets, and passes them to the mapper. The special primitives are SDFFR, SDFFS, SDFFRS, SDFFPAT, SDFFRE, SDFFSE and SDFFPATE. The compiler does this automatically and does not require a directive.

Mapping to ATOM Primitives

Because the Synplify synthesis tool maps technology components directly to Quartus II ATOM primitives (for example, logic elements, memory elements, and I/O cells), the total runtime for placement and routing is reduced. Using the Verilog netlist as input, Quartus II builds the logic database, skips synthesis, and performs placement and routing. Thus, the synthesis tool is able to more precisely control the logic implementation in APEX20K/E/C, Stratix, Excalibur, APEX II, and Mercury devices, and more accurately estimate timing.

Selecting Implementation Options

To achieve optimal design results, set the correct implementation options. These options control the following: target technology and related parameters, mapping options to use during synthesis, state machine optimization, output and output formats, results directories, and so forth. Use one of the following methods to set implementation options:

- Select Project -> Implementation Options in the Project view.
- Click the Target button.
Viewing Synthesis Reports

Synplicity tools generate resource usage and timing reports. For the MAX family, only a resource usage report is generated. To view these reports, click the View Log button in the Project view or select View -> View Log File from the menu.

Resource Usage Reports

Resource usage reports for Altera include:

- Cell usage
  - carry chains
  - flip-flops
  - logic element atoms (APEX20K/E/C, APEX II, Mercury, Excalibur, Stratix)
  - LUTs
  - DSP blocks (Stratix)

- Memory cell usage
  - ESB (APEX20K/E/C, APEX II, Mercury, Excalibur)
  - MegaRAMs (Stratix)
  - M4Ks (Stratix)
  - M512s (Stratix)

- Number of I/Os

- Percent of utilization
Example: Altera Apex20K Resource Usage Report

I/O ATOMs: 42
Total LUTs: 247 of 4160 (5%)
Logic resources: 258 ATOMs of 4160 (6%)
ATOM count by mode:
  normal: 242
  arithmetic: 11
  counter: 4
  qfbk_counter: 1
ESBs: 0 (0% of 26)
ATOMs using regout pin: 72
  also using enable pin: 56
  also using combout pin: 0
  with no input combinational logic: 11 (uses cell for routing)
ATOMs using combout pin: 141
ATOMs using cascin pin: 50

Number of Inputs on ATOMs: 1056
Number of Nets: 470

Example: Altera Stratix Resource Usage Report

I/O ATOMs: 42
Total LUTs: 275 of 10570 (2%)
Logic resources: 276 ATOMs of 10570 (2%)
ATOM count by mode:
  normal: 263
  arithmetic: 13
DSP Blocks: 0 (0 nine-bit DSP elements).
MegaRAMs: 0 (0% of 1)
M4Ks: 0 (0% of 60)
M512s: 0 (0% of 94)
ATOMs using regout pin: 72
  also using enable pin: 56
  also using combout pin: 0
  with no input combinational logic: 1 (uses cell for routing)
ATOMs using combout pin: 203
ATOMs using cascin pin: 0

Number of Inputs on ATOMs: 1144
Number of Nets: 458

Example: Altera FLEX10K Resource Usage Report

Total LUTs: 1314 of 576 (228%)
Logic resources: 1314 LCs of 576 (228%)
Number of Nets: 3176
Number of Inputs: 8438
Register bits: 712 (364 using enable)
EABs: 0 (0% of 3)
I/O cells: 268

Details:
Cells in logic mode: 656
Cells in arith mode: 12
Cells in cascade mode: 148
Cells in counter mode: 23
DFFs with no input combinational logic: 476 (uses cell for routing)
LUTs driving both DFF and logic: 54

Crossprobing with Quartus II

You can do bidirectional crossprobing between Quartus II Floorplanner and the Synplify synthesis tool’s HDL Analyst. From the synthesis tool, open a schematic view and enable external crossprobing from the HDL Analyst menu. Open the project in Quartus II and enable crossprobing (Tools -> Options -> EDA Tool Options).
Appendix B: *Designing with Altera*

Maximizing Results with APEX, APEX II, Mercury, Excalibur

This section discusses the following techniques that will help you maximize your results with APEX20K/E/C, APEX II, Excalibur, and Mercury architectures:

- **Device Mapping Options (APEX, APEX II, Mercury, Excalibur)** on page B-14
- **Setting File Format Options (APEX, APEX II, Mercury, Excalibur)** on page B-16
- **Setting Fanout Limits** on page B-17
- **Packing Registers into I/O Cells** on page B-17
- **Forward-annotating Synthesis Files** on page B-17
- **Performing Post-synthesis Simulation** on page B-18
- **Running Quartus II from Within the Synthesis Tool** on page B-18

**Device Mapping Options (APEX, APEX II, Mercury, Excalibur)**

Use one of the following methods to set device mapping options, such as target technology, device architecture, and synthesis styles:
Using the Menu

Select Project -> Implementation Options from the menu. Select the Device tab and specify options as follows:

**Table B-3: APEX20K/E/C, APEX II, Excalibur, and Mercury, menu options**

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Logic to Atoms</td>
<td>Maps logic directly to Altera ATOM primitives. See Mapping to ATOM Primitives on page B-10 for more information. By default, this is enabled.</td>
</tr>
<tr>
<td>Disable I/O Insertion</td>
<td>Enables or disables I/O insertion during synthesis. The default is false, so I/O ATOMs are inserted.</td>
</tr>
<tr>
<td>PerformCliquing</td>
<td>Groups critical path logic functions in floating regions and forward-annotates them to Quartus II. Does not apply to Mercury devices.</td>
</tr>
</tbody>
</table>

**set_option Command for APEX, APEX II, Mercury, Excalibur**

The `set_option` Tcl command offers an alternative way to set implementation options like target technology, device architecture, and synthesis styles. Options set with this command can be saved as a file and re-executed.

The following table lists the arguments specific to the Altera APEX20K/E/C, APEX II, Excalibur, and Mercury families. For a complete list of all available `set_option` options, see `set_option` on page 4-15.

**Table B-4: APEX, APEX II, Excalibur, Mercury set_option command arguments**

<table>
<thead>
<tr>
<th><code>set_option</code> Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-technology keyword</code></td>
<td>Specifies the target technology. The keyword can be APEXII, EXCALIBUR_ARM, MERCURY, APEX20K, APEX20KC, or APEX20KE.</td>
</tr>
<tr>
<td><code>-part part_name</code></td>
<td>Specifies a part for the implementation. Refer to the databook for choices.</td>
</tr>
<tr>
<td><code>-speed_grade value</code></td>
<td>Sets the speed grade. Refer to the databook for choices.</td>
</tr>
<tr>
<td><code>-package package_name</code></td>
<td>Specifies the package. Refer to the databook for choices.</td>
</tr>
</tbody>
</table>
Setting File Format Options (APEX, APEX II, Mercury, Excalibur)

Use one of the following methods to set options for result file formats and design filenames:

**Using the Menu**

Select Project -> Implementation Options from the menu. Select the Implementation Results tab and specify options for results files.

**Using the project Tcl Command**

The project Tcl command specifies the result file formats and filenames for your design. The table below describes the command options that are specific to APEX20K/E/C, APEX II, Mercury and Excalibur. For a complete list of the options, see *project on page 4-12*.

---

**Table B-4: APEX, APEX II, Excalibur, Mercury set_option command arguments**

<table>
<thead>
<tr>
<th>set_option Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-block 1</td>
<td>0</td>
</tr>
<tr>
<td>-disable_io_insertion 1</td>
<td>0</td>
</tr>
<tr>
<td>-clique 1</td>
<td>0</td>
</tr>
<tr>
<td>-cliquing 1</td>
<td>0</td>
</tr>
<tr>
<td>-map_logic 1</td>
<td>0</td>
</tr>
</tbody>
</table>
Setting Fanout Limits

When a design’s high-fanout nets are reported by the placement and routing tool as critical nets, use the syn_maxfan attribute to control the fanouts selectively. This generally results in either buffering or replication of the net’s driver. See syn_maxfan on page 7-70 for further details.

Packing Registers into I/O Cells

Use the syn_useioff attribute to pack registers into APEX20K/E/C, Excalibur, APEX II, and Mercury I/O cells, thereby improving input or output path timing. To pack registers globally, set syn_useioff=1 on the top-level module. To pack them locally, set syn_useioff=1 on a specific port. By default, the I/O registers are not packed. See syn_useioff (Altera) on page 7-108 for further details.

Forward-annotating Synthesis Files

Synthesis automatically generates Altera .tcl constraint files for APEX20K/E/C, APEX II, Stratix, Mercury and Excalibur devices. These files contain timing constraints, pin assignments, part information, and cliques. If you launch Quartus II from the Synplify synthesis tool, these files are automatically forward-annotated. If you execute Quartus II from its own environment, request that the files be read.

Table B-5:  APEX, APEX II, Excalibur, Mercury project Tcl command arguments

<table>
<thead>
<tr>
<th>project Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-result_format vqm</td>
<td>Specifies the Verilog synthesis result file format and its keyword vqm. This argument should appear at the end of the Tcl file, before any project run or project save commands.</td>
</tr>
<tr>
<td>-result_file filename</td>
<td>Names the Verilog synthesis result file. This argument should appear at the end of the Tcl file, before any project run or project save commands.</td>
</tr>
</tbody>
</table>
To disable generation of these files, select the Impl Options button in the Project view. Choose the Implementation Results tab and toggle the Write Vendor Constraint File button. Or, use the set_option Tcl command.

The constraint file generated for Quartus II place-and-route tools has a “tcl” file extension (.tcl). The following constraints are forward-annotated to Quartus II in this file:

- `define_clock`
- `define_false_path`
- `define_input_delay`
- `define_multicycle_path`
- `define_output_delay`

In addition to these constraints, the Synplify synthesis tool forward-annotates relationships between different clocks.

**Performing Post-synthesis Simulation**

To perform post-synthesis simulation with the Synplify synthesis tool, use as input the Quartus II ATOM and component simulation library files from the Quartus II EDA simulation library directory (eda/sim_lib). For example, to map a VHDL design to APEX 20KE devices, use `apex_20ke_atoms.vhd` and `apex_20ke_components.vhd` as input files. To map a Verilog design to APEX 20KE devices, use `apex20ke_atoms.v` as input file.

See Altera documentation for further details.

**Running Quartus II from Within the Synthesis Tool**

You can run Quartus II from within the Synplify synthesis tool, synthesizing your design. You do this by choosing one of the following items from the Options -> Quartus submenu: Set Options, Foreground Compile or Background Compile. Each of these builds a Quartus II project based on the device, synthesized Verilog netlist, timing constraints, and pin assignments that are forward-annotated via the design’s .tcl file.

Refer to Altera’s documentation for further information on the Quartus II place-and-route tool.
Table 2-6:  Options -> Quartus submenu items

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Options</td>
<td>Runs the Quartus II user interface. You can set place-and-route options, such as locations and slew rates, before initiating placement and routing from the Quartus II interface.</td>
</tr>
<tr>
<td>Foreground Compile</td>
<td>Runs the Quartus II user interface, launching placement and routing immediately, without giving you a chance to change place-and-route settings.</td>
</tr>
<tr>
<td>Background Compile</td>
<td>Runs Quartus II in the background (batch) immediately. The Quartus II user interface is not opened. You are not given a chance to change place-and-route settings. A log of placement and routing is displayed as the process progresses. When finished, the Quartus II Computation Report File (.csf.rpt) is displayed.</td>
</tr>
</tbody>
</table>
Maximizing Results with Stratix

This section discusses the following techniques that will help you maximize your results with the Stratix architecture:

- Device Mapping Options (Stratix) on page B-20
- Setting File Format Options (Stratix) on page B-21

Device Mapping Options (Stratix)

Use one of the following methods to set device mapping options, such as target technology, device architecture, and synthesis styles:

Using the Menu

Select Project -> Implementation Options from the menu or click the Impl Options or New Impl buttons in the Project view. Select the Device tab and specify options as follows:

Table B-7: Stratix menu options

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Logic to Atoms</td>
<td>Maps logic directly to Altera ATOM primitives. See <em>Mapping to ATOM Primitives</em> on page B-10 for more information. The default is enabled.</td>
</tr>
<tr>
<td>Disable I/O Insertion</td>
<td>Enables or disables I/O insertion during synthesis. The default is false, so I/O ATOMs are inserted.</td>
</tr>
</tbody>
</table>

set_option Command for Stratix

The set_option Tcl command offers an alternative way to set implementation options like target technology, device architecture, and synthesis styles. Options set with this command can be saved as a file and re-executed.

The table below lists the arguments specific to the Stratix family. For a complete list of all available set_option options, see *set_option* on page 4-15. You can also enter the following in a Tcl Script window:

```
help set_option
```
Setting File Format Options (Stratix)

Use one of the following methods to set options for result file formats and design filenames:

**Using the Menu**

Select Project -> Implementation Options from the menu or click the Impl Options button in the Project view. Select the Implementation Results tab and specify options for results files.

**Using the project Tcl Command**

The project Tcl command specifies the result file formats and filenames for your design. The table below describes the command options that are specific to the Stratix family. For a complete list of the options, see `project` on page 4-12. You can also type the following in a Tcl Script window:

```
help project
```
Appendix B: Designing with Altera

Maximizing Results with Stratix

Table B-9: Stratix project Tcl command arguments

<table>
<thead>
<tr>
<th>project Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-result_file filename</td>
<td>Names the Verilog synthesis result file. This argument should appear at the end of the Tcl file, before any project run or project save commands.</td>
</tr>
</tbody>
</table>

RAM Inference

The table below shows how the Synplify synthesis tool infers RAMs for Stratix. In general, the tool recognizes and extracts the altsyncram megafunction. (For information on ROM inference for Stratix, see Inferring General-purpose ROMs on page B-6.)

Some Verilog statements with blocking assignments may not be mapped to block RAM, so Verilog users are advised to avoid blocking statements when modeling RAMs.

Table B-10: RAM inference in Altera Stratix

<table>
<thead>
<tr>
<th>RAMs with...</th>
<th>Mapped to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous read and write</td>
<td>altsyncram. The read operation can be synchronized by registering either the read address or output of the RAM.</td>
</tr>
<tr>
<td>Asynchronous read or write</td>
<td>Logic.</td>
</tr>
<tr>
<td>Only one address bus</td>
<td>altsyncram, in single-port mode. However, because old data cannot be obtained using altsyncram in single-port mode, these RAM cases are mapped to dual-port mode.</td>
</tr>
<tr>
<td>Two address buses (one each for read and write)</td>
<td>altsyncram, in dual-port mode. Glue logic is created if new data has to be obtained in dual-port mode. You can disable the creation of glue logic by setting the syn_ramstyle value to no_rw_check.</td>
</tr>
<tr>
<td>Two address buses, (one for read and write, the other for read only)</td>
<td>altsyncram, in BIDIR mode.</td>
</tr>
</tbody>
</table>
DSP MAC Inference

Altera Stratix devices have dedicated DSP MAC blocks that can be configured as multipliers, multiplier adders or multiplier accumulators. The DSP MAC blocks are specified in a VQM netlist file by means of Altera’s megafunctions `altmult_add`, `altmult_accum` and `lpm_mult`. Structural VHDL and Verilog simulation netlists also contain these functions.

The Synplify synthesis tool automatically recognizes and extracts these functions, then passes them to Quartus II, which maps them directly to the dedicated DSP MAC blocks. The following table summarizes when DSP MAC blocks are inferred.

Table B-11: DSP MAC block inference in Altera Stratix

<table>
<thead>
<tr>
<th>Configuration</th>
<th>The Synplify synthesis tool implements...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiply-only mode</strong></td>
<td></td>
</tr>
<tr>
<td>A standalone signed or unsigned multiplier x = a*b;</td>
<td>The <code>lpm_mult</code> megafunction.</td>
</tr>
<tr>
<td>One of the following:</td>
<td></td>
</tr>
<tr>
<td>36x36-bit multiplier (36-bit inputs, at most)</td>
<td></td>
</tr>
<tr>
<td>18x18-bit multiplier (18-bit inputs, at most)</td>
<td></td>
</tr>
<tr>
<td>9x9-bit multiplier (9-bit inputs, at most)</td>
<td></td>
</tr>
</tbody>
</table>
## Multiply-add mode

2, 3, or 4 multipliers (all signed or all unsigned) feeding an adder

The second and/or fourth multipliers' outputs can be negated.

\[
\begin{align*}
x &= a*b + c*d; \\
x &= a*b - c*d; \\
x &= a*b + c*d + e*f; \\
x &= a*b - c*d + e*f; \\
x &= a*b + c*d - e*f; \\
x &= a*b - c*d - e*f; \\
x &= (a*b + c*d) + (e*f + g*h); \\
x &= (a*b - c*d) + (e*f + g*h); \\
x &= (a*b + c*d) + (e*f - g*h); \\
x &= (a*b - c*d) + (e*f - g*h);
\end{align*}
\]

The altmult_add megafunction.

The multiplier inputs and outputs can be registered. The output of the final adder can also be registered. The registers can have an asynchronous reset and/or enable. A maximum of four clocks (with or without enables) and four asynchronous resets can be used in an altmult_add to control the registers.

The width of each multiplier bus is rounded up to the next highest multiple of 9 bits, and the larger of the two widths is taken as the size of the multiplier. Unused most significant bits (MSBs) are automatically sign extended.

## Multiply-accumulate mode

1 multiplier driving an accumulator

\[
\begin{align*}
x &= x + a*b; \\
x &= x - a*b;
\end{align*}
\]

The altmult_accum megafunction.

The multiplier can be signed or unsigned. The multiplier input and outputs can be registered. The accumulator register can have an asynchronous reset and/or enable.

The accumulator width can be between 4 and 52; the multiplier width must be between 4 and 18.

The width of each multiplier bus is rounded up to the next highest multiple of 9 bits, and the larger of the two widths is taken as the size of the multiplier. Unused most significant bits (MSBs) will automatically be set to zero, sign extended.
DSP MAC inference takes the following attributes and directives into account:

- **syn_useioff, syn_preserve** – Registers with either of these are not packed into MACs.
- **syn_keep** – A syn_keep directive between the adder and the multiplier prevents inference of multipliers altmult_add and altmult_accum, but not plain multipliers.
- **syn_multstyle** (Stratix only) – This attribute is used to choose between lpm implementation and logic. The default implementation uses dedicated multiplier blocks (MAC); you specify logic or lpm_mult for alternative implementations. A multiplier with syn_multstyle attribute value of logic is mapped to logic, not packed into a MAC.
Maximizing Results with FLEX and ACEX

This section discusses manual, fine-tuning techniques and automatic-optimization steps that will help you maximize your results with FLEX and ACEX architectures.

- Device Mapping Options (FLEX and ACEX) on page B-26
- Setting File Format Options (FLEX and ACEX) on page B-27
- Forward-annotating Synthesis Files on page B-28
- Configuring and Executing MAX+PLUS II on page B-29

Device Mapping Options (FLEX and ACEX)

Use one of the two following methods to set device mapping options, such as target technology, device architecture, and synthesis styles:

Using the Menu

Select Project -> Implementation Options from the menu. In the Device tab, specify options as follows:

Table B-12: FLEX and ACEX menu options

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map logic to LCELLs</td>
<td>Maps logic directly to Altera LCELLs, instead of to primitive gates. See Configuring and Executing MAX+PLUS II on page B-29 and Forward-annotating Synthesis Files on page B-28 for more information.</td>
</tr>
<tr>
<td>Perform Cliquing</td>
<td>Groups critical path logic functions together and forward-annotates them to MAX+PLUS II.</td>
</tr>
</tbody>
</table>

set_option Command for FLEX and ACEX

The set_option Tcl command is an alternative way to specify implementation options like target technology, device architecture, and synthesis styles. Commands entered this way can be saved as a file and...
re-executed. The following table lists options that are specific to FLEX and ACEX. For a complete list of set_option command arguments, see set_option on page 4-15.

Table B-13: FLEX and ACEX set_option Tcl command arguments

<table>
<thead>
<tr>
<th>set_option Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Specifies the target technology. The keyword is FLEX10K, FLEX6000, FLEX8000, or ACEX1K.</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Refer to the databook for choices.</td>
</tr>
<tr>
<td>-speed_grade value</td>
<td>Sets the speed grade. Refer to the databook for choices.</td>
</tr>
<tr>
<td>-package package_name</td>
<td>Specifies the package. Refer to the databook for choices.</td>
</tr>
<tr>
<td>-map_logic 1</td>
<td>Enables/disables direct mapping to LCELLs during synthesis. Default is 1 for ON.</td>
</tr>
<tr>
<td>-clique 1</td>
<td>Enables/disables the grouping of logic functions, for example all logic critical to speed. If enabled, the compiler keeps cliqued members together when it fits the project. Default is 1 for ON.</td>
</tr>
<tr>
<td>-block 1</td>
<td>Enables/disables I/O insertion during synthesis. The default is 0 for Insert I/Os.</td>
</tr>
</tbody>
</table>

Setting File Format Options (FLEX and ACEX)

Use one of the two following methods to set options for result file formats and design filenames:

Using the Menu

Select Project -> Implementation Options from the menu. Select the Device tab and specify options for results files:
Using the project Tcl Command

The project Tcl command specifies the result file formats and filenames for your design. The project command arguments in the following table are specific to ACEX and FLEX designs. For a complete list of the arguments, see `project` on page 4-12.

Table B-14: FLEX and ACEX project Tcl command arguments

<table>
<thead>
<tr>
<th>project Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-result_format</code> filename</td>
<td>Names the Verilog synthesis result file. This argument should appear at the end of the Tcl file, before any <code>project run</code> or <code>project save</code> commands.</td>
</tr>
<tr>
<td>`-result_file edif</td>
<td>ahdl`</td>
</tr>
</tbody>
</table>

Forward-annotating Synthesis Files

Synthesis automatically generates Altera .acf constraint files for FLEX and ACEX devices. These files can then be forward-annotated to MAX+PLUS II and Quartus II for use during placement and routing.

To disable generation of these files, select the Impl Options button in the Project view. Choose the Implementation Results tab and toggle the Write Vendor Constraint File button. Or, use the `set_option` Tcl command.

The constraint file generated for Altera place-and-route tools has an “acf” file extension (.acf). The following constraints are forward-annotated to Max+Plus II and Quartus II in this file:

- define_clock
- define_input_delay
- define_output_delay
Configuring and Executing MAX+PLUS II

Altera’s MAX+PLUS II software provides LAB arrays containing Logic Cells (LCELLs), global routing between LABs, and local routing within LABs. LCELLs implement many, simple logic gates. Taken together, these elements control mapping for the FLEX and ACEX architectures.

Although most synthesis tools pass unmapped, random logic gates or higher level modules to MAX+PLUS II, and depend upon the place-and-route software to map them to devices, the Synplify synthesis tool maps technology directly to Altera LCELLs. The Synplify tool inserts LCELLS into its netlist to improve performance and logic use. By default MAX+PLUS II maps the logic and packs it into LCELLs on the device.

Refer to With Mapped LCELLs on page B-29 for further details.

Even if you specify the Synplify tool to map your design, and you have configured MAX+PLUS II properly to work with the resulting netlist, proper packing does not always result—the design may be too big or too slow. If this occurs, turn off mapping.

Refer to Without Mapped LCELLs on page B-30 to learn more about the results when MAX+PLUS II does the mapping.

With Mapped LCELLs

To map logic to LCELLs, follow these steps:

1. Do the following in the Synplify synthesis tool graphic user interface, then run synthesis:
   - Turn on the Map Logic to LCELLs option (Device tab of the Options for implementation dialog box), and synthesize the design.
   - If necessary, set the syn_edif_name_length attribute to restricted. This attribute ensures that port names in the EDIF output file do not exceed 30 characters in length.

2. In MAX+PLUS II, select Assign -> Global Project Logic Synthesis and change Global Project Synthesis Style to WYSIWYG (the default is NORMAL). You may also turn ON the following two options:
   - Automatic I/O Cell Registers—This option moves registers into the I/Os, reducing area and improving I/O performance, but selecting this option might worsen internal clock frequency.
Automatic Register Packing—This option (in FLEX10K) packs unrelated logic and flip-flops into the same cells, reducing area and sometimes delay, but selecting this option might worsen routability.

3. Click the Define Synthesis Style button, then the Advanced Options button. The Advanced Options dialog box displays.

4. Turn OFF the NOT Gate Push-Back option. Click OK three times to exit the dialog boxes.

5. Run the place-and-route software.

6. Review the cell types in the MAX+PLUS II report file. If cell types include LCELL, carry, cascade, dff, and dffe, then MAX+PLUS II honored the mapping instructions, and you are finished. If the report file contains a high percentage of and2, or2, or similar cell types, or if many such cells exist along your critical paths, then MAX+PLUS II did not honor the instructions and may have worsened the results. If such unwanted cell types occur, see the instructions in *Without Mapped LCELLs on page B-30*.

7. If you want faster timing, reset timing options in the synthesis tool interface, re-execute the tool on your design, and re-execute MAX+PLUS II.

**Without Mapped LCELLs**

To obtain either fast performance or small area, when mapping cannot be used, follow the steps below.

1. Use either of the following methods to turn off the LCELLS option:
   - Select Project -> Implementation Options -> Device and click off Map Logic to LCELLs.
   - Use the `set_option` Tcl command.

2. If necessary, set the `syn_edif_name_length` attribute to restricted, and synthesize the design.

   This attribute ensures that port names in the EDIF output file do not exceed 30 characters in length.

3. If you want fast performance, take these steps:
– In MAX+PLUS II, choose Assign -> Global Project Logic Synthesis. In the
dialog box, change Style to FAST (the default is NORMAL).
– Set Automatic I/O Cell Register and Automatic Register Packing to ON.
 Automatic I/O Cell Registers moves registers into the I/Os, saving area
and improving I/O performance, but perhaps worsening internal
clock frequency.
 Automatic Register Packing (in FLEX10K) packs unrelated logic and
flip-flops into the same cells. This saves area and sometimes delay,
but may worsen routability.
– Click OK to exit the form.
– Execute placement and routing.

4. If you want small area, perform these steps:
– In MAX+PLUS II, choose Assign -> Global Project Logic Synthesis. In the
dialog box, change Style to SLOW (the default is NORMAL).
– Set Automatic I/O Cell Register and Automatic Register Packing to ON.
 Automatic I/O Cell Registers moves registers into the I/Os, saving area
and improving I/O performance, but perhaps worsening internal
clock frequency.
 Automatic Register Packing (in FLEX10K) packs unrelated logic and
flip-flops into the same cells. This saves area and sometimes delay,
but may worsen routability.
– Click the Define Synthesis Style button.
– Click the Use Default button in the dialog box. Change the Carry Chain
and Cascade Chain settings from Ignore to Manual.
– Click OK twice to exit the forms.
– Execute placement and routing.
Maximizing Results with MAX

This section discusses manual, fine-tuning techniques and automatic-optimization steps that will help you maximize your results with MAX architectures.

- Device Mapping Options (MAX) on page B-32
- Setting File Format Options (MAX) on page B-33
- Forward-annotating Synthesis Files on page B-34
- Configuring and Executing MAX+PLUS II for MAX Designs on page B-34

Device Mapping Options (MAX)

Use one of the two following methods to set device mapping options, such as target technology, device architecture, and synthesis styles:

Using the Menu

Select Project -> Implementation Options from the menu. Select the Device tab and specify options for MAX as follows:

Table B-15: MAX menu options

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Cell Fanin</td>
<td>Sets the maximum fanin for synthesis.</td>
</tr>
<tr>
<td>Map logic to LCELLs</td>
<td>Maps logic directly to Altera LCELLs, instead of to primitive gates. See Configuring and Executing MAX+PLUS II for MAX Designs on page B-34 for more information.</td>
</tr>
<tr>
<td>Soft LCELLs</td>
<td>Adds soft buffers during synthesis.</td>
</tr>
<tr>
<td>Percent of designs to optimize for timing</td>
<td>Specifies the percentage of nets to optimize during synthesis. See Area/Delay Trade-off for CPLDs on page 1-19.</td>
</tr>
</tbody>
</table>
set_option Command for MAX

The set_option Tcl command offers an alternative way to enter user interface implementation options like target technology, device architecture, and synthesis styles. Commands entered this way can be saved as a file and re-executed. The following table lists the MAX-specific arguments. For a complete list of the arguments, see set_option on page 4-15.

Table B-16: MAX set_option Tcl command arguments

<table>
<thead>
<tr>
<th>set_option Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Specifies the target technology. The keyword can be MAX3000, MAX7000, or MAX9000.</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Refer to the databook for choices.</td>
</tr>
<tr>
<td>-map_logic 1</td>
<td>0</td>
</tr>
<tr>
<td>-area_delay_percent value</td>
<td>Specifies the percentage of nets to optimize during synthesis. Use only when map_logic is true. The default value is 0.</td>
</tr>
<tr>
<td>-fanin_limit value</td>
<td>Sets the maximum fanin during synthesis. Use only when map_logic is true. To improve routability reduce the quantity. The default value is 40.</td>
</tr>
<tr>
<td>-softBuffers true</td>
<td>false</td>
</tr>
</tbody>
</table>

Setting File Format Options (MAX)

Use one of the two following methods to set options for result file formats and design filenames:

Using the Menu

Select Project -> Implementation Options from the menu. Select the Implementation Results tab and specify options for results files.
Using the project Tcl Command

The project Tcl command specifies the result file formats and filenames for your design. The following table lists the MAX-specific arguments. See project on page 4-12 for a complete list of command arguments.

<table>
<thead>
<tr>
<th>project Argument</th>
<th>Purpose/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-result_file edif</td>
<td>Specifies the format of the Verilog synthesis result file. Enter either edif or ahdl. This argument should appear at the end of the Tcl file, before any project run or project save commands.</td>
</tr>
</tbody>
</table>

Forward-annotating Synthesis Files

Synthesis automatically generates Alteran .acf constraint files for MAX devices. These files can then be forward-annotated to MAX+PLUS II. To disable generation of these files, select the Impl Options button in Project view. Choose the Implementation Results tab and toggle the Write Vendor Constraint File button. Or, use the set_option Tcl command.

The following constraints are forward-annotated to MAX+PLUS II in the .acf constraint file for MAX devices:

- define_clock
- define_input_delay
- define_output_delay

Configuring and Executing MAX+PLUS II for MAX Designs

In the Synplify synthesis tool, make sure the port names in your output file are the right length for MAX+PLUS II. Then select a synthesis style that matches your objectives and configure MAX+PLUS II to access the LMF. See Selecting Synthesis Style on page B-35 and Accessing the LMF on page B-35 for further configuration details.
Setting Length of Port Names

Make sure that the `syn_edif_name_length` attribute is set to restricted, so that the port names in the EDIF output file do not exceed 30 characters.

Selecting Synthesis Style

Select a synthesis style as follows:

1. From Altera MAX+PLUS II open the Assign -> Global Project Logic Synthesis dialog box.

2. Set the Global Project Synthesis Style option to either FAST or NORMAL.

   If you select FAST for a congested part, MAX+PLUS II may be unable to place-and-route your design, because global routing connections to LABs are overloaded with LCELLs. If this congestion occurs, let synthesis automatically map the logic into LCELLs.

Accessing the LMF

MAX+PLUS II uses the LMF to interpret primitives in EDIF netlists. After choosing the compiler, configure it as follows:

1. From the Interfaces menu, select EDIF Netlist Reader Settings.

2. Click the CUSTOMIZE button. A text box will display.

3. Enter the full path to the LMF:

   `synplify_install_dir/lib/synplcty.lmf`

4. Click the small box to the left of the path name.

5. Make sure the power and ground signal names are set to default values of VCC and GND.
Appendix B: *Designing with Altera*  Altera Attribute and Directive Summary

The table below summarizes attributes available with Altera technology.

Table B-18: Attributes and directives for Altera technology

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altera_auto_use_eab</td>
<td>Automatically implements large RAMs and ROMs in embedded array blocks (EABs). (FLEX, ACEX)</td>
</tr>
<tr>
<td>altera_auto_use_esb</td>
<td>Automatically implements large RAMs and ROMs in embedded system blocks (ESBs). (Excalibur, APEX II, Mercury, APEX20K/E/C, Stratix)</td>
</tr>
<tr>
<td>altera_chip_pin_lc</td>
<td>Specifies pin locations for Altera I/Os and forward-annotations information to the place-and-route tool.</td>
</tr>
<tr>
<td>altera_implement_in_eab</td>
<td>Implements a logic design unit as embedded array blocks (EABs). (FLEX, ACEX)</td>
</tr>
<tr>
<td>altera_implement_in_esb</td>
<td>Implements a logic design unit as a PTERM in an embedded system block (ESB). (APEX20K/E/C, APEX II, Excalibur, Mercury, Stratix)</td>
</tr>
<tr>
<td>altera_io_opendrain</td>
<td>Sets open-drain mode for output and bidirectional ports. (APEX20KE, APEX II, Mercury)</td>
</tr>
<tr>
<td>altera_io_powerup</td>
<td>Controls the power-up mode for I/O registers that do not have predefined preset or clear conditions. (APEX20KE, APEX II, Mercury)</td>
</tr>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
</tbody>
</table>

(D) indicates directive; (D) and (A) indicates both directive and attribute. All others are attributes.
<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parallel_case (D)</code></td>
<td>Specifies a parallel, multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td><code>syn_black_box (D)</code></td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td><code>syn_direct_enable (D and A)</code></td>
<td>Identifies which signal is to be used as the enable input to an enable flip-flop when multiple candidates are possible.</td>
</tr>
<tr>
<td><code>syn_edif_name_length</code></td>
<td>Determines the length of port names in an Altera output EDIF file.</td>
</tr>
<tr>
<td><code>syn_encoding</code></td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td><code>syn_enum_encoding (D)</code></td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
<tr>
<td><code>syn_forward_io_constraints</code></td>
<td>Enables I/O constraints to be forward-annotated to the place-and-route tool.</td>
</tr>
<tr>
<td><code>syn_hier</code></td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td><code>syn_isclock (D)</code></td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td><code>syn_keep (D)</code></td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td><code>syn_maxfan</code></td>
<td>Sets a fanout limit for an individual port or register output.</td>
</tr>
<tr>
<td><code>syn_multstyle</code></td>
<td>(Stratix only) Determines whether multipliers are implemented in logic or hardware blocks.</td>
</tr>
<tr>
<td><code>syn_netlist_hierarchy</code></td>
<td>Determines whether the EDIF output netlist is flat or hierarchical.</td>
</tr>
<tr>
<td><code>syn_noarrayports</code></td>
<td>Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.</td>
</tr>
</tbody>
</table>

(D) indicates directive; (D) and (A) indicates both directive and attribute. All others are attributes.
### Table B-18: Attributes and directives for Altera technology (Continued)

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>syn_noprune</code> (D)</td>
<td>Controls the automatic removal of instances with outputs that are not driven.</td>
</tr>
<tr>
<td><code>syn_preserve</code> (D)</td>
<td>Enables or prevents sequential optimizations, which eliminate redundant registers and registers with constant drivers.</td>
</tr>
<tr>
<td><code>syn_ramstyle</code></td>
<td>Determines how RAMs are implemented.</td>
</tr>
<tr>
<td><code>syn_reference_clock</code></td>
<td>Specifies a clock frequency other than that implied by the signal on the clock pin of the register.</td>
</tr>
<tr>
<td><code>syn_resources</code></td>
<td>Specifies the resources available for black boxes. It is attached to Verilog black-box modules or VHDL architectures or component definitions.</td>
</tr>
<tr>
<td><code>syn_romstyle (Altera)</code></td>
<td>Determines how ROM architectures are implemented.</td>
</tr>
<tr>
<td><code>syn_sharing</code> (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
<tr>
<td><code>syn_state_machine</code> (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td><code>syn_tco&lt;n&gt;</code> (D)</td>
<td>Defines the clock to output delay timing through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tpd&lt;n&gt;</code> (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tristate</code></td>
<td>Specifies that a black box is a tristate pin.</td>
</tr>
<tr>
<td><code>syn_tsu&lt;n&gt;</code> (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>translate_off/translate_on</code> (D)</td>
<td>Specifies sections of code to exclude from synthesis, such as simulation-specific code.</td>
</tr>
</tbody>
</table>

(D) indicates directive; (D) and (A) indicates both directive and attribute. All others are attributes.
Verilog Examples

This section shows the following Verilog design examples:

- Specifying I/O Locations on page B-39
- Mapping Altera EABs on page B-40
- Instantiating LPMs on page B-41
- Instantiating Special Buffers as Black Boxes on page B-43

Specifying I/O Locations

The following examples show I/O locations specified with the attribute `altera_chip_pin_loc` for different technologies. See `altera_chip_pin_lc` on page 7-31 for further details.

Example: FLEX

```verilog
module adder_8(cout, sum, a, b, cin);

/* Put the cout output on pin 159. */
output cout /* synthesis altera_chip_pin_lc="@159" */;
output [7: 0] sum
/* synthesis
altera_chip_pin_lc="@17,@166,@191,@152,@15,@148,@147,@149" */;
```
Appendix B: Designing with Altera

Example: APEX20K/E/C, Excalibur, and Mercury

module alu(out1, opcode, a, b, sel, clk);
output [7:0] out1 /*synthesis altera_chip_pin_lc = */
7:0] a; 14,12,11,5,21,18,16,15 */; input [2:0] opcode;
input [7:0] a,b;
input sel, clk;
reg [7:0] alu_tmp;
reg [7:0] out1;

Mapping Altera EABs

The following example shows use of the altera_implement_in_eab attribute to map logic into EABs for Altera FLEX devices. An integer square root of an input bus of width 2n with a result bus size of n was computed. This complicated function is mapped to a 256x4 block of RAM. See altera_implement_in_eab on page 7-33 for further details.

module sqrtb(z, a);
parameter asize = 8;
output [(asize/2)-1:0] z;
input [asize-1:0] a;
reg [(asize/2)-1:0] z;
always @(a) begin :lbl
integer i;
// r is remainder, tt is delta for adding one bit
// v is current sqrt value
reg [asize-1:0] v, r, tt;
```verilog
v = 0;
r = a;
for (i = asize/2 - 1; i >= 0; i = i - 1) begin
    tt = (v << (i + 1)) | (1 << (i + i));
    if (tt <= r) begin
        v = v | (1 << i);
r = r - tt;
    end
end
z = v;
end
endmodule

The error in an integer sqrt as the difference between the original input and the square of the square root was computed.

```verilog
module sqrtterr(e, a);
    output [7:0] e;
    input [7:0] a;
    wire [3:0] sqa;
    sqrtb sq (.z(sqa), .a(a)) /* synthesis
        altera_implement_in_eab=1 */;
    defparam sq.asize = 8;
    wire [7:0] zz = sqa * sqa;
    assign e = a - zz;
endmodule

**Instantiating LPMs**

In the following examples, the Altera LPM_RAM_DQ is defined as a black box and instantiated at a higher level. In addition, LVDS is instantiated as a black box.

LPM_RAM_DQ is a parameterized RAM with separate input and output ports, for implementing asynchronous memory or memory with synchronous inputs and/or outputs.

See [Instantiating Black Boxes in Verilog](#) on page 8-40 for further details.
LPM_RAM_DQ Defined as a Black Box

Define a black box with the name myram_64x16. The syn_black_box directive and its arguments have been specified between the port list and the semicolon ';'. LPM_TYPE is specified as “LPM_RAM_DQ.” See syn_black_box on page 7-149 for further details.

```verilog
module myram_64x16 (data, address, inclock, outclock, we, q)
/* synthesis syn_black_box
   LPM_WIDTH=16
   LPM_WIDTHAD=6
   LPM_TYPE="LPM_RAM_DQ" */;
input [15:0] data;
input [5:0] address;
inout inclock, outclock;
inout we;
output [15:0] q;
// This is an empty module
endmodule
```

LPM_RAM_DQ Instantiated in a Higher-level Module

```verilog
module myram(clock, we, data, address, q);
input clock, we;
input [15:0] data;
input [5:0] address;
output [15:0] q;
myram_64x16 inst1 (data, address, clock, clock, we, q);
endmodule
```

LVDS Instantiated as a Black Box

An I/O standard used for high-speed I/O interfaces, LVDS (low-voltage differential signaling) uses differential inputs without a reference voltage. It uses two wires carrying differential values, to create a single channel. Two pins are required to determine the state of the signal. The LVDS I/O standard uses a differential signal, an output (VCCIO) voltage of 3.3 V and the input/output buffers are determined by LVDS. The LVDS I/O standard is supported by the following Altera device families: APEX20K/C/E, Excalibur, Mercury, Stratix.

In this example, LVDS is instantiated as a black box.
See `syn_black_box` on page 7-149 for further details.

```verilog
module mylvds_tx (tx_in, tx_inclock, tx_out)
    /* synthesis syn_black_box 
       number_of_channels = 1 
       deserialization_factor = 4 
       inclock_period = 20000 
       lpm_type = "ALTLVDS_TX" */;
    input [3:0] tx_in;
    input tx_inclock;
    output [0:0] tx_out;
endmodule

module top (tx_in, tx_inclock, tx_out);
    input [3:0] tx_in;
    input tx_inclock;
    output [0:0] tx_out;
mylvds_tx u1 (tx_in, tx_inclock, tx_out);
endmodule
```

### Instantiating Special Buffers as Black Boxes

With the Synplify synthesis tool, define a black-box module named `global`. In your Verilog source files, include global buffers for clocks, resets, sets and other heavily loaded signals.

See Altera documentation for more information.

See `syn_black_box` on page 7-149 for further details.

#### Example

```verilog
module global(a_out, a_in) /* synthesis syn_black_box */ ;
    output a_out;
    input a_in;

    /* This continuous assignment is used for simulation, 
    but is ignored by synthesis. */
    assign a_out = a_in;
endmodule
```
Appendix B: Designing with Altera

VHDL Examples

This section shows the following VHDL design examples:

- Specifying I/O Locations on page B-44
- Mapping Altera EABs on page B-40
- Instantiating LPMs in VHDL on page B-46
- Instantiating Special Buffers as Black Boxes on page B-43

Specifying I/O Locations

The following examples show I/O locations specified with the `altera_chip_pin_lc` attribute for different technologies. See `altera_chip_pin_lc` on page 7-31 for further details.

FLEX10K, ACEX, and MAX Example

```
module top(clk, pad_clk);
  output pad_clk;
  input clk;
  // pad clk is the primary input
  global clk_buf(pad_clk, clk);
endmodule
```

```
library ieee, synplify;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adder_8 is
  port ( a, b: in std_logic_vector (7 downto 0); 
              result: out std_logic_vector(7 downto 0) );

  -- Assign the "result" output bus from bits 7 down
  -- through 0 on pins 17, 166, 191, 152, 15, 148, 147,
  -- and 149, respectively
  attribute altera_chip_pin_lc : string;
  attribute altera_chip_pin_lc of result : signal is
    "@17, @166, @191, @152, @15, @148, @147, @149";
```
-- Assign the "a" input bus from bits 7 through 0 on
-- pins 194, 177, 70, 97, 109, 6, 174, and 204.
attribute altera_chip_pin_lc of a : signal is
 "@194, @177, @70, @97, @109, @6, @174, @204";

-- Let the "b" input be placed by Altera
end adder_8;

architecture behave of adder_8 is
begin
 result <= a + b;
end;

Example: APEX20K/E/C, APEX II, Mercury, and Excalibur
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of data0 : signal is "14, 12, 11, 5,
21, 18, 16, 15";

Mapping Altera EABs
This example shows use of the altera_implement_in_eab attribute to map logic
into EABs for Altera FLEX devices. See altera_implement_in_eab on page 7-33
for further details.

library ieee, synplify;
use ieee.std_logic_1164.all;

entity mymux is
port (in1: in std_logic_vector(9 downto 0);
 sel : in std_logic;
 dout : out std_logic_vector(9 downto 0));
end mymux;

architecture behave of mymux is
begin
 dout <= in1 when sel = '1' else
 (NOT in1) when sel = '0' else
 (others => 'X');
end behave;

library ieee;
use ieee.std_logic_1164.all;
entity eab_test is
  port(a: in std_logic_vector(9 downto 0);
    s: in std_logic;
    o: out std_logic_vector(9 downto 0));
end eab_test;
architecture arch1 of eab_test is
component mymux is
  port (in1: in std_logic_vector (9 downto 0);
        in2: in std_logic_vector (9 downto 0);
        sel : in std_logic;
        dout: out std_logic_vector (9 downto 0));
end component mymux;
attribute altera_implement_in_eab : boolean;
attribute altera_implement_in_eab of U1: label is true;
begin
  U1: mymux port map (
    in1 => a,
    sel => s,
    dout => o);
end arch1;

Instantiating LPMs in VHDL

In VHDL two methods exist for instantiating Altera LPMs:

- the black-box method uses coded attributes, and
- the prepared-components method uses selected components you declare from LPM_COMPONENTS.

For further details on attributes, see Chapter 7, Synthesis Attributes and Directives.

For further details on instantiation, see Component Instantiation in VHDL on page 9-24 and Prepared-components Method on page B-48.

Black-box Method

In the following examples, the Altera LPM_RAM_DQ is defined as a black box and instantiated at a higher level. LPM_RAM_DQ is a parameterized RAM with separate input and output ports, for implementing asynchronous memory or memory with synchronous inputs and/or outputs.
LPM_RAM_DQ uses EABs in FLEX devices, and latch arrays in other device families. In FLEX devices, use synchronous rather than asynchronous RAM functions.

See *Instantiating Black Boxes in VHDL* on page 9-86 for further details.

**Example of Instantiating LPM_RAM_DQ**

```vhdl
library synplify;
entity myram is
    port (clock, we: in bit;
    data : in bit_vector (3 downto 0);
    address: in bit_vector (1 downto 0);
    q: out bit_vector (3 downto 0));
end myram;

architecture arch1 of myram is
component myram_4x4
    port (data: in bit_vector (3 downto 0);
    address: in bit_vector (1 downto 0);
    inclock, outclock, we: in bit;
    q: out bit_vector (3 downto 0));
end component;

-- Assign the appropriate attribute values.
attribute syn_black_box : boolean;
attribute syn_black_box of myram_4x4: component is true;
attribute LPM_WIDTH : POSITIVE;
attribute LPM_WIDTH of myram_4x4: component is 4;
attribute LPM_WIDTHAD: POSITIVE;
attribute LPM_WIDTHAD of myram_4x4: component is 2;

-- Specify that the LPM to be used is "LPM_RAM_DQ"
attribute LPM_TYPE: STRING;
attribute LPM_TYPE of myram_4x4: component is "LPM_RAM_DQ";

begin
    -- Instantiate the LPM component
    u1: myram_4x4  port map
        (data, address, clock, clock, we, q);
end arch1;
```
Example of Instantiating LVDS

See `syn_black_box` on page 7-149 for further details.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
--Use Lpm.lpm_components.all;

entity test2 is port (rx_in : std_logic_vector (0 downto 0);
    Rx_inclock : in std_logic;
    Rx_out : out std_logic_vector (3 downto 0));
end test2;

architecture archtest of test2 is
component altlvds_rx_syn is port
    (rx_in : std_logic_vector (0 downto 0);
    Rx_inclock : in std_logic;
    Rx_out : out std_logic_vector (3 downto 0));
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of altlvds_rx_syn : component is true;
attribute number_of_channels : integer;
attribute number_of_channels of altlvds_rx_syn : component is 1;
attribute deserialization_factor : integer;
attribute deserialization_factor of altlvds_rx_syn : component is 4;
attribute inclock_period : integer;
attribute inclock_period of altlvds_rx_syn : component is 20000;
attribute lpm_type : string;
attribute lpm_type of altlvds_rx_syn : component is "ALTLVDS_RX";

begin
I0: altlvds_rx_syn port map
    (rx_in => rx_in,
    Rx_inclock => rx_inclock,
    Rx_out => rx_out);
end archtest;
```

Prepared-components Method

Prepared LPM Components, available as component declarations in `LPM_COMPONENTS`, use generics instead of attributes to specify different design parameters. After you specify `library` and `use` clauses, instantiate the components and assign (map) the ports and the values for the generics. Refer to the MAX+PLUS II or Quartus II documentation for ports and
generics that require mapping. See Component Instantiation in VHDL on page 9-24 and VHDL Prepared Components on page B-51 for available LPM components.

Example of Instantiating a Prepared Component

This example shows the instantiation of the prepared component lpm_ram_dq:

```vhdl
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 Special Buffers as Black Boxes

With the Synplify synthesis tool, you can specify special buffers in your VHDL source files that Altera will accept. To access global buffers for clocks, resets, sets, and heavily loaded signals, create a black-box module named global. Use global to buffer the signals you want assigned to special...
buffers. The quantity of available global buffers varies with the part. Refer to your Altera documentation for more information. See syn_black_box on page 7-149 for further details.

Example of Instantiating Special Buffers as Black Boxes

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;

entity top is
    port (clk : out std_logic;
          pad_clk : in std_logic);
end top;

architecture structural of top is
    -- In this example, "global" is an Altera vendor macro
    -- that is to be directly instantiate in the Altera
    -- VHDL design as a black box.

    component global
        port(a_out : out std_logic; a_in : in std_logic);
    end component;

    -- Set the syn_black_box attribute on global to be "true".
    attribute syn_black_box of global: component is true;

    -- Declare clk, the internal global clock signal
    begin
        -- pad_clk is the primary input
        clk_buf: global port map (clk, pad_clk);
    end structural;
```
VHDL Prepared Components

The following VHDL prepared LPM components are provided by Synplicity. You will find their VHDL source code definitions in this file:

\texttt{synplify\_install\_dir/lib/vhd/lpm.vhd}.

\begin{tabular}{llll}
altcam\_syn & altcdr\_rx & altcdr\_tx & altclklock \\
altclklock\_syn & altddio\_bidir & altddio\_out & altdpram \\
altlvds\_rx & altlvds\_tx & altqram & csfifo \\
dcfifo & lpm\_abs & lpm\_add\_sub & lpm\_and \\
lpm\_bustri & lpm\_clshift & lpm\_compare & lpm\_components \\
lpm\_constant & lpm\_counter & lpm\_decode & lpm\_divide \\
lpm\_ff & lpm\_fifo & lpm\_fifo\_dc & lpm\_inv \\
lpm\_latch & lpm\_mult & lpm\_mux & lpm\_ram\_dp \\
lpm\_ram\_dq & lpm\_ram\_io & lpm\_or & lpm\_rom \\
lpm\_shiftreg & lpm\_xor & scfifo & \\
\end{tabular}
APPENDIX C

Designing with Atmel

This chapter discusses the following topics:

• General Guidelines on page C-2
• Atmel-specific Tcl Command Options on page C-4
• Atmel Attribute and Directive Summary on page C-5
Appendix C: Designing with Atmel

General Guidelines

The following sections discuss Atmel-specific synthesis features.

Supported Atmel Device Families

The Synplify synthesis tool creates technology-specific netlists for a number of Atmel FPGA families. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

These netlists are input to the Figaro place-and-route tool.

Obtaining Quality Results Using Constraint Files

Constraint files are for user-specified timing constraints and vendor-specific attributes and are added to the source files list along with your HDL source files. Constraint files can contain timing constraints and general attributes (Verilog or VHDL). Use the SCOPE constraints and attributes editor to manage your constraint files.

Atmel Device Mapping Options

Device mapping options are selected from the Device tab in the Options for Implementation dialog box (click the Impl Options button in the Project window or select Project -> Implementation Options from the Project view to display the dialog box).

For Atmel 40K FPGAs, the only supported device mapping option is Disabling I/O Insertion.

Disabling I/O Insertion

The Synplify synthesis tool inserts I/O pads into your design. You can manually insert I/O pads in which case the tool will insert I/O pads only for the pins that require them.
To prevent the tool from inserting any I/O pads into your design, click the Disable I/O Insertion check box on the Device panel of the Options for Implementation dialog box.

![Options for Implementation dialog box](image)

Figure C-1: Device panel in Options for Implementation dialog box

Setting this option is useful to see how much area your blocks of logic take up before synthesizing to an entire FPGA.

**Note:** If you disable I/O insertion, you will not get any I/O pads in your design unless you instantiate them.
Atmel-specific Tcl Command Options

This section describes the Atmel-specific `set_option` Tcl command options. These are the options you set for synthesis such as the target technology, device architecture, and synthesis styles.

set_option Command for Atmel

The `set_option` command lets you specify the same device mapping options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on specific options for Atmel architectures. For a complete list of options for this command, see `set_option` on page 4-15.

help set_option

The `set_option` command supports the following options for the Atmel architecture:

Table C-1: Atmel set_option command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-technology keyword</code></td>
<td>Sets the target technology for the implementation. <code>keyword</code> must be AT40K, AT40KAL, or ATFPSLIC.</td>
</tr>
<tr>
<td><code>-part part_name</code></td>
<td>Specifies a part for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available part choices.</td>
</tr>
<tr>
<td><code>-speed_grade value</code></td>
<td>Sets the speed grade for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available speed grades.</td>
</tr>
<tr>
<td>`-block 1</td>
<td>0`</td>
</tr>
</tbody>
</table>
Atmel Attribute and Directive Summary

The following table summarizes the synthesis and Atmel-specific attributes and directives. For complete descriptions and examples of the attributes and directives, see Chapter 7, *Synthesis Attributes and Directives*.

Table C-2: Atmel attributes and directives

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>parallel_case (D)</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>syn_black_box (D)</td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_enum_encoding (D)</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_isclock (D)</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td>syn_keep (D)</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>Sets a fanout limit for an individual input port or register output.</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Turns off the automatic insertion of clock buffers.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
### Table C-2: Atmel attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>syn_noprune</code> (D)</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td><code>syn_preserve</code> (D)</td>
<td>Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.</td>
</tr>
<tr>
<td><code>syn_ramstyle</code></td>
<td>Determines the way in which RAMs are implemented for the 40K family only.</td>
</tr>
<tr>
<td><code>syn_sharing</code> (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
<tr>
<td><code>syn_state_machine</code> (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td><code>syn_tco&lt;n&gt;</code> (D)</td>
<td>Defines timing clock to output delay through the black box. The ( n ) indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_lpd&lt;n&gt;</code> (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The ( n ) indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tristate</code> (D)</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td><code>syn_tsu&lt;n&gt;</code> (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The ( n ) indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>translate_off/translate_on</code> (D)</td>
<td>Specifies sections of code to exclude from synthesis, such as simulation-specific code.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
APPENDIX D

Designing with Cypress

This chapter describes the following topics for Cypress:

- Overview on page D-2
- Cypress-specific Options on page D-11
- Cypress Attribute and Directive Summary on page D-12
Appendix D: Designing with Cypress

Overview

Supported Cypress Device Families

The Synplify synthesis tool creates .vhn technology-specific netlist files for various Cypress CPLD families. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

The synthesis tool outputs netlists ready for Cypress Warp place-and-route tools.

Obtaining Quality Results Using Constraint Files

Constraint files are for user-specified timing constraints and vendor-specific attributes and are added to the source files list along with your HDL source files. Constraint files can contain timing constraints and general attributes. Use the SCOPE spreadsheet to manage your constraint files.

Instantiating LPMs

LPMs (Library of Parameterized Modules) are technology-independent logic functions that are parameterized to achieve scalability and adaptability. Cypress has implemented LPMs, also called parameterized functions, from LPM version 2.2.0 that offer architecture-independent design entry for Ultra37000, Quantum38K and Delta39K devices.

The Synplify synthesis tool supports instantiation of LPMs. It also supports inference of LPMs for the following structures:

- adders
- comparators
- counters
- multipliers
- RAMs
- subtractors
- incrementors
- decrementors
Instantiating LPMs in Verilog

The Synplify synthesis tool supports instantiating LPMs as black boxes in Verilog when you define the LPM parameters as comments. You can find the available descriptions of ports and parameters in the Cypress Warp documentation. When instantiating Cypress LPMs in Verilog, you must add the parameter lpm_type and set it equal to the exact LPM name. For example, lpm_type = "MADD_SUB".

Verilog Example
The following example shows you how to implement a MADD_SUB component in the design of an adder.

```verilog
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
```
Instantiating LPMs in VHDL

In VHDL there are two methods for instantiating Cypress LPMs:

- Black-box method using attributes
- Prepared-component method using generics. For a list of these prepared components, see the Generics Method section of this chapter.

Black-box Method

The black-box method can be used for any LPM supported by Cypress, but requires more coding than the prepared components method. The prepared-component method is simpler and easier to use, so check to see if there is a prepared component available that you can use before choosing the black-box method.

In the black-box method, instantiate a black-box component for the LPM, then assign LPM-specific attributes. This method works for all Cypress LPMs.

VHDL Example

The following shows you how to instantiate an adder as a black-box component.

```vhdl
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;

Generics Method
In VHDL, you can instantiate Cypress LPMs using generics. Selected LPMs have been prepared (written) for you that use generics. These selected LPMs are available as component declarations in a package called lpmpkg. After providing the appropriate library and use clauses, instantiate the components and assign (map) the ports and the values for the generics. Refer to Cypress documentation for which ports and generics are required to be mapped. See List of Prepared LPM Components (VHDL) on page D-7.
VHDL Example
The following example shows you how to implement a MADD_SUB component in the design of an adder.

```vhdl
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;
```
List of Prepared LPM Components (VHDL)

This section provides a list of available Cypress LPM components and related VHDL source code.

madd_sub  
component madd_sub  
generic( lpm_width: positive;  
lpm_representation : repre_type := LPM_UNSIGNED;  
lpm_direction: arith_type := LPM_NO_TYP;  
lpm_type: string := "madd_sub";  
lpm_hint: goal_type := SPEED);  
port ( dataa: in  std_logic_vector(lpm_width-1 downto 0);  
datab : in  std_logic_vector(lpm_width-1 downto 0);  
cin : in  std_logic := '0';  
add_sub: in  std_logic := '1';  
result: out std_logic_vector(lpm_width-1 downto 0);  
cout : out std_logic;  
overflow: out std_logic);  
end component;

mbuf  
component mbuf  
generic( lpm_width: positive;  
lpm_type: string := "mbuf";  
lpm_hint: goal_type := SPEED);  
port ( data: in  std_logic_vector(lpm_width-1 downto 0);  
result: out std_logic_vector(lpm_width-1 downto 0));  
end component;

mclshift  
component mclshift  
generic( lpm_width: positive;  
lpm_widthdist: positive;  
lpm_shifttype: shift_type := LPM_LOGICAL;  
lpm_type: string := "mclshift";  
lpm_hint: goal_type := SPEED);  

madd_sub  
mcompare  
mcounter  
mmult  
mram_dq  
cy_ram_dp  
mrom
Appendix D: Designing with Cypress

Overview

D-8 Synplify Reference Manual, April 2002

```vhdl
port   (data: in  std_logic_vector(lpm_width-1 downto 0);
distance: in  std_logic_vector(lpm_widthdist-1 downto 0);
direction : in  std_logic := '0';
result: out std_logic_vector(lpm_width-1 downto 0);
overflow: out std_logic;
underflow: out std_logic);
end component;

mcompare
component mcompare
generic(lpm_width: positive;
     lpm_representation : repre_type := LPM_UNSIGNED;
     lpm_type: string := "mcompare";
     lpm_hint: goal_type  := SPEED);
port  ( dataa: in  std_logic_vector(lpm_width-1 downto 0);
datab : in  std_logic_vector(lpm_width-1 downto 0);
alb : out std_logic;
aeb : out std_logic;
agb : out std_logic;
ageb : out std_logic;
aleb : out std_logic;
aneb : out std_logic);
end component;

mcounter
component mcounter
generic(lpm_width: positive;
     lpm_direction: ctdir_type := LPM_NO_DIR;
     lpm_avalue: std_logic_vector := "";
     lpm_svalue: std_logic_vector := "";
     lpm_pvalue: std_logic_vector := "";
     lpm_hint: goal_type := SPEED;
     lpm_type: string := "mcounter";
     lpm_modulus: positive := 1);
port   (data: in  std_logic_vector(lpm_width-1 downto 0);
clock : in  std_logic;
clk_en: in  std_logic := '1';
cnt_en: in  std_logic := '1';
updown: in  std_logic := '1';
q : out std_logic_vector(lpm_width-1 downto 0);
aset : in  std_logic := '0';
aclr : in  std_logic := '0';
end component;
```

D-8 Synplify Reference Manual, April 2002
mmult

component mmult
  generic(lpm_widtha: positive;
            lpm_widthb: positive;
            lpm_widths: natural := 0;
            lpm_widthp: positive;
            lpm_representation : repre_type := LPM_UNSIGNED;
            lpm_hint: goal_type := SPEED;
            lpm_type: string := "mmult";
            lpm_avalue: std_logic_vector := ""
  port   (dataa: in  std_logic_vector(lpm_widtha-1 downto 0);
            datab : in  std_logic_vector(lpm_widthb-1 downto 0);
            sum : in  std_logic_vector(lpm_widths-1 downto 0);
            result: out std_logic_vector(lpm_widthp-1 downto 0));
end component;

mram_dq

component mram_dq
  generic(lpm_width: positive;
            lpm_widthad: positive;
            lpm_numwords: natural := 0;
            lpm_indata: regis_type := LPM_REGISTERED;
            lpm_address_control : regis_type := LPM_REGISTERED;
            lpm_outdata: regis_type := LPM_REGISTERED;
            lpm_file: string := "";
            lpm_type: string := "mram_dq";
            lpm_hint: goal_type := SPEED);
  port   (data: in  std_logic_vector(lpm_width-1 downto 0);
            address: in  std_logic_vector(lpm_widthad-1 downto 0);
            q : out std_logic_vector(lpm_width-1 downto 0);
            inclock: in  std_logic := '0';
            outclock: in  std_logic := '0';
            we : in  std_logic;
            outreg_ar: in  std_logic := '0');
end component;
Appendix D: Designing with Cypress

Overview

D-10 Synplify Reference Manual, April 2002

cy_ram_dp

component cy_ram_dp
  generic(lpm_width: positive;
    lpm_widthad: positive;
    lpm_numwords: natural := 0;
    lpm_indata: regis_type := LPM_REGISTERED;
    lpm_address_control : regis_type := LPM_REGISTERED;
    lpm_outdata_a: regis_type := LPM_REGISTERED;
    lpm_outdata_b: regis_type := LPM_REGISTERED;
    lpm_file: string := "";
    lpm_type: string := "cy_ram_dp";
    lpm_hint: goal_type := SPEED);
  port   (data_a: in  std_logic_vector(lpm_width-1   downto 0);
          data_b: in  std_logic_vector(lpm_width-1   downto 0);
          address_a: in  std_logic_vector(lpm_widthad-1 downto 0);
          address_b: in  std_logic_vector(lpm_widthad-1 downto 0);
          q_a : out std_logic_vector(lpm_width-1   downto 0);
          q_b : out std_logic_vector(lpm_width-1   downto 0);
          addr_matchb: out std_logic;
          wea : in  std_logic;
          web : in  std_logic;
          inclock_a: in  std_logic := '0';
          inclock_b: in  std_logic := '0';
          outclock_a: in  std_logic := '0';
          outclock_b: in  std_logic := '0';
          outrega_ar: in  std_logic := '0';
          outregb_ar: in  std_logic := '0');
end component;

mrom

component mrom
  generic(lpm_width:positive;
    lpm_widthad:positive;
    lpm_numwords:natural := 0;
    lpm_address_control :regis_type := LPM_UNREGISTERED;
    lpm_file:string := "init";
    lpm_type: string := "mrom";
    lpm_outdata:regis_type := LPM_UNREGISTERED);
  port (address: in  std_logic_vector(lpm_widthad-1 downto 0);
          q: out std_logic_vector(lpm_width-1   downto 0);
          memenab : in  std_logic := '1');
end component;
Cypress-specific Options

This section describes the Cypress-specific options that you can set in either the GUI or a Tcl script.

Cypress Device Mapping Options

When you select Project -> Implementation Options, you can set the target Cypress technology.

set_option Command for Cypress

The set_option command lets you specify the same device mapping options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on specific options for the Cypress architecture. For a complete list of available options, refer to set_option on page 4-15.

The set_option command supports the following option for the Cypress architecture:

- **-technology keyword** – sets the target technology for the implementation. The keyword must be one of the following Cypress architecture names:

  Delta39K  Flash370i  PSI  Quantum38K  Ultra37000
Cypress Attribute and Directive Summary

Attribute and Directive

The following table summarizes the synthesis and Cypress-specific attributes and directives available with the Cypress technology. Complete descriptions and examples can be found in Chapter 7, Synthesis Attributes and Directives.

Table D-1: Cypress attributes and directives

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>parallel_case (D)</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>syn_black_box (D)</td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td>syn_encoding (D)</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_enum_encoding (D)</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_isclock (D)</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td>syn_keep (D)</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_noprune (D)</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td>syn_preserve (D)</td>
<td>Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.</td>
</tr>
<tr>
<td>syn_ramstyle</td>
<td>Determines the way in which RAMs are implemented for the 38K and 39K families only.</td>
</tr>
<tr>
<td>syn_sharing (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
<tr>
<td>syn_state_machine (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td>syn_tco&lt;n&gt; (D)</td>
<td>Defines timing clock to output delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_tpd&lt;n&gt; (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_tristate (D)</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td>syn_tsu&lt;n&gt; (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>translate_off/translate_on (D)</td>
<td>Specifies sections of code to exclude from synthesis, such as simulation-specific code.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
APPENDIX E

Designing with Lattice

This chapter discusses the following topics for synthesizing Lattice designs:

- Overview on page E-2
- Lattice ORCA Devices on page E-4
- Lattice GAL and isp Devices on page E-11
- MACH Devices on page E-12
- Lattice Attribute and Directive Summary on page E-16
Overview

Supported Lattice Device Families

The Synplify synthesis tool creates technology-specific netlists for various Lattice device families. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

Synthesis Output

The Synplify synthesis tool outputs EDIF netlist files for use with the Lattice ispExpert application and the ORCA Foundry place-and-route tool. These EDIF files have extension “edf” for ispDesignExpert and “edn” for ORCA Foundry.

Constraint Files

You use constraint files to specify timing constraints, general attributes (Verilog and/or VHDL) and Lattice-specific attributes. You can use the SCOPE spreadsheet (see SCOPE Window on page 6-3) to generate and manage your constraint files.

Instantiating Macros and Black Boxes

You can instantiate Lattice macros, such as gates, counters, flip-flops and I/Os, by using the Synplify synthesis tool-supplied Lattice macro libraries, which contain predefined Lattice black-box macros.

Using Lattice Macro Libraries (Verilog)

To instantiate the supplied Verilog Lattice macro libraries, you add the appropriate Lattice macro library filename to the top of your source files list. These are the files containing the Verilog Lattice macro libraries:

- for ORCA devices:
  synplify_install_dir/lib/lucent/orca*.v
• for non-ORCA devices:
  
  synplify_installation_dir/lib/cpld/Lattice.v

For general information about instantiating black boxes in Verilog, see
_Instaniating Black Boxes in Verilog_ on page 8-40.

**Using Lattice Macro Libraries (VHDL)**

The supplied VHDL Lattice macro libraries are only required when performing simulation. However, it is generally good practice to add them to the code. The names of the libraries depend on the map name that the simulator uses to point to the specified libraries. The map name is often user-defined.

To use the supplied VHDL Lattice macro libraries, you add the appropriate _library_ and _use_ clauses to your VHDL source code at the beginning of the design units that instantiate the macros.

    library Lattice;
    use Lattice.components.all;

See also _Instantiating Black Boxes in VHDL_ on page 9-86.

Here are examples of the VHDL syntax to add for the Lattice ORCA Series 2 and ORCA Series 3:

**Example: Orca Series 2 VHDL Syntax**

    library orca2;
    use orca2.orcacomp.all;

**Example: Orca Series 3 VHDL Syntax**

    library orca3;
    use orca3.orcacomp.all;

See also _Instantiating Black Boxes in VHDL_ on page 9-86.
Lattice ORCA Devices

Device Mapping Options (ORCA)

Device mapping options are selected from the Device tab in the Options for Implementation dialog box (click Impl Options in the Project window or select Project -> Implementation Options from the Project view to display the dialog box).

Fanout Limits

Large fanouts can cause large delays and routability problems. During technology mapping, the Synplify synthesis tool automatically maintains reasonable fanout limits, keeping the fanout under the limit you specified.

1. To set a guideline other than the default for the fanout limit, display the Device tab of the Options for Implementation dialog box.

2. Type in an integer for the guideline limit. This is a guideline for the number of fanouts for a given driver. The default is 100.

   The synthesis tool uses the fanout limit as a guideline unless you specify it as a hard limit. The tool first reduces fanout by replicating the driver of the high fanout net and splitting the net into segments. Replication can affect the number of register bits in your design. If replication is not possible, the tool buffers the signals. Buffering is more expensive in terms of intrinsic delay and resource consumption, and is therefore not used unless the fanout exceeds the guideline.

3. Click the View Log button to display the net buffering report.

   The log file contains a net buffering report that shows how many nets were buffered or had their source replicated, and the number of segments created for the net.
I/O Insertion

The Synplify synthesis tool inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist unless you disable I/O insertion. You can override which I/O pads are used by instantiating the I/O pads directly. If you manually insert I/O pads, you only insert them for the pins that require them.

1. If you do not want the synthesis tool to insert any I/O pads, click the Disable I/O Insertion check box on the Device tab of the Options for Implementation dialog box.

   This option is useful to see how much area your blocks of logic take up before synthesizing an entire FPGA. Note that if you disable automatic I/O insertion, you will not get any I/O pads in your design unless you manually instantiate them yourself.

2. Manually instantiate I/O pads for specific pins, if you need them.

GSR Resource

The ORCA family has a resource for a GSR (global set reset) signal. The resource is a prerouted signal that goes to the reset input of each flip-flop in the FPGA. The GSR is connected to all flip-flops that are tied to the GSR line, regardless of any other defined reset signals. When the GSR is activated, all registers will be reset.

The Synplify synthesis tool creates a GSR instance to access the resource, if it is appropriate for your design. Using this resource instead of general routing for a reset signal can have a significant positive impact on the routability and performance of your design.

By default, if there is a single reset in your design, the synthesis tool connects that reset signal to a GSR instance. It forces this even if some flip-flops have no reset. Usually, flip-flops without set or reset can be safely initialized because the reset is only used when the device is turned on. If this is not the case, you must turn off the tool’s forced use of GSR.

To turn off the use of GSR, click off the Force GSR Usage check box on the Device tab of the Options for Implementation dialog box. When this option is off, all flip-flops must have resets, and all the resets use the same signal before the tool uses GSR.
Appendix E: Designing with Lattice

Lattice ORCA Devices

set_option Command for Lattice ORCA

Through the set_option command you specify the same device mapping options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on specific options for the Lattice ORCA architecture. For a complete list of options, see set_option on page 4-15.

The set_option command supports the following options for Lattice ORCA.

Table E-1: Lattice ORCA set_option command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Sets the target technology for the implementation. Keyword must be one of</td>
</tr>
<tr>
<td></td>
<td>the following Lattice ORCA architecture names: orcafpsc, orca2c, or orca3c,</td>
</tr>
<tr>
<td></td>
<td>orca4e.</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Refer to the Project -&gt;</td>
</tr>
<tr>
<td></td>
<td>Implementation Options dialog box for available part choices. The Package</td>
</tr>
<tr>
<td></td>
<td>option is set in Lattice’s ORCA Foundry.</td>
</tr>
<tr>
<td>-speed_grade value</td>
<td>Sets the speed grade for the implementation.</td>
</tr>
<tr>
<td>-maxfan value</td>
<td>Sets the fanout limit guideline for the open Project. The default fanout</td>
</tr>
<tr>
<td></td>
<td>limit is 100. For information about setting fanout limits, see Fanout</td>
</tr>
<tr>
<td></td>
<td>Limits on page E-4.</td>
</tr>
<tr>
<td>-force_gsr 1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>routing resources for the open Project. The default value is true.</td>
</tr>
<tr>
<td></td>
<td>For additional information about GSR, see GSR Resource on page E-5.</td>
</tr>
<tr>
<td>-block 1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Project. The default value is false (enable I/O pad insertion). For</td>
</tr>
<tr>
<td></td>
<td>additional information about disabling I/O pads, see I/O Insertion on page</td>
</tr>
<tr>
<td></td>
<td>E-5.</td>
</tr>
</tbody>
</table>

Inferring Registers with the Compiler

The compiler can infer additional register primitive types so that the mapper can implement a better solution when it targets Lattice ORCA technologies:

- Registers with clock enables—The compiler identifies and extracts clock enable logic for registers. The RTL view displays registers with
enables as special primitives with an extra pin for the enable signal. The special primitives are DFFE, DFFRE, DFFSE, DFFRSE, DFFPATRE and DFFPATRSE. Set syn_direct_enable in the HDL source file (not in the SCOPE spreadsheet or a constraint file) to direct the compiler to infer desirable clock enables. For further syntax details see syn_direct_enable on page 7-53. By default, registers without clock enables will be inferred.

Figure E-1: Registers with Clock Enables
- Registers with synchronous sets/resets—The compiler identifies and extracts registers with synchronous sets and resets, and passes them to the mapper. The special primitives are SDFFR, SDFFS, SDFFRS, SDFFPAT, SDFFRE, SDFFSE and SDFFPATE. The compiler does this automatically and does not require a directive.

Figure E-2: Flip-flops with Synchronous Sets and Resets
Appendix E: Designing with Lattice

Forward annotation

The Synplify synthesis tool generates Lattice ORCA-compliant constraint files from selected constraints that are forward-annotated (read in and then used) by the Lattice ORCA place-and-route program. The Lattice ORCA constraint files have a “prf” extension (.prf).

By default, Lattice ORCA constraint files are generated from the synthesis tool’s constraints. You can then forward-annotate these files to the place-and-route tool. To disable this feature, untoggle the Write Vendor Constraint File box (on the Implementation Results tab of the Options for implementation dialog box).

The synthesis tool generates constraint files for Lattice’s ORCA Foundry place-and-route tool. Lattice ORCA’s flow automatically copies the contents of the synthesis tool’s .prf file into the Lattice ORCA .prf file.

To forward-annotate your constraint files from the SCOPE spreadsheet to the ORCA Foundry:

1. Set your constraints using the SCOPE spreadsheet.

   Multicycle Paths (to, from or through) – Choose the Multicycle Paths tab, select or type the relevant instance under the to, from or through column. Next, type the number of clock cycles under the cycles column.

   False Paths (to, from or through) – Choose the False Paths tab, select or type the relevant instance under the to, from or through column.

   Other pane from and to – Type define_false_path and/or define multicycle path in the Command field. In the Arguments field, type -from and the source port or register name, and -to and the destination port or register name. The table below shows you an example of using from and to in the SCOPE spreadsheet’s Other tab.

Table E-2: Example: the SCOPE spreadsheet’s Other tab, from and to

<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>define_false_path</td>
<td>-from in1_int -to output</td>
</tr>
<tr>
<td>define_multicycle_path</td>
<td>-from in0_int -to output 2</td>
</tr>
<tr>
<td>define_false_path</td>
<td>-from in* -to out*</td>
</tr>
</tbody>
</table>
2. Run your design. The Synplify synthesis tool creates a .prf file in the same directory as your result files.

3. Open Lattice’s ORCA Foundry place-and-route tool. Run the Map stage, which will create the .prf file.

4. This .prf file will automatically incorporate all information present in the .prf file generated by the synthesis tool.

5. Run the PAR and BIT stages in Foundry.

The following table summarizes forward annotation options supported by the Synplify synthesis tool for Lattice ORCA in the SCOPE spreadsheet.

Table E-3: Forward-annotated Lattice ORCA options using the SCOPE spreadsheet

<table>
<thead>
<tr>
<th>ORCA Options in the SCOPE Spreadsheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>False Paths</td>
</tr>
<tr>
<td>X X X</td>
</tr>
<tr>
<td>Multicycle Paths</td>
</tr>
<tr>
<td>X X X</td>
</tr>
<tr>
<td>(Timing-driven synthesis and forward annotation)</td>
</tr>
</tbody>
</table>

**Supported Timing Constraints**

The following timing constraints are supported for Lattice ORCA:

- `define_reg_input_delay`
- `define_reg_output_delay`
- `define_clock`
- `define_multicycle_path`
- `define_false_path`
- `global frequency`
Synthesis Reports

The Synplify synthesis tool generates synthesis reports for Lattice ORCA designs that include a resource usage report, a timing report, and a net buffering report. You view these reports via View -> View Log File.

- The net buffering report lists each net that had its source replicated, and the number of segments created for the net. The summary reports the number of registers and LUTs added by replication.

- The resource usage report lists the number of each type of cell used, and the number of look-up tables and registers.

In the ORCA families, four 4-input look-up tables and four registers fit into each “function unit” (PFU). After placement and routing, the ORCA Foundry software issues a report with the number of occupied PFUs. An occupied PFU indicates that at least one look-up table or register was used. Because the synthesis tool report references LUTs, comparison between the tool’s report and the Foundry report should be based on LUTs.

Example: Lattice ORCA 2C Resource Usage Report

```
Logic resources: 2 PFUs of 100 (2%)
Register bits: 2 of 400 (1%)
I/O cells: 10

Details:
FD1S3AX: 1
FD1S3IX: 1
GSR: 1
IBM: 6
INV: 2
OB6: 4
ORCALUT4: 0 (packable)
ORCALUT4: 1 (not packable)
VHI: 1
VLO: 1
```
Lattice GAL and isp Devices

Device Mapping Options (GAL and isp)
You set device mapping options for the Lattice GAL and isp technologies from the Device tab in the Options for implementation dialog box (click Impl Options in the Project window or select Project -> Implementation Options from the Project view to display the dialog box). You can set the technology, part, speed grade, and package.

set_option Command for GAL and isp
You use this command to set options like the target technology, device architecture, and synthesis styles. This command lets you specify the same device mapping options as you do with Project -> Implementation Options.

This section provides information on specific options for the Lattice GAL and isp architectures. For a complete list of options for this command refer to set_option on page 4-15.

Table E-4: Lattice GAL and isp set_option command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Sets the target technology for the implementation. Keyword must be one of the following Lattice GAL and isp architecture names: ispgal, ispgdx, ispgdx2, plsi1k, plsi2k, islpgal5000ve_ups, plsi5k, or plsi8k.</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Refer to the GUI for the most current list (Project -&gt; Implementation Options).</td>
</tr>
<tr>
<td>-speed_grade value</td>
<td>Sets the speed grade for the implementation. Refer to the GUI for the most current list (Project -&gt; Implementation Options).</td>
</tr>
<tr>
<td>-package package_name</td>
<td>Specifies a package for the implementation. Refer to the GUI for the most current list (Project -&gt; Implementation Options).</td>
</tr>
</tbody>
</table>
Appendix E: Designing with Lattice

MACH Devices

GAL and isp Attributes

See *Lattice Attribute and Directive Summary* on page E-16 for a summary of the synthesis attributes and directives you can use with GAL and isp technologies.

MACH Devices

Device Mapping Options (MACH)

Select device mapping options for the Lattice MACH technology from the Device tab in the Options for implementation dialog box (click Impl Options in the Project window or select Project -> Implementation Options from the Project view to display the dialog box).

Map Logic to Macrocells

When this option is set (checked), the design is mapped to macrocells cells (instead of primitive gates) during synthesis. The default value is unchecked which can sometimes give a smaller result.

If you set this option, the following three additional options are available:

- Percentage of Design to Optimize for Timing on page E-12
- Maximum Cell Fanin on page E-13
- Maximum Terms/Macrocell on page E-13

Percentage of Design to Optimize for Timing

The Percent of design to optimize for timing option is available only when the Map logic to macrocells option is enabled. This option sets the percentage of nets you want optimized during synthesis. The default value is 0 percent. See *Area/Delay Trade-off for CPLDs* on page 1-19.
Maximum Cell Fanin

The Maximum Cell Fanin option sets the maximum fanin during synthesis. You may be able to improve routability by reducing the fanin limit. The default value is 20. The Maximum Cell Fanin option is available only when the Map logic to macrocells option is enabled.

Maximum Terms/Macrocell

The Maximum Terms/Macrocell option sets the maximum number of terms per macrocell during synthesis. The default value is 16.

MACH Tcl Command Options

set_option Command for MACH

You use this command to set synthesis options such as the target technology, device architecture, and synthesis styles. Through the set_option command you specify the same device mapping options as you do through the dialog box displayed in the Project view with Project -> Implementation Options or the Impl Options or New Impl button.

This section provides information on specific options for the Lattice MACH architecture. For a complete list of options for this command, refer to set_option on page 4-15.

Table E-5: Lattice MACH set_option command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology keyword</td>
<td>Sets the target technology for the implementation. Keyword is the MACH architecture name: mach, ispmach4000b, ispmach4000c, ispmach5000b, ispmach5000mx or ispmach5000vg.</td>
</tr>
<tr>
<td>-part part_name</td>
<td>Specifies a part for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available part choices. The package and speedgrade options are set in the Lattice software.</td>
</tr>
</tbody>
</table>
Table E-5: Lattice MACH set_option command options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-map_logic 1</td>
<td>0</td>
</tr>
<tr>
<td>-areadelay percent_value</td>
<td>Sets the percentage of nets you want optimized during synthesis for the open project. This option is available only if set_option -map_logic is set to true. The default value is 0.</td>
</tr>
<tr>
<td>-maxfanin value</td>
<td>Sets the maximum fanin during synthesis for the open project. You might improve routability by reducing the fanin limit. The default value is 20. This option is available only if set_option -map_logic is set to true.</td>
</tr>
<tr>
<td>-maxterms value</td>
<td>Sets the maximum number of terms per macrocell during synthesis for the open project. The default value is 16. This option is available only if set_option -map_logic is set to true.</td>
</tr>
</tbody>
</table>

**project Command for MACH**

This section provides information on specific project command options for the Lattice MACH architecture. For a complete list of options for this command, refer to *project on page 4-12.*

The project command supports the following option for the Lattice MACH architecture:

- `-result_format edif` Changes the synthesis result file format for the open Project. The synthesis result file contains the synthesized netlist created by the Synplify synthesis tool.

If you use this command, it should immediately precede the command for setting the path to the synthesis result file (project -result_file) at the end of the Tcl file, just before any project -run or project -save commands.
Using this command is equivalent to interactively choosing the Implementation Results tab in the Options for implementation dialog box and setting the Result Format value.

**MACH Attributes**

See *Lattice Attribute and Directive Summary on page E-16* for a summary of the synthesis attributes and directives you can use with MACH technologies.
Lattice Attribute and Directive Summary

The following table summarizes the synthesis and Lattice-specific attributes and directives available with the Lattice Technology. Complete descriptions and examples can be found in Chapter 7, Synthesis Attributes and Directives.

Table E-6: Lattice attributes and directives

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>din</td>
<td>Specifies that the input register be placed next to the I/O pad.</td>
</tr>
<tr>
<td>dout</td>
<td>Specifies that the output register be placed next to the I/O pad. (ORCA 2 family only; in ORCA 3 and ORCA 4 FPGAs this has been replaced by registered I/Os)</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>loc, lock</td>
<td>Specifies pin locations for Lattice I/Os. (loc: ORCA; lock: non-ORCA)</td>
</tr>
<tr>
<td>orca_padtype</td>
<td>Specifies the pad type for a Lattice ORCA I/O. (ORCA only)</td>
</tr>
<tr>
<td>orca_props</td>
<td>Specifies Lattice ORCA I/O properties for forward annotation. (ORCA only)</td>
</tr>
<tr>
<td>parallel_case (D)</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>syn_black_box (D)</td>
<td>Defines a black box for synthesis.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
### Table E-6: Lattice attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_direct_enable (D)</td>
<td>Identifies which signal is to be used as the enable input to an enable flip-flop when multiple candidates are possible. (ORCA only)</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_enum_encoding (D)</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_isclock (D)</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td>syn_keep (D)</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>Sets a fanout limit for an individual input port or register output. (ORCA only)</td>
</tr>
<tr>
<td>syn_netlist_hierarchy</td>
<td>Determines if the EDIF output netlist is flat or hierarchical. (ORCA only)</td>
</tr>
<tr>
<td>syn_noarrayports</td>
<td>Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals. (ORCA only)</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Turns off the automatic insertion of clock buffers. (ORCA only)</td>
</tr>
<tr>
<td>syn_noprune (D)</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td>syn_preserve (D)</td>
<td>Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.</td>
</tr>
<tr>
<td>syn_ramstyle</td>
<td>Determines the way in which RAMs are implemented for the Orca2 and Orca3 families. (ORCA only)</td>
</tr>
<tr>
<td>syn_sharing (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>syn_state_machine</code> (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td><code>syn_tco&lt;n&gt;</code> (D)</td>
<td>Defines timing clock to output delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tpd&lt;n&gt;</code> (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tristate</code> (D)</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td><code>syn_tsu&lt;n&gt;</code> (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_useioff</code> (Lattice ORCA)</td>
<td>Packs flip-flops in the I/O ring to improve input/output path timing in Orca3 designs. (ORCA only)</td>
</tr>
<tr>
<td><code>translate_off/translate_on</code> (D)</td>
<td>Specifies sections of code to exclude from synthesis, such as simulation-specific code.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
This chapter contains guidelines for synthesizing QuickLogic designs. It discusses the following topics:

- Synthesizing with QuickLogic on page F-2
- QuickLogic-specific Tcl Command Options on page F-6
- QuickLogic Attribute and Directive Summary on page F-7
- QuickLogic SpDE Commands on page F-9
Synthesizing with QuickLogic

The Synplify synthesis tool creates technology-specific netlists for various QuickLogic technologies. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

The synthesis tool outputs QDIF netlists (.qdf files), ready for use with the QuickLogic SpDE place-and-route program.

The rest of this section discusses the following topics:

- Design Guidelines on page F-2
- Device Mapping Options (QuickLogic) on page F-3
- Running the Synthesis Tool From SpDE on page F-4

Design Guidelines

Consider the following guidelines when working with QuickLogic designs:

- Use the asynchronous sets and resets built into the QuickLogic architectures.

- The QuickLogic architectures do not have internal tristates, only output and bidirectional tristates. If you code an internal tristate and target a QuickLogic device, the Synplify synthesis tool reports an error message such as this one:

  "Mapper Error: Found internal tristate driver on net <netname>. Tristates may be used to drive top level ports only."

- The synthesis tool buffers high fanout nets, but the QuickLogic SpDE program does not. You may get better timing results if you turn off buffering by the Synplify synthesis tool and turn on SpDE buffering. To turn off buffering, enter a very high number as the fanout limit (for example, 10,000). To turn on SpDE buffering, set the Tools -> Options -> LogicOptimization Speed option in SpDE.
Obtaining Quality Results Using Constraint Files

Use the SCOPE spreadsheet to manage your constraint files. Use constraint files (.sdc) to specify your timing constraints and QuickLogic-specific attributes. Add them to the source files list along with the HDL source files. The directives must be specified in your Verilog or VHDL source code, not in the SCOPE spreadsheet.

Packing IOBs Using syn_useioff

For some QuickLogic families (Eclipse, QuickSD, and QuickDSP), you can enable register packing. By default, the Synplify synthesis tool packs I/Os based on timing preferences. However, you can override this behavior by using the syn_useioff attribute to locally or globally embed peripheral registers into the I/O cells.

- Applying the attribute globally packs input, output, and bidirectional registers into the pads.
- Applying the attribute to a single port packs all flip-flops attached to that port into the pad.

See syn_useioff (QuickLogic) on page 7-112 for syntax details.

Handling Mixed Schematic and HDL Designs

QuickLogic lets you create a mixed schematic and behavioral HDL design. Create a schematic in QuickWorks, write it out as a Verilog or VHDL source code file, and then include that file with the project source files. When you synthesize, the Synplify synthesis tool recompiles and remaps your design.

Device Mapping Options (QuickLogic)

Select device mapping options from the Device tab in the Options for implementation dialog box (click Impl Options in the Project window or select Project -> Implementation Options from the Project view to display the dialog box).
**Fanout**

For faster devices, lower the fanout limit. Enter a smaller Fanout Guide value on the Device tab in the Options for implementation dialog box. The default is 16; the lowest limit is 6. In QuickLogic’s SpDE, select the options to remove buffers in the input netlist and insert buffers for timing.

The Synplify synthesis tool might do some buffering to avoid fanout limit errors when your design is read into SpDE. Remember that the synthesis tool buffers will be removed in SpDE when you set the two SpDE options as described above.

**Running the Synthesis Tool From SpDE**

To run the Synplify synthesis tool from SpDE:

1. From the SpDE File menu, choose Import -> Verilog or Import -> VHDL and select the source file that contains your design. SpDE automatically fills in the source file and synthesis result filenames in the Project window.

2. If needed, change the default part, package, and speed grade on the Device panel of the Options for implementation dialog box.

3. If you want to, change the synthesis result filename on the Implementation Results panel of the Options for implementation dialog box.

4. Optionally, specify a command filename.

   The command file is used for additional commands such as pad placements. Click the Add File button and select your command file from the dialog box. The command file must have an sc* extension (.sc*).

5. Synthesize the design by clicking Run and check the log file.

   If there are problems, a red status line appears below the source file (example: “Errors: 2, Warnings: 1, Notes: 0”). Double click the red error status line to view problems in the QuickLogic editor.

6. Save this configuration in a project file by selecting File -> Save As. Projects are files that contain the data you entered for your design such as a list of source files, the synthesis result filename, and your part, package, and speed-grade settings.
Handling Synthesis Problems

If there are errors in your source files, the Synplify synthesis tool informs you of the number and type of error (errors, warnings, or notes). To view the problems in the QuickLogic editor, double-click the status line with the error.

If you get warnings but no errors, the synthesis tool completes compilation and mapping to your target device. Even though the tool completes its run, it is important for you to analyze any warning messages before continuing further.

If you want SpDE to use the result file even if there are warnings, choose Exit from the File menu. To abort the synthesis run and not use the result file, first click Cancel button, and then Exit.

If there are syntax or synthesis errors, you can correct them, then click Run to resynthesize.

Viewing the Synthesis Tool Reports

The Synplify synthesis tool generates an estimate resource usage report for QuickLogic FPGAs. To view the usage report, click View Log.
QuickLogic-specific Tcl Command Options

This section describes the QuickLogic-specific set_option Tcl command options. These are the options you set for synthesis such as the target technology, device architecture, and synthesis styles.

set_option Command for QuickLogic

Through the set_option command you specify the same device mapping options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on specific options for the QuickLogic architecture. For a complete list of options for this command, see set_option on page 4-15 or enter the following in the Tcl Script window:

```
help set_option
```

The set_option command supports the following options for the QuickLogic architecture:

Table F-1: QuickLogic set_option command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-technology keyword</code></td>
<td>Sets the target technology for the implementation. Keyword must be one the following QuickLogic architecture name: eclipse, pASIC1, pASIC2, pASIC3, QuickFC, QuickDSP, QuickPCI, QuickRAM, or QuickSD.</td>
</tr>
<tr>
<td><code>-part part_name</code></td>
<td>Specifies a part for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available part choices.</td>
</tr>
<tr>
<td><code>-package package_name</code></td>
<td>Specifies the package for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available package choices.</td>
</tr>
<tr>
<td><code>-speed_grade value</code></td>
<td>Sets the speed grade for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available speed grades.</td>
</tr>
<tr>
<td><code>-maxfan value</code></td>
<td>Sets the maximum fanout during synthesis. You might be able to improve routability by reducing the fanout limit. The default is 16.</td>
</tr>
</tbody>
</table>
QuickLogic Attribute and Directive Summary

The following table summarizes the synthesis and QuickLogic-specific attributes and directives available with the QuickLogic technology. Complete descriptions and examples can be found in Chapter 7, Synthesis Attributes and Directives.

Table F-2: QuickLogic attributes and directives

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>parallel_case (D)</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>ql_padtype</td>
<td>Specifies pad types.</td>
</tr>
<tr>
<td>ql_placement</td>
<td>Specifies pad or instance locations.</td>
</tr>
<tr>
<td>syn_black_box (D)</td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_enum_encoding (D)</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_isclock (D)</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
Table F-2: QuickLogic attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute/Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_keep (D)</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td>syn_macro (D)</td>
<td>Prevents instantiated macros from being optimized.</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>Sets a fanout limit for an individual port or register output.</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Turns off the automatic insertion of clock buffers.</td>
</tr>
<tr>
<td>syn_noprune (D)</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td>syn_preserve (D)</td>
<td>Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.</td>
</tr>
<tr>
<td>syn_sharing (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
<tr>
<td>syn_state_machine (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td>syn_tco&lt;n&gt; (D)</td>
<td>Defines timing clock to output delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_tpd&lt;n&gt; (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_tristate (D)</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td>syn_tsu&lt;n&gt; (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The $n$ indicates a value between 1 and 10.</td>
</tr>
<tr>
<td>syn_useioff (QuickLogic)</td>
<td>Controls I/O packing.</td>
</tr>
<tr>
<td>translate_off/translate_on (D)</td>
<td>Specifies sections of code to be excluded from synthesis, such as simulation-specific code.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
QuickLogic SpDE Commands

The Synplify synthesis tool accepts an optional command file for QuickLogic devices in which you can enter special synthesis commands. This section discusses the command file and the command you can include in the file.

The .scp Command File

SpDE creates the .scp command file, which contains the pad placements made during placement and routing, and maintains those you specify in the command file. To preserve pad locations, execute this file the next time you resynthesize.

Because the command file is case sensitive, character names and case in the command file should exactly match the corresponding names and cases in the VHDL source file.

Use the command file to specify where to place input, output, and inout pads, flip-flops, and to specify particular padtypes. You can include the following commands:

- QuickLogic portprop Command on page F-9
- QuickLogic instprop Command on page F-10
- QuickLogic include Directive on page F-11

QuickLogic portprop Command

The portprop command overrides the automatically selected pad placements and pad types for your input, output, and inout ports.

Syntax

```
portprop port_name ql-placement = "placement", ql-padtype = "padtype";
```

Use ql_placement = placement to override the default pad location chosen by QuickWorks. The placement value is a string specifying the pad location for the port_name. Legal values, which depend on device packages, are documented in the QuickLogic data book.
**Note:** Start all placement arguments with "io". For example: "io8," "IO6," "Io32," "io3."

Use ql_padtype = padtype to override the pad type.

Legal values for padtype strings are as follows:

- **BIDIR** – A normal input, output, or bidirectional pad will be used.
- **INPUT** – A high-drive input pad will be used. The number of high-drive pads available varies with the part. Refer to the QuickLogic data book for more information.
- **CLOCK** – A clock pad is used. The QuickLogic clock pads are routed to special clock wires that cover the entire chip. The QuickLogic software also uses clock pads for reset and set signals.

You can specify either ql_placement or ql_padtype by itself, or you can specify them together, in any order. Their values (strings) are case insensitive. The command line should conclude with a semicolon (;).

**Examples**

```
portprop inb ql_placement = "IO5";
portprop ina ql_placement = "io3", ql_padtype = "INPUT";
```

**QuickLogic instprop Command**

Use the instprop command to specify QuickLogic’s placement of instances during placement and routing. You can place explicitly netlisted QuickLogic I/O pads (high-drive, clock, input, output, tristate, and bidirectional pads) and flip-flops. Refer to the information on Fixed Placement in the *QuickLogic pASIC Toolkit User’s Guide* for more information.

**Syntax**

```
instprop instance_path ql_placement = "placement" ;
```

- The placement argument is a string giving the placement location for your instance_path. Legal values depend on the device package. (Examples: "G5", "io6", "io32", "io3").
• The placement argument is a case-insensitive string. Do not forget the semicolon at the end of the command line.

**Note:** The instance_path value can be a hierarchical path name to the instance. Following Tcl formatting rules, if instance_path is hierarchical, you do not include the top-level module in the value. Start with the name of the first instance in the top-level module that makes up the hierarchical name.

**Examples**

```tcl
instprop fast_flip_flop ql_placement = "G5";
instprop my_io_pad ql_placement = "IO5";
```

**QuickLogic include Directive**

The include directive is used to read in another command file from within your command file (which could be either a project file or a Tcl script).

**Syntax**

```tcl
include "filename";
```

The filename argument follows the naming conventions of the operating system (not case sensitive for PCs, but case sensitive for UNIX), and it must be enclosed in double quotes (""").

include allows you to use the SpDE place-and-route pad placement location file during resynthesis, after placement and routing. For example, synthesize a design, place-and-route it through SpDE, and burn in some parts. Then, modify and resynthesize the design, keeping the same pin locations so that the new part works in the existing system, without changed FPGA connections. If you include the pad placement location file initially generated by SpDE, the Synplify synthesis tool preserves the pad locations during synthesis. The file extension of the pad placement location file created by SpDE is “scp” (.scp).
This chapter contains guidelines for synthesizing QuickLogic designs. It discusses the following topics:

- Synthesizing Triscend Designs on page G-2
- Triscend-specific Tcl Command Options on page G-3
- Triscend Attribute and Directive Summary on page G-4
Appendix G: Designing with Triscend

Synthesizing Triscend Designs

Supported Triscend Device Families

The Synplify synthesis tool creates technology-specific netlists for various Triscend FPGA families. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

Obtaining Quality Results Using Constraint Files

Constraint files contain user-specified timing constraints and vendor-specific attributes. Add them to the source files list along with your HDL source files. Constraint files can contain timing constraints, general attributes (Verilog or VHDL), and Triscend-specific Attributes. Use the SCOPE constraints editor to manage your constraint files.

Device Mapping Options (Triscend)

Device mapping options vary from vendor to vendor and, when included, are selected from the Device tab in the Options for implementation dialog box (click Impl Options in the Project window or select Project -> Implementation Options from the Project view to display the dialog box).

I/O Insertion

The Synplify synthesis tool suppresses I/O pad insertion for inputs, outputs, and bidirectional pins in the output netlist unless you enable I/O insertion. The default setting turns this option off. You can override which I/O pads are used by instantiating the I/O pads directly. If you manually insert I/O pads, you only insert them for the pins that require them.

1. If you want the synthesis tool to insert any I/O pads, click the Disable I/O Insertion check box on the Device tab of the Options for implementation dialog box.
Triscend-specific Tcl Command Options

This section describes the Triscend-specific set_option Tcl command options. These are the options you set for synthesis such as the target technology, device architecture, and synthesis styles.

set_option Command for Triscend

Through the set_option command you specify the same device mapping options as you do through the dialog box displayed in the Project view with Project -> Implementation Options or the Impl Options or New Impl button.

This section provides information on specific options for the Triscend architecture. For a complete list of options for this command, see set_option on page 4-15

The set_option command supports the following options for Triscend:

Table G-1: Triscend set_option Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology</td>
<td>Sets the target technology for the implementation. Keyword</td>
</tr>
<tr>
<td>-part</td>
<td>Specifies a part for the implementation. Refer to Project -&gt; Implementation Options dialog box for available part choices.</td>
</tr>
<tr>
<td>-speed_grade</td>
<td>Sets the speed grade for the implementation. Refer to Project -&gt; Implementation Options dialog box for available choices.</td>
</tr>
<tr>
<td>-block 1</td>
<td>0</td>
</tr>
</tbody>
</table>
Triscend Attribute and Directive Summary

The following table summarizes the synthesis and Triscend-specific attributes and directives available with the Triscend Technology. Complete descriptions and examples can be found in Chapter 7, Synthesis Attributes and Directives.

Table G-2: Triscend attributes and directives

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_netlist_hierarch</td>
<td>Determines if the EDIF output netlist is flat or hierarchical.</td>
</tr>
<tr>
<td>syn_noarrayports</td>
<td>Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Turns off the automatic insertion of clock buffers.</td>
</tr>
<tr>
<td>syn_props</td>
<td>Specifies Triscend attributes to forward-annotate to the gate-level netlist.</td>
</tr>
<tr>
<td>syn_useenables</td>
<td>Prevents generation of registers with clock enable pins.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
This chapter contains guidelines for synthesizing Xilinx designs. It discusses the following topics:

- Supported Xilinx Devices on page H-2
- Device Mapping Options on page H-2
- Design Considerations on page H-5
- Design Elements on page H-10
- I/Os and Pin Locations on page H-18
- Using Spartan on page H-27
- Using Virtex on page H-30
- Using XC4000 on page H-34
- Using XC3000 and XC5200 on page H-37
- Using Xilinx CLPDs on page H-40
- Xilinx Attribute and Directive Summary on page H-43
Supported Xilinx Devices

The Synplify synthesis tool creates technology-specific netlists for various Xilinx FPGA and CPLD families. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Options for implementation dialog box (see Options for implementation Dialog Box on page 3-31).

The synthesis tool generates XNF or EDIF netlists that are used as input to the Xilinx place-and-route tools. EDIF netlists are for use with the Xilinx ISE place-and-route software.

With the exception of Virtex and Spartan2 families, Xilinx does not support D flip-flops with both asynchronous sets and resets. A warning message is reported if you code them in your design. You can find definitions for D flip-flops in the synplify_install_dir/lib/xilinx directory.

Device Mapping Options

Set device mapping optimization options before you synthesize your design. Specific instructions are included in the following sections: Using Spartan on page H-27, Using Virtex on page H-30, Using XC4000 on page H-34, and Using XC3000 and XC5200 on page H-37.

The following device mapping options are briefly described:

- Fanout Guide on page H-2
- Disable I/O Insertion on page H-3
- Force GSR Usage on page H-4

Fanout Guide

Large fanouts can cause large delays and routability problems. The Synplify synthesis tool maintains reasonable fanout guides automatically, while providing you the flexibility to suggest maximum fanout guidelines. You set this global fanout guide via the Device panel of the Options for
implementation dialog box (see Options for implementation Dialog Box on page 3-31), by entering an integer value in the Fanout Guide field. The minimum value you can enter is 8. The default is 100.

During technology mapping, the Synplify synthesis tool tries to keep the fanout to less than the specified fanout guide. However, the fanout guide serves only as a guideline, not a hard limit. This means that the synthesis tool takes it into account but does not always respect it absolutely. If it imposes constraints that interfere with optimization then it is not respected.

Fanout is first reduced by replicating the driver of the high fanout net and splitting the net into segments. This replication can affect the number of register bits and LUTs in your design. If replication is not possible, the signals are buffered. Buffering increases intrinsic delay and consumes more resources, and is therefore not used until a slightly higher fanout limit has been reached than is specified.

For extremely large clock fanout nets, consider 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.

Click View Log to display the net buffering report in the log file. The report shows how many nets were buffered or had their sources replicated, and the number of segments created for the net.

See also syn_maxfan on page 7-70, for information on the syn_maxfan attribute. The fanout guide provides a global fanout guideline, whereas the syn_maxfan attribute provides a fanout guideline that you can assign locally to an individual input port or register output.

**Disable I/O Insertion**

I/O (input/output) pads are automatically inserted into the design. If you manually instantiate I/Os, I/Os are inserted only for the pins that need them.

If you do not want to insert any I/Os in the design, select the Disable I/O Insertion check box in the Set Device Options dialog box. This is useful for bottom-up compiles, because you can check the area your logic blocks use before you synthesize the whole design. If you disable I/O insertion, you will not get any I/Os in the design unless you manually instantiate them.
Appendix H: Designing with Xilinx
Device Mapping Options

Force GSR Usage

All Xilinx XC families have a startup block for Global Set/Reset (GSR). The GSR resource is a pre-routed signal that goes to the set or reset input of each flip-flop in your design. Using this resource instead of general routing for a set or reset signal can have a significant positive impact on the routability and performance of your design.

For XC4000 families, the Synplify synthesis tool forces the creation of a startup block to access the GSR resource, if it is appropriate for your design.

• Single set/reset design –
  By default, if your design has a single set or reset used in your design, it is connected to a startup block. This is done even if some flip-flops have no sets or resets. Usually, flip-flops without sets or resets can be safely initialized because the set or reset is only used when the device is turned on. If this is not the case, then you will need to turn off the Force GSR Usage option. To turn off the Force GSR Usage option, choose Project->Implementation Options -> Device, and clear the Force GSR Usage check box. With the Force GSR Usage option disabled, all flip-flops are required to have a set or reset and the set or reset must be the same before GSR is used.

• Multiple set/reset design –
  When multiple set or reset signals are used in your design, the Synplify synthesis tool does not automatically create a startup block for GSR. If you still want to use one of the set or reset signals for GSR, you must instantiate a STARTUP_GSR component manually.

For Virtex designs, you must manually instantiate the startup block with a STARTUP_GSR component. Refer to Instantiating Startup Blocks on page H-6 for more information about this process.
Design Considerations

The following topics should be considered when implementing designs that use any of the supported Xilinx technology families:

- Specifying Constraints and Attributes on page H-5
- Creating Mixed Schematic and HDL Designs on page H-5
- Instantiating Startup Blocks on page H-6
- Packing IOBs Using syn_useioff on page H-7
- Instantiating Macros and Black Boxes on page H-7
- Forward-annotating Timing Constraints to Placement and Routing on page H-8
- Viewing Resource Usage Reports on page H-9

Specifying Constraints and Attributes

Specify timing constraints, general HDL attributes, and Xilinx-specific attributes to improve your design. Manage these files with the SCOPE constraints and attributes editor. See Synthesis Attributes and Directives on page 7-1 and the Synplify User Guide for more information.

Creating Mixed Schematic and HDL Designs

To create a mixed schematic and behavioral HDL design:

1. Create the schematic. Use your Xilinx-compatible schematic editor and write out the schematic as an EDIF or XNF file.

2. Write the HDL design. When you create your Verilog or VHDL design, instantiate a black box for the schematic block to define the port interface. Make sure the entity/module name matches the EDIF/XNF name from the schematic.

3. Synthesize the design. The generated EDIF or XNF file will contain a reference to your black box. See Instantiating Black Boxes in Verilog on page 8-40, or Instantiating Black Boxes in VHDL on page 9-86 for more information.
4. Merge the schematic and HDL design. Place the EDIF or XNF file for your schematic in the same directory as the EDIF or XNF file generated during synthesis, then use the Xilinx place-and-route tool to merge them.

### Instantiating Startup Blocks

Xilinx startup blocks are provided in the `synplify_install_dir/lib/xilinx` directory. They are available in both Verilog (.v) and VHDL (.vhdl) formats. The following table summarizes the information for different families.

**Table H-1: Xilinx startup block assignment, by family**

<table>
<thead>
<tr>
<th>Family</th>
<th>Filename</th>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex</td>
<td>virtex</td>
<td>STARTUP_VIRTEX_GSR</td>
<td>Instantiate the components directly. You can use them independently, because the three blocks are merged to form a single startup block in the EDIF result file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP_VIRTEX_GTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP_VIRTEX_CLK</td>
<td></td>
</tr>
<tr>
<td>Virtex-E</td>
<td>virtexe</td>
<td>STARTUP_VIRTEX2</td>
<td>Instantiate the component.</td>
</tr>
<tr>
<td>Virtex2</td>
<td>virtex2</td>
<td>STARTUP_VIRTEX2</td>
<td>Instantiate the component.</td>
</tr>
<tr>
<td>Virtex2p</td>
<td>virtex2p</td>
<td>STARTUP_VIRTEX2</td>
<td>Instantiate the component.</td>
</tr>
<tr>
<td>XC4000 or XC5200</td>
<td>xc4000</td>
<td>STARTUP_GSR</td>
<td>Instantiate the components directly. You can use them independently, because the three blocks are merged to form a single startup block in the EDIF result file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP_GTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP_CLK</td>
<td></td>
</tr>
<tr>
<td>Spartan2</td>
<td>spartan2</td>
<td>STARTUP_SPARTAN2_GSR</td>
<td>Instantiate the components directly. You can use them independently, because the three blocks are merged to form a single startup block in the EDIF result file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP_SPARTAN2_GTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP_SPARTAN2_CLK</td>
<td></td>
</tr>
</tbody>
</table>

a. Both Verilog (.v) and VHDL (.vhdl) files are provided.
Packing IOBs Using syn_useioff

When a register drives an output, or an input drives a register, you can place the register in a CLB or in an IOB. Depending on the design requirements, you may want to pack the registers in the IOB instead of the CLB. Reasons for this include:

- The chip interfaces with another chip and either the register-to-output or input-to-register delay has to be minimized. Packing the register in the IOB instead of the CLB is more advantageous.

- If CLB resources are limited, it may be worth packing the registers in IOBs to free up CLB resources.

By default, the Synplify synthesis tool packs I/Os based on timing preferences. However, you can override this behavior by using the syn_useioff attribute to locally or globally embed flip-flops in the IOBs.

Instantiating Macros and Black Boxes

You can create an instance of any externally defined macro, including a user-defined macro or Xilinx macro such as an I/O or flip-flop, by instantiating a black box in your Verilog or VHDL source code. These black boxes are empty Verilog module descriptions and VHDL component declarations.

To instantiate Xilinx macros such as gates, counters, flip-flops, or I/Os in a Verilog design, use the predefined macro libraries that define the Xilinx macros as black boxes. The Verilog libraries are in the synplify_install_dir/lib/xilinx directory.

For VHDL designs, add the appropriate library and use clauses to the beginning of the design units that instantiate the macros.

```
library family;
use family.components.all;
```

To review the available macros, check the VHDL libraries (family.vhd) in the synplify_install_dir/lib/xilinx directory.
Forward-annotating Timing Constraints to Placement and Routing

The Synplify synthesis tool forward-annotates different timing constraints to placement and routing through a Xilinx constraint (.ncf) file. These timing constraints include input/output delay, clock period, multicycle paths, and false paths. During synthesis, the Xilinx constraint file is generated using synthesis tool attributes and constraints.

By default, Xilinx constraint files are generated. You can disable this feature in the Project view. To do this, bring up the Options for implementation dialog box (Project -> Implementation Options), then, on the Implementation Results panel, disable Write Vendor Constraint File.

The following constraints are forward-annotated to Xilinx in the .ncf file:

- define_clock
- define_false_path
- define_input_delay
- define_multicycle_path
- define_output_delay

By default, the Synplify synthesis tool forward-annotates the define_input_delay and define_output_delay timing constraints to the Xilinx .ncf file.

The syn_forward_io_constraints attribute controls the forward annotation of the input and output delay constraints. A value of "1" or true (default) enables forward annotation, while a value of "0" or false disables forward annotation.

Use this attribute on the top level VHDL entity/architecture or Verilog module, or use the Attributes panel of the SCOPE spreadsheet to add the attribute as a global object.

If the constraint is too tight for the input-to-register or register-to-output paths, the tool will try to relax the constraints to these paths. (See Relaxing Forward-annotated Constraints on page 6-43.)
Viewing Resource Usage Reports

Synthesis reports are generated that include a resource usage report, a timing report, and a net buffering report for Xilinx designs. To view these synthesis reports, click View Log in the Project view.

Resource usage reports for Xilinx include:

- Cell usage (for example, carry chains, flip-flops)
- Total registers
- Registers packed in I/Os
- Memory cell usage
- Number of I/Os
- Global buffer usage
- LUTs
- Percent of utilization

The number of CLBs reported is within a few percentage points of the number reported by Xilinx after placement and routing.

Example: Xilinx Virtex2 Resource Usage Report

Mapping to part: xc2v40cs144-5

Cell usage:

<table>
<thead>
<tr>
<th>Cell Usage</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDP</td>
<td>4</td>
</tr>
<tr>
<td>FDC</td>
<td>13</td>
</tr>
<tr>
<td>MUXFS</td>
<td>25</td>
</tr>
<tr>
<td>MUXCY_L</td>
<td>8</td>
</tr>
<tr>
<td>MUXCY</td>
<td>1</td>
</tr>
<tr>
<td>XORCY</td>
<td>5</td>
</tr>
<tr>
<td>FDCE</td>
<td>55</td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
</tr>
<tr>
<td>VCC</td>
<td>1</td>
</tr>
</tbody>
</table>

I/O primitives:

<table>
<thead>
<tr>
<th>I/O Primitives</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBUF</td>
<td>12</td>
</tr>
<tr>
<td>OBUF</td>
<td>29</td>
</tr>
<tr>
<td>BUFGP</td>
<td>1</td>
</tr>
</tbody>
</table>
I/O Register bits: 0
Register bits not including I/Os: 72 (14%)

Global Clock Buffers: 1 of 8 (12%)

Mapping Summary:
Total LUTs: 190 (37%)

Design Elements

The following topics describe how specific design elements are implemented in the Xilinx technologies:

- Inferring RAMs on page H-10
- Inferring Registers with the Compiler on page H-14
- Mapping Latches on page H-15
- Inferring Dynamic Shift Register Lookup (SRL) Tables on page H-15

Inferring RAMs

Depending on the family, you can infer general purpose or block RAMs. You can also disable automatic RAM inference.

Inferring General Purpose RAMs

The Synplify synthesis tool can infer synchronous RAMs for the Xilinx 4000 and Virtex families from your HDL source code and generate single- or dual-port RAMs using the distributed RAM resources in the CLBs. No special input such as attributes or directives is needed. See RAM Inference on page 8-34, RAM Inference on page 9-77, and the application note RAM Inferencing with Synplify in the Synplify Support section of the Synplicity web site (www.synplicity.com) for in-depth details.
Inferring Single- and Dual-port Block RAMs in Virtex Designs

Block RAMs allow the use of enable and reset to control RAM read and write and also support multiple clocks. You can infer a Virtex family Block SelectRAM by specifying the `syn_ramstyle` attribute with a value of `block_ram` (by default, inferred RAMs use the distributed RAM resources in the CLBs). This attribute is specified for the signal (VHDL) or reg port (Verilog) that defines the RAM, or the label of the RAM component instance. For syntax information, see `syn_ramstyle` on page 7-87.

See *RAM Inference on page 8-34* and *RAM Inference on page 9-77* for in-depth details.

Single-port Block RAMs

In single-port block RAMs, the Enable signal has the highest priority; while this signal is inactive, all read and write operations are disabled. When Enable is active:

- If Reset is active, `data_out` is 0 (write operations and memory contents are unaffected).

- If Reset is inactive:
  - if Write Enable is active, `data_in` goes to `data_out` and is updated in memory
  - if Write Enable is inactive, `data_out` reflects the memory contents

The following segment of Verilog code defines the behavior of a Xilinx single-port block RAM.

```verilog
module RAMB4_S4 (data_out, ADDR, data_in, EN, CLK, WE, RST);
output [3:0] data_out;
input [7:0] ADDR;
input [3:0] data_in;
input EN, CLK, WE, RST;
reg [3:0] mem [255:0] /* synthesis syn_ramstyle="block_ram" */;
reg [3:0] data_out;

always@(posedge CLK)
  if(EN)
    if(RST == 1)
      data_out <= 0;
    else
      begin
```
if(WE == 1)
    data_out <= data_in;
else
    data_out <= mem[ADDR];
end

always @(posedge CLK)
    if (EN && WE) mem[ADDR] = data_in;
endmodule

For a code segment to map to a single-port block RAM, there must be a one-to-one correspondence between the following signals:

- read and write clocks
- read and write addresses
- enable signal
- write enable signal

Virtex2/Virtex2p block RAM supports three write modes: *writefirst*, *readfirst*, and *nochange*. These modes determine what appears at data_out when Write Enable is active.

- **Writefirst** – data_in goes to data_out
- **Readfirst** – memory goes to data_out
- **Nochange** – data_out remains unchanged

Virtex and Virtex-E only support the writefirst mode. Accordingly, a code segment that can be mapped to a Virtex2/Virtex2p single-port block RAM may not be able to be mapped to Virtex or Virtex-E single-port block RAM. These code segments can be mapped to distributed RAM. For more information, see the description of block RAM inference in the *User Guide*.

**Dual-port Block RAMs**

Except for the addition of address collision recovery logic, a dual-port RAM is essentially two, independent single-port RAMs sharing a common memory. The dual-port RAM has only one write port.

The following segment of Verilog code defines the behavior of a Virtex/Virtex2p dual-port block RAM (address collision recovery code is not shown).
module DP_Block_Ram (ADDRA, ADDRB, data_in, CLK, WEA, data_out);
output [3:0] data_out;
input [9:0] ADDRA;
input [9:0] ADDRB;
input [3:0] data_in;
input CLK, WEA;

reg [3:0] mem [1023:0] /* synthesis syn_ramstyle="block_ram" */;
reg [9:0] ADDRB_reg;

always@(posedge CLK)
if (WEA)
mem[ADDRA] = data_in;

always @(posedge CLK)
ADDRB_reg <= ADDRB;
assign data_out = mem[ADDRB_reg];
endmodule

Disabling RAM Inference

If your RAM resources are limited, designate additional instances of inferred RAMs as flip-flops and logic using the syn_ramstyle attribute (Verilog or VHDL) with a value of registers. This attribute is specified for the signal (VHDL) or reg port (Verilog) that defines the RAM, or the label of the RAM component instance. For syntax information, see syn_ramstyle on page 7-87.

Mapping ROM into Block RAM

The software in Xilinx Virtex and Virtex2 architectures supports the mapping of ROM into block RAM, provided you meet the conditions listed below and specify the block_rom option of the syn_romstyle attribute (this option is selected by default).

1. Place a dff register in front of the ROM, or place one of the following after the ROM:
Table H-2: Registers that can follow ROM (Xilinx)

<table>
<thead>
<tr>
<th>Asynchronous</th>
<th>Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>dff, dffe</td>
<td></td>
</tr>
<tr>
<td>dffr, dffre</td>
<td>sdffr, sdffre</td>
</tr>
<tr>
<td>dffs, dffse</td>
<td>sdffs, sdffse</td>
</tr>
<tr>
<td>dffpatr, dffpatre</td>
<td>sdffpatr, sdffpatre</td>
</tr>
</tbody>
</table>

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 registers and ROMs are within the same hierarchy.

3. Ensure that the number of the candidate ROM's inputs is:

   – no fewer than 8 and no more than 12 for Virtex/VirtexE, Spartan2

   or

   – no fewer than 9 and no more than 14 for Virtex2/Virtex2p

4. Ensure that the number of the candidate ROM’s outputs is 64 or fewer.

5. At least half of the addresses must 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.

Inferring Registers with the Compiler

The compiler can infer additional register primitive types so that the mapper can implement a better solution when it targets Xilinx technologies:

- Registers with clock enables—The compiler identifies and extracts clock-enable logic for registers. The RTL view displays registers with enables as special primitives with an extra pin for the enable signal. The special primitives are DFFE, DFFRE, DFFSE, DFFRSE, DFFPATRE and DFFPATRSE. Set syn_direct_enable in the HDL source file (not in the
SCOPE spreadsheet or a constraint file) to direct the compiler to infer desirable clock enables. For further syntax details see `syn_direct_enable on page 7-53`. By default, registers without clock enables will be inferred.

- Registers with synchronous sets/resets—The compiler identifies and extracts registers with synchronous sets and resets, and passes them to the mapper. The special primitives are SDFFR, SDFFS, SDFFRS, SDFFPAT, SDFFRE, SDFFSE and SDFFPATE. The compiler does this automatically and does not require a directive.

**Mapping Latches**

For Xilinx architectures that do not support level-sensitive latches, the latches are mapped to the latch components Ld_1 (level-sensitive latch), Ldc_1 (level-sensitive latch with clear), Ldp_1 (level-sensitive latch with preset), and Ldcp_1 (level-sensitive latch with preset and clear) so that you can supply implementations. Sample XNF files are included containing implementations you can use when you run the Xilinx place-and-route tool. Alternatively, you can change the XNF files for these latches to create your own implementations, before performing placement and routing.

The sample XNF files (with extension “.xnf”) for these latches are located in the `synplify_install_dir/lib/xilinx` directory.

**Inferring Dynamic Shift Register Lookup (SRL) Tables**

The Synplify synthesis tool infers dynamic and static sequential shift components in Xilinx Virtex architectures. If you do not want to automatically infer the shift registers, set the `syn_srlstyle` attribute to `registers`.

For static sequential shift components, the synthesis tool uses the output of the last register. For dynamic sequential shift components, all words in the component must have the same reset pattern in order to be inferred. Words are reset to all 0 or all 1. See the `Synplify User Guide` for details.

**VHDL Example**

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;
Verilog Example

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
I/Os and Pin Locations

I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs), are automatically inserted and the Xilinx place-and-route tool chooses the I/O locations. You can specify your own I/O locations, rather than relying on the I/O locations automatically chosen by Xilinx.

You can also manually instantiate I/Os, and even specify I/O locations for some or all of your instantiated I/Os. If you do that, I/Os only for the pins that need them are inserted.

Specifying Pin Locations and Inserting I/Os

I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs), are automatically inserted, and the Xilinx place-and-route tool chooses the I/O locations. See Automatically Inserting I/Os and Assigning Locations in Verilog on page H-19 and Manually Inserting I/Os and Assigning Locations in VHDL on page H-23.

You can also manually instantiate I/Os—see Manually Inserting I/Os and Assigning Locations in Verilog on page H-20 and Manually Inserting I/Os and Assigning Locations in VHDL on page H-23. You can also specify I/O locations for some or all of your instantiated I/Os. If you do that, I/Os are inserted by the place-and-route tool only for the pins that need them.

Using Xilinx Macro Libraries with Verilog and VHDL

For Verilog designs, add the appropriate Xilinx macro library file to the top of your project’s source files list. The files (family.v) for the libraries are located in the synplify_install_dir/lib/xilinx directory. Review the file for your target family for available macros.

For VHDL designs, add the appropriate library and use clauses to the beginning of your design units that instantiate the macros. You do not need to add files to your Project’s source files list to make the Xilinx libraries available to your designs. The files (family.vhd) for the libraries are located in the synplify_install_dir/lib/xilinx directory. Review the file for your target family for available macros.
Here is the syntax:

```
library family;
use family.components.all;
```

where `family` is `xc3000`, `xc4000`, `xc9500`, `coolrunner`, `coolrunner2`, `virtex`, `virtexe` or `virtex2`.

Here is an example:

```
library virtex;
use virtex.components.all;
```

**Assigning I/O Locations with the SCOPE Spreadsheet**

You can assign I/O locations using the Attributes panel of the SCOPE spreadsheet. This is a convenient method that also makes it easy to maintain vendor independence. Here is an example of assigning locations using the `xc_loc` attribute.

![Assigning I/O locations with the SCOPE spreadsheet](image)

**Automatically Inserting I/Os and Assigning Locations in Verilog**

Create a new top-level module and instantiate your Verilog design. Create this new top-level module to specify your I/O locations so that your vendor-specific information is separate from the rest of your design. Once you have done this, set the `xc_loc` attribute with the appropriate value for the I/Os you want to specify the locations for. If you do not specify the `xc_loc` attribute for certain I/Os, the Xilinx place-and-route tool will choose the I/O locations for them.
For example:

1. Starting with the following module declaration:

   module cnt4 (cout, out, in, ce, load, clk, rst);
   // Counter definition goes here
   endmodule

2. Create a new top-level module to place I/Os for Xilinx. This module would typically be in another file so that your original design remains untouched and technology independent.

   module cnt4_xilinx (cout, out, in, ce, load, clk, rst);

3. Specify that cout should be located at A1.

   output cout /* synthesis xc_loc="A1" */;

4. Specify that out should be in the top left (TL) of the chip.

   output [3:0] out /* synthesis xc_loc="TL" */;


   input [3:0] in /* synthesis xc_loc="P20,P19,P18,P17" */;

6. Let Xilinx automatically place the rest of the inputs.

   input ce, load, clk, rst;

7. Instantiate your design.

   cnt4 my_counter (.cout(cout), .out(out), .in(in),
   .ce(ce), .load(load), .clk(clk), .rst(rst));
   endmodule

**Manually Inserting I/Os and Assigning Locations in Verilog**

Create instances of I/Os by instantiating a black box in your Verilog source code. These black boxes are empty Verilog module descriptions that are found in supplied Xilinx macro libraries. Add the appropriate Xilinx macro library file to the top of your project’s source files list. The files (family.v) for the libraries are located in the synplify_install_dir/lib/xilinx directory.
If you instantiate I/Os, the Xilinx place-and-route tool will choose their locations. Or, you can specify your own I/O locations as follows:

1. Create a new top-level module and instantiate your Verilog design.

2. Manually instantiate the Xilinx I/Os.

3. Add the appropriate Xilinx macro library filename to the top of your Project’s source files list.

4. If you want to specify I/O locations, set the \texttt{xc\_loc} attribute for those I/Os you want to specify the locations for. If you do not specify I/O locations for some of your I/Os, the Xilinx place-and-route tool will choose the locations for them.

Verilog Example

```verilog
module cnt4 (cout, out, in, ce, load, clk, rst);
/* Your counter definition goes here, */
endmodule

/* Create a top level to place I/Os specifically for Xilinx. Any top level pins which do not have I/Os will be automatically inserted */
module cnt4_xilinx(cout, out, in, ce, load, clk, rst);
output [3:0] out;
output cout;
input [3:0] in;
input ce, load, clk, rst;
wire [3:0] out_c, in_c;
wire cout_c;

/* The `xc\_loc` attribute can be added right after the instance name like that shown below, or right before the semicolon. */

IBUF i3 /* synthesis xc\_loc="P20" */ (.O(in_c[3]), .I(in[3]));
IBUF i2 /* synthesis xc\_loc="P19" */ (.O(in_c[2]), .I(in[2]));
IBUF i1 /* synthesis xc\_loc="P18" */ (.O(in_c[1]), .I(in[1]));
IBUF i0 /* synthesis xc\_loc="P17" */ (.O(in_c[0]), .I(in[0]));

OBUF o3 /* synthesis xc\_loc="TL" */ (.O(out[3]), .I(out_c[3]));
OBUF o2 /* synthesis xc\_loc="TL" */ (.O(out[2]), .I(out_c[2]));
OBUF o1 /* synthesis xc\_loc="TL" */ (.O(out[1]), .I(out_c[1]));
OBUF o0 /* synthesis xc\_loc="TL" */ (.O(out[0]), .I(out_c[0]));
OBUF cout_p /* synthesis xc\_loc="BL" */ (.O(cout), .I(cout_c));
```
cnt4 it(.cout(cout_c), .out(out_c), .in(in_c),
    .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule

**Automatically Inserting I/Os and Assigning Locations in VHDL**

Create a new top-level design unit and instantiate your VHDL design. Use this new top-level design unit to specify your I/O locations so that your vendor-specific information is separate from the rest of your design. Specify the `xc_loc` attribute with the appropriate value for the locations of the I/Os you want to specify. If you do not specify the `xc_loc` attribute for certain I/Os, the Xilinx place-and-route tool will choose locations for them.

**VHDL Example**

```vhdl
library synplify;
entity cnt4 is
    port (cout: out bit;
        output: out bit_vector (3 downto 0);
        input: in bit_vector (3 downto 0);
        ce, load, clk, rst: in bit);
end cnt4;
architecture behave of cnt4 is
begin
    -- Behavioral description of the counter.
end behave;

    -- New top level entity, created specifically
    -- to place I/Os for Xilinx. This entity typically
    -- would be in another file, so that your original
    -- design stays untouched, and technology independent.
entity cnt4_xilinx is
    port (cout: out bit;
        output: out bit_vector (3 downto 0);
        input: in bit_vector (3 downto 0);
        ce, load, clk, rst: in bit);

    -- Place a single I/O for cout at location A1.
    attribute xc_loc :  string;
    attribute xc_loc of cout:  signal is "A1";
```
-- Place all bits of "output" in the
top-left of the chip.
attribute xc_loc of output: signal is "TL";

-- Place input(3) at P20, input(2) at P19,
-- input(1) at P18, and input(0) at P17
attribute xc_loc of input: signal is "P20, P19, P18, P17";

-- Let Xilinx place the rest of the inputs.
end cnt4_xilinx;

-- New top level architecture instantiates your design.
architecture structural of cnt4_xilinx is
-- Component declaration for your entity.
component cnt4
port (cout: out bit;
     output: out bit_vector (3 downto 0);
     input: in bit_vector (3 downto 0);
     ce, load, clk, rst: in bit);
end component;
begin
-- Instantiate your VHDL design here:
my_counter: cnt4 port map (cout, output, input,
     ce, load, clk, rst);
end structural;

Manually Inserting I/Os and Assigning Locations in VHDL

For VHDL designs, add the appropriate library and use clauses to the beginning of your design units that instantiate the macros. The Xilinx libraries are always visible in the Synplify synthesis tool, so do not add library files to your project's source files list. To see which design units are available, use a text editor of your choice and view the files (family.vhd) for the libraries which are located at synplify_install_dir/lib/xilinx (do not edit these files in any way).

Syntax

library family;
use family.components.all;

Where family is, xc4000, virtex, virtexe, or virtex2.
Steps
1. Create a new top-level module and instantiate your VHDL design.

2. Instantiate the Xilinx I/Os.

3. Add the appropriate library and use clauses (see source code below) to the beginning of your design unit that instantiates the I/Os.

4. If you wish to specify I/O locations, add the xc_loc attribute to the I/O instances for which you want to specify the locations, and give the locations values. If you leave out the xc_loc attribute, the Xilinx place-and-route tool will choose the locations.

The following example is a behavioral D flip-flop with instantiated data input I/O. The other ports will have synthesized I/Os.

VHDL Example

```vhdl
ilaium ieee, synplify;
use synplify.attributes.all;
use ieee.std_logic_1164.all;
-- Now the library and use clauses for access
-- to the Xilinx Macro Library.
library xc4000;
use xc4000.components.all;
entity place_example is
  port (q: out std_logic;
      d, clk: in std_logic);
end place_example;

architecture behave of place_example is
  signal dz: std_logic;
attribute xc_loc of I1: label is "P3";
begin
I1: IBUF port map (I=>d,O=>dz);

process (clk) begin
  if rising_edge(clk) then
    q<=dz;
  end if;
end process;
end behave;
```
Specifying Relative Location

The following three Xilinx-specific attributes specify relative position of components in your designs.

- `xc_map`
- `xc_rloc`
- `xc_uset`

The general steps are as follows:

1. Create design units (modules) and use the `xc_map` attribute to specify them as `fmap` or `hmap`, or `lut` (for Virtex, Virtex-E, and Virtex2/Virtex2p).
2. Instantiate these design units at a higher level.
3. Use the `xc_uset` attribute to group a number of instances together.
4. Use the `xc_rloc` attribute to specify the relative locations of all instances with the same `xc_uset` value.
5. Create a top-level design unit instantiating your design.

See `xc_map` on page 7-125, `xc_rloc` on page 7-134, and `xc_uset` on page 7-137 for syntax details.

Controlling Port Names in EDIF

You can control the appearance of port names in the EDIF output files generated for your designs. This is especially useful in controlling the appearance of port names for black-box components (usually IP components) that you instantiate. By default, the character case of port names is preserved as it appears in the HDL source code.

Two attributes are used, depending on whether you want to control the names of scalar (individual) ports or bus ports. Specify the attributes on individual components in your HDL source code, or globally by using constraint files and the SCOPE spreadsheet.
Scalar Ports

For scalar port names, the syn_edif_scalar_format attribute determines if port names are all uppercase or all lowercase (for example, AbCd becomes ABCD).

Bus Ports

For bus port names, the syn_edif_bit_format attribute determines:

- If names are all uppercase or all lowercase (for example, AbCd becomes ABCD)
- If additional characters are appended to the end of names (for example, qout becomes qout_IP)
- If the bus range is appended to the end of names (for example, qout becomes qout[31:1])
- The appearance of the range delimiter when a range is appended (for example, qout<31:0> or qout[31:0])
Using Spartan

Device Mapping Options (Spartan)

Device mapping options are optimization algorithms that you can enable during synthesis. You can set the device mapping options for the design in the UI, through Project -> Implementation Options -> Device (or you can use Impl Options or New Impl buttons to display this dialog box).

- Fanout Guide – sets the fanout limit. Default is 100. See Fanout Guide on page H-2 for details.
- Disable I/O Insertion – prevents I/O insertion during synthesis when you click on this option. Default is false (I/O insertion mode). See Disable I/O Insertion on page H-3 for details.
- Force GSR Usage – enables the forced use of the global set/reset routing resources during synthesis. See Force GSR Usage on page H-4 for details. This option is not available for Spartan2 designs.

You can also use the set_option Tcl command to enable/disable these options. See set_option Command for Spartan below for more information.

set_option Command for Spartan

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles.

Through the set_option Tcl command you specify the same implementation options as you do through the dialog box displayed in the Project view with Project -> Implementation Options

This section provides information on some of the specific options for the Spartan architectures. For a complete list of options, see set_option on page 4-15.

The set_option command has the following options for Spartan technologies:
Table H-3: Spartan set_option Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -technology *keyword* | Sets the target technology for the implementation.  
*keyword* can be SPARTAN, SPARTAN2E, SPARTAN-XL, or SPARTAN2. |
| -part *part_name* | Specifies a part for the implementation. Refer to the  
Project -> Implementation Options dialog box for the part choices. |
| -speed_grade *value* | Sets the speed grade for the implementation. Refer to the  
Project -> Implementation Options dialog box for available speed grade choices. |
| -package *package_name* | Specifies the package for the implementation. Refer to the  
Project -> Implementation Options dialog box for available package choices. |
| -maxfan *value* | Sets the fanout limit. Default is 100. |
| -force_gsr 1 | 0 | Enables/disables the forced use of the global set/reset routing resources during synthesis (only available for Spartan and Spartan-XL). The default is true. |
| -pipe 1 | 0 | Allows you to run designs at a faster frequency by  
moving registers after the multiplier into the multiplier.  
This option is only available for Spartan2 and Spartan2E. |
| -block 1 | 0 | Prevents I/O insertion during synthesis. The default is  
false, (I/O insertion mode). To disable I/O insertion, set  
this option to true. |
| -retiming 1 | 0 | Determines whether registers are moved into  
combinatorial logic to improve performance. You must  
also set -pipe. The -retiming option is only available for  
Spartan2 and Spartan2E. |

**project Command for Spartan**

This section provides information on some of the options for the project Tcl command for Spartan. See *project on page 4-12* for syntax. .

The project command has the following options specific to Spartan technologies:
Table H-4: Spartan project Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-result_format &quot;format&quot;</td>
<td>Changes the synthesis result file format for the implementation which contains the synthesized netlist created by the Synplify synthesis tool. The format must be in lower-case letters: edif or xnf (Spartan and Spartan-XL only).</td>
</tr>
<tr>
<td>-result_file value</td>
<td>Specifies the name of the synthesized netlist for the implementation. If you use this option, it should precede the synthesis result file specification using project -result_file at the end of the Tcl file, and before any project -run or project -save commands. The result file format can also be set in the UI through Project -&gt; Implementation Options (Implementation Results panel).</td>
</tr>
</tbody>
</table>
Using Virtex

Device Mapping Options (Virtex)

Device mapping options are optimization algorithms that you can enable during synthesis. You can set the device mapping options for the design in the UI, through Project -> Implementation Options -> Device (or you can use Impl Options or New Impl buttons to display this dialog box).

- Fanout Guide – sets the fanout limit. Default is 100. See Fanout Guide on page H-2 for details.

- Disable I/O Insertion – prevents I/O insertion during synthesis when you click on this option. Default is false (I/O insertion mode). See Disable I/O Insertion on page H-3 for details.

You can also use the `set_option` Tcl command to enable/disable these options. See set_option Command for Virtex on page H-30 for more information.

set_option Command for Virtex

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles.

Through the `set_option` Tcl command you specify the same implementation options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on some of the specific options for the Virtex architectures. See set_option on page 4-15 for the syntax.

The `set_option` command has the following options for Virtex, Virtex-E, Virtex2, and Virtex2p:
Table H-5: Virtex set_option Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology <em>keyword</em></td>
<td>Sets the target technology for the implementation. <em>keyword</em> can be <em>virtex</em>, <em>virtex-e</em> or <em>virtex2</em>.</td>
</tr>
<tr>
<td>-part <em>part_name</em></td>
<td>Specifies a part for the implementation. Refer to the Project -&gt; Implementation Options form for available part choices.</td>
</tr>
<tr>
<td>-speed_grade <em>value</em></td>
<td>Sets the speed grade for the implementation. Refer to the Project -&gt; Implementation Options form for available speed grade choices.</td>
</tr>
<tr>
<td>-package <em>package_name</em></td>
<td>Specifies the package for the implementation. Refer to the Project -&gt; Implementation Options form for available package choices.</td>
</tr>
<tr>
<td>-maxfan <em>value</em></td>
<td>Sets the fanout limit. Default is 100.</td>
</tr>
<tr>
<td>-block 1</td>
<td>Prevents I/O insertion during synthesis. The default is false, (I/O insertion mode). To disable I/O insertion, set this option to 1.</td>
</tr>
</tbody>
</table>

Table H-6: Virtex project Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-result_format &quot;<em>format</em>&quot;</td>
<td>Changes the synthesis result file format for the implementation which contains the synthesized netlist created by the Synplify synthesis tool. The format must be in lower-case letters: <em>edif</em>.</td>
</tr>
<tr>
<td>-result_file <em>value</em></td>
<td>Specifies the name of the synthesized netlist for the implementation.</td>
</tr>
</tbody>
</table>
Resource Usage

The Synplify synthesis tool generates a resource usage report in the log file. To view it, click the View Log button in the Project window.

Resource usage reports for Xilinx include the following information:

- Xilinx-specific components and the number of times they are used
- I/O primitives
- Global buffers
- Total number of LUTs
- Memory cells used

Interfacing with Xilinx Place-and-route Tools

When you invoke the Xilinx ISE Project Navigator or Design Manager place-and-route tool directly from the Synplify synthesis tool, the corresponding Xilinx project is created automatically.

To start the place-and-route tool from the synthesis tool, do the following:

1. Create a new project or open an existing project in the synthesis tool, select a Xilinx technology, and add the project source files,

2. Before you run synthesis, review the option settings for your design in the Project -> Implementation Options dialog box.

3. Synthesize your design with the synthesis tool. After successful synthesis, choose Options -> Xilinx -> Start ISE Project Navigator or Options -> Xilinx -> Start Design Manager to start the place-and-route tool.

   This displays the place-and-route tool user interface, places the synthesis-generated netlist in a Xilinx project, and names that project.

4. Configure your Xilinx project settings in the place-and-route tool.

5. Run the Xilinx place-and-route tool.

For more information on Xilinx place-and-route tools, refer to the appropriate Xilinx documentation.
**Guidelines for NGDBuild**

When reading the Synplify synthesis tool netlist files into the Xilinx place-and-route tool, define the timing constraints that are not forward-annotated by the synthesis tool in a user constraint file (.ucf). From the command line, read the .ucf file into Xilinx place-and-route NGDBuild with the -uc command.

Refer to the Xilinx documentation for more information on setting timing constraints.

**Guidelines for MAP**

Do not map to 5-input functions. Do not use the -k command line option.

**Guidelines for PAR**

Do not use the default effort level of 5. Instead, set it to 4 with the -l 4 command line option.
Appendix H: Designing with Xilinx

Using XC4000

Device Mapping Options (XC4000)

Device mapping options are optimization algorithms that you can enable during synthesis. You can set the device mapping options for the design in the UI, through Project -> Implementation Options -> Device (or you can use Impl Options or New Impl buttons to display this dialog box). See Device Mapping Options (XC3000 and XC5200) on page H-37 for explanations of the options for other XC families.

- Disable I/O Insertion – prevents I/O insertion during synthesis when you click on this option. Default is false (I/O insertion mode). See Disable I/O Insertion on page H-3 for details.
- Force GSR Usage – enables the forced use of the global set/reset routing resources during synthesis. See Force GSR Usage on page H-4 for details.

set_option Command for XC4000

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles.

Through the set_option Tcl command you specify the same implementation options as you do through the dialog box displayed in the Project view with Project -> Implementation Options

This section provides information on some of the specific options for the XC architectures. For the complete set of options, see set_option on page 4-15.

The set_option command has the following options for XC4000 technologies:
Table H-7: XC4000 set_option Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology <em>keyword</em></td>
<td>Sets the target technology for the implementation. <em>keyword</em> can be XC4000, XC4000E, XC4000EX, XC4000XL, XC4000XLA, XC4000L, or XC4000XV.</td>
</tr>
<tr>
<td>-part <em>part_name</em></td>
<td>Specifies a part for the implementation. Refer to the Project -&gt; Implementation Options form for available part choices.</td>
</tr>
<tr>
<td>-speed_grade <em>value</em></td>
<td>Sets the speed grade for the implementation. Refer to the Xilinx databook for available speed grade choices.</td>
</tr>
<tr>
<td>-package <em>package_name</em></td>
<td>Specifies the package for the implementation. Refer to the Project -&gt; Implementation Options for available package choices.</td>
</tr>
<tr>
<td>-maxfan <em>value</em></td>
<td>Sets the fanout limit. Default is 100.</td>
</tr>
<tr>
<td>-force_gsr 1</td>
<td>Enables/disables the forced use of the global set/reset routing resources during synthesis. The default is true.</td>
</tr>
<tr>
<td>-block 1</td>
<td>Prevents I/O insertion during synthesis. The default is false, (I/O insertion mode). To disable I/O insertion, set this option to true.</td>
</tr>
</tbody>
</table>

**project Command for XC4000**

This section provides information on some of the options for the project Tcl command for the XC4000 technologies. For the complete set of options, see *project* on page 4-12.
The project command has the following options specific to these technologies:

Table H-8: XC4000 project Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>result_format &quot;format&quot;</code></td>
<td>Changes the synthesis result file format for the implementation which contains the synthesized netlist created by the Synplify synthesis tool. The format must be in lower-case letters: edif or xnf.</td>
</tr>
<tr>
<td><code>result_file value</code></td>
<td>Specifies the name of the synthesized netlist for the implementation.</td>
</tr>
</tbody>
</table>

If you use this option, it should immediately precede the synthesis result file specification using `project -result_file` at the end of the Tcl file, and before any `project -run` or `project -save` commands.

The result file format can also be set from the UI in the Results Format field through Project -> Implementation Options -> Implementation Results.
Device Mapping Options (XC3000 and XC5200)

Device mapping options are optimization algorithms that you can enable during synthesis. You can set the device mapping options for the design in the UI, through Project -> Implementation Options -> Device (or you can use Impl Options or New Impl buttons to display this dialog box).

- Fanout Guide – sets the fanout limit. Default is 100. See Fanout Guide on page H-2 for details.
- Disable I/O Insertion – prevents I/O insertion during synthesis when you click on this option. Default is false (I/O insertion mode). See Disable I/O Insertion on page H-3 for details.
- Force GSR Usage – enables the forced use of the global set/reset routing resources during synthesis (XC3000 and XC5200 only). See Force GSR Usage on page H-4 for details.
- Use Dedicated Reset Pin – enables the use of the dedicated reset pin (XC3000 only).

You can also use the set_option Tcl command to enable/disable these options. See set_option Command for Spartan below for more information.

set_option Command for XC3000 and XC5200

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles.

Through the set_option Tcl command you specify the same implementation options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on some of the specific options for the XC3000 and XC5200 architectures. For a complete list of options, see set_option on page 4-15.

The set_option command has the following options for these technologies:
Table H-9: XC3000 and XC5200 set_option Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-technology <em>keyword</em></td>
<td>Sets the target technology for the implementation. <em>keyword</em> can be XC3000 or XC5200.</td>
</tr>
<tr>
<td>-part <em>part_name</em></td>
<td>Specifies a part for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available part choices.</td>
</tr>
<tr>
<td>-speed_grade <em>value</em></td>
<td>Sets the speed grade for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available speed grade choices.</td>
</tr>
<tr>
<td>-package <em>package_name</em></td>
<td>Specifies the package for the implementation. Refer to the Project -&gt; Implementation Options dialog box for available package choices.</td>
</tr>
<tr>
<td>-maxfan <em>value</em></td>
<td>Sets the fanout limit. Default is 100.</td>
</tr>
<tr>
<td>-force_gsr 1</td>
<td>0</td>
</tr>
<tr>
<td>-userstpin 1</td>
<td>0</td>
</tr>
<tr>
<td>-block 1</td>
<td>0</td>
</tr>
</tbody>
</table>

**project Command for XC3000 and XC5200**

This section provides information on specific options for the project Tcl command for these technologies. See *project on page 4-12* for a complete list of options.

The project command has the following option specific to these technologies:
Table H-10: XC3000 and XC5200 project Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-result_format</td>
<td>Changes the synthesis result file format for the implementation which contains the synthesized netlist created by the Synplify synthesis tool. The format must be in lower-case letters: edif or xnf.</td>
</tr>
<tr>
<td>-result_file</td>
<td>Specifies the name of the synthesized netlist for the implementation. If you use this option, it should precede the synthesis result file specification using project -result_file at the end of the Tcl file, and before any project -run or project -save commands. The result file format can also be set in the UI through Project -&gt; Implementation Options (Implementation Results panel).</td>
</tr>
</tbody>
</table>
Using Xilinx CLPDs

Device Mapping Options (CPLDs)

Device mapping options are optimization algorithms that you can enable during synthesis. You can set the device mapping options for the design in the UI, through Project -> Implementation Options -> Device (or you can use Impl Options or New Impl buttons to display this dialog box). Xilinx CPLDs include the XC9500, CoolRunner and CoolRunner-II CPLDs technologics. The device mapping options for these technologies are the following:

- Fanout Guide – sets the fanout limit. Default is 100. See Fanout Guide on page H-2 for details.
- Disable I/O Insertion – prevents I/O insertion during synthesis when you click on this option. Default is false (I/O insertion mode). See Disable I/O Insertion on page H-3 for details.

You can also use the set_option Tcl command to enable/disable these options. See set_option Command for Spartan below for more information.

set_option Command for Xilinx CPLDs

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles.

Through the set_option Tcl command you specify the same implementation options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

This section provides information on some of the specific options for the Xilinx CPLD architectures (XC9500, CoolRunner and CoolRunner-II CPLDs). For a complete list of options, see set_option on page 4-15.

The set_option command has the following options for these technologies:
This section provides information on specific options for the project Tcl command for the Xilinx CPLD architectures (XC9500, CoolRunner and CoolRunner-II CPLDs). See project on page 4-12 for a complete list of options.
The project command has the following option specific to these technologies:

Table H-12: Xilinx CPLDs project Tcl command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-result_format &quot;format&quot;</code></td>
<td>Changes the synthesis result file format for the implementation which contains the synthesized netlist created by the Synplify synthesis tool. The format must be in lower-case letters: edif or xnf.</td>
</tr>
<tr>
<td><code>-result_file value</code></td>
<td>Specifies the name of the synthesized netlist for the implementation. If you use this option, it should precede the synthesis result file specification using project -result_file at the end of the Tcl file, and before any project -run or project -save commands. The result file format can also be set in the UI through Project -&gt; Implementation Options (Implementation Results panel).</td>
</tr>
</tbody>
</table>
Xilinx Attribute and Directive Summary

The following table summarizes the synthesis and Xilinx-specific attributes and directives available with the Xilinx Technology. Complete descriptions and examples can be found in Chapter 7, Synthesis Attributes and Directives.

Table H-13: Synthesis and Xilinx attributes and directives

<table>
<thead>
<tr>
<th>Attribute / Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>black_box_pad_pin (D)</td>
<td>Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>black_box_tri_pins (D)</td>
<td>Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.</td>
</tr>
<tr>
<td>full_case (D)</td>
<td>Specifies that a Verilog case statement has covered all possible cases.</td>
</tr>
<tr>
<td>parallel_case (D)</td>
<td>Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.</td>
</tr>
<tr>
<td>syn_black_box (D)</td>
<td>Defines a black box for synthesis.</td>
</tr>
<tr>
<td>syn_direct_enable</td>
<td>Identifies which signal is to be used as the enable input to an enable flip-flop when multiple candidates are possible.</td>
</tr>
<tr>
<td>syn_edif_bit_format</td>
<td>Controls the character formatting and style of bus signal names, port names, and vector ranges in the EDIF output file.</td>
</tr>
<tr>
<td>syn_edif_scalar_format</td>
<td>Controls the character formatting of scalar signal and port names in the EDIF output file.</td>
</tr>
<tr>
<td>syn_encoding</td>
<td>Specifies the encoding style for state machines.</td>
</tr>
<tr>
<td>syn_enum_encoding (D)</td>
<td>Specifies the encoding style for enumerated types (VHDL only).</td>
</tr>
<tr>
<td>syn_forward_io_constraints</td>
<td>Enables I/O constraints to be forward-annotated to the Xilinx place-and-route tools.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
### Table H-13: Synthesis and Xilinx attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute / Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn_hier</td>
<td>Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.</td>
</tr>
<tr>
<td>syn_isclock (D)</td>
<td>Specifies that a black-box input port is a clock, even if the name does not indicate it is one.</td>
</tr>
<tr>
<td>syn_keep(D)</td>
<td>Prevents the internal signal from being removed during synthesis and optimization.</td>
</tr>
<tr>
<td>syn_maxfan</td>
<td>Sets a fanout limit for an individual input port or register output.</td>
</tr>
<tr>
<td>syn_multstyle</td>
<td>Determines implementation style for multipliers in Virtex2/Virtex2p designs.</td>
</tr>
<tr>
<td>syn_netlist_hierarchy</td>
<td>Determines if the EDIF output netlist is flat or hierarchical.</td>
</tr>
<tr>
<td>syn_noarrayports</td>
<td>Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.</td>
</tr>
<tr>
<td>syn_noclockbuf</td>
<td>Controls the automatic insertion of global clock buffers.</td>
</tr>
<tr>
<td>syn_noprune (D)</td>
<td>Controls the automatic removal of instances that have outputs that are not driven.</td>
</tr>
<tr>
<td>syn_preserve (D)</td>
<td>Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.</td>
</tr>
<tr>
<td>syn_ramstyle</td>
<td>Determines the way in which RAMs are implemented.</td>
</tr>
<tr>
<td>syn_reference_clock</td>
<td>Specifies a clock frequency other than that implied by the signal on the clock pin of the register.</td>
</tr>
<tr>
<td>syn_replicate</td>
<td>Disables replication.</td>
</tr>
<tr>
<td>syn_romstyle (Xilinx)</td>
<td>Determines how ROM architectures are implemented.</td>
</tr>
<tr>
<td>syn_sharing (D)</td>
<td>Specifies resource sharing of operators.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
### Table H-13: Synthesis and Xilinx attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute / Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>syn_srlstyle</code></td>
<td>Determines how to implement the sequential shift (seqShift) components. This attribute applies to Virtex families.</td>
</tr>
<tr>
<td><code>syn_state_machine</code> (D)</td>
<td>Determines if the FSM Compiler extracts a structure as a state machine.</td>
</tr>
<tr>
<td><code>syn_tco&lt;n&gt;</code> (D)</td>
<td>Defines timing clock to output delay through the black box. The <code>n</code> indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tpd&lt;n&gt;</code> (D)</td>
<td>Specifies timing propagation for combinational delay through the black box. The <code>n</code> indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_tristate</code> (D)</td>
<td>Specifies that a black-box pin is a tristate pin.</td>
</tr>
<tr>
<td><code>syn_tristatetomux</code></td>
<td>Converts tristate drivers that drive nets below a certain limit to muxes. This only applies to the XC4000 and Virtex families.</td>
</tr>
<tr>
<td><code>syn_tsu&lt;n&gt;</code> (D)</td>
<td>Specifies the timing setup delay for input pins, relative to the clock. The <code>n</code> indicates a value between 1 and 10.</td>
</tr>
<tr>
<td><code>syn_useenables</code></td>
<td>Prevents generation of registers with clock enable pins.</td>
</tr>
<tr>
<td><code>syn_useioff</code> (Xilinx)</td>
<td>Packs flip-flops in the I/Os to improve input/output path timing.</td>
</tr>
<tr>
<td><code>translate_off/translate_on</code> (D)</td>
<td>Specifies sections of code to be excluded from synthesis, such as simulation-specific code.</td>
</tr>
<tr>
<td><code>xc_alias</code></td>
<td>Changes the cell name in XNF for all families except Virtex families.</td>
</tr>
<tr>
<td><code>xc_clockbuftype</code></td>
<td>Specifies that a clock port use the Clock Delay Locked Loop primitive, CLKDLL, in Virtex designs.</td>
</tr>
<tr>
<td><code>xc_fast</code></td>
<td>Speeds up the transition time of the output driver in XC4000 designs.</td>
</tr>
<tr>
<td><code>xc_fast_auto</code></td>
<td>Controls the instantiation of fast output buffers in Virtex designs.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
### Table H-13: Synthesis and Xilinx attributes and directives (Continued)

<table>
<thead>
<tr>
<th>Attribute / Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xc_globalBuffers</td>
<td>Controls the number of global buffers to use in a design.</td>
</tr>
<tr>
<td>xc_isgsr</td>
<td>Specifies that a black-box port is connected to an internal STARTUP block in non-Virtex designs, and prevents the Synplify synthesis tool from inferring a STARTUP block.</td>
</tr>
<tr>
<td>xc_loc</td>
<td>Specifies the placement of ports and design units.</td>
</tr>
<tr>
<td>xc_map</td>
<td>Specifies the Xilinx fmap or hmap primitive in XC4000, and the LUT in Virtex designs.</td>
</tr>
<tr>
<td>xc_ncf_auto_relax</td>
<td>Controls the automatic relaxation of constraints that are forward-annotated to the .ncf file.</td>
</tr>
<tr>
<td>xc_nodelay</td>
<td>Controls the Xilinx insertion of input delay for flip-flops and latches in the XC4000 family.</td>
</tr>
<tr>
<td>xc_padtype</td>
<td>Specifies an I/O buffer standard in Virtex designs.</td>
</tr>
<tr>
<td>xc_props</td>
<td>Specifies Xilinx attributes to forward to the gate-level netlist.</td>
</tr>
<tr>
<td>xc_pullup / xc_pulldown</td>
<td>Specifies that a port is a pull-up or pull-down port.</td>
</tr>
<tr>
<td>xc_rloc</td>
<td>Specifies the relative locations of all instances with the same xc_uset attribute value. This attribute applies to XC4000 and Virtex families.</td>
</tr>
<tr>
<td>xc_slow</td>
<td>Specifies a slow transition time for the driver of an output port.</td>
</tr>
<tr>
<td>xc_uset</td>
<td>Assigns a group name to component instances.</td>
</tr>
</tbody>
</table>

(D) indicates directives; all others are attributes.
Index

Symbols
&f header and footer variable 3-15
&p header and footer variable 3-15
... action button
   Editor Options dialog box 3-57
   New dialog box 3-8
   .fse file 1-30
   .info file 1-30, 8-38
   .ini file 2-14, 5-7
   .prj file 1-23
   .scp command file F-9
   .sdc file 1-25, 6-2, 6-20, 8-47
      attribute syntax 7-11
   .srm file 1-27
   .srr file 1-27, 4-23
   .srs file 1-27
   .vhm file 1-26
   .vma file 1-26
< action button
   Object Query dialog box 3-50
< Add action button
   Select Files to Add to Project dialog box 3-40
   Select Projects to Include in Workspace dialog box 3-82
< Add All action button
   Select Files to Add to Project dialog box 3-40
   Select Projects to Include in Workspace dialog box 3-82
< All action button
   Object Query dialog box 3-50
-> action button
   Object Query dialog box 3-50
   --include command option 4-9
   %\#c header and footer variable 3-15
   %\#x header and footer variable 3-15
   %c header and footer variable 3-15
   %x header and footer variable 3-15
   %o header and footer variable 3-15

Numerics
3D-Buttons check box, Display Settings dialog box 3-28

A
aborting a synthesis run 3-45
About this program menu item
   Help menu 3-65
Absolute Path radio button
   File Options dialog box 3-78
accessing commands 3-2
accessing LMF, Max+Plus II B-35
accessing packages, VHDL 9-13
ACEX set_option Tcl command arguments
   (table) B-27
Actel A-1
   ACTgen macros (Verilog) A-6
   ACTgen macros (VHDL) A-7
   alsloc attribute 7-19
   alspin attribute 7-21
   alspreserve attribute 7-23
   assigning I/O pin numbers to ports 7-21
   attributes A-10
device mapping options A-3
directives A-10
fanout limits A-3
features A-2
I/O insertion A-4
instantiating black boxes A-5
instantiating macros A-5
macro libraries examples A-7
preserving relative placement 7-19
product families A-2
reports A-3
set_option Tcl command A-8
syn_preserve_sr_priority attribute 7-81
Actel (continued)
  syn_radhardlevel attribute 7-85
Tcl commands A-8
  using macro libraries (Verilog) A-6
  using macro libraries (VHDL) A-6
Actel set_option command (table) A-8
action buttons 3-2
... 3-8, 3-57
<- 3-50
<- All 3-50
<- Add
  Select Files to Add to Project dialog box 3-40
  Select Projects to Include in Workspace dialog box 3-82
<- Add All
  Select Files to Add to Project dialog box 3-40
  Select Projects to Include in Workspace dialog box 3-82
-> 3-50
Add icon 3-70
Add Row 3-87
All -> 3-50
Cancel Job 3-45
Change 3-40, 3-61
Clear All Bookmarks (toolbar icon) 3-92
Clipboard 3-70
closed book icon (online help) 3-66
Customize 3-25
Cut (toolbar icon) 3-89
Delete 3-25
Delete icon 3-70
document icon (online help) 3-66
Existing Project 3-12
Filter on Selected Gates (toolbar icon) 3-90
Find (toolbar icon) 3-89
Find Next
  Find dialog box 3-20
  Replace dialog box 3-21
Mark All 3-20
Move Down icon 3-70
Move Up icon 3-70
New 3-25
New Constraint (SCOPE) (toolbar icon) 3-89
New HDL file (toolbar icon) 3-89
New Project 3-12
New Workspace 3-12
Next Bookmark (toolbar icon) 3-92
Next Sheet (toolbar icon) 3-91
Next Tip 3-72
Open 3-10
Open (toolbar icon) 3-89
open book icon (online help) 3-66
Open project (toolbar icon) 3-89
Options 3-57
Paste (toolbar icon) 3-89
Previous Bookmark (toolbar icon) 3-92
Previous Sheet (toolbar icon) 3-91
Previous Tip 3-72
Print (toolbar icon) 3-89
Project view 3-97
Project Wizard 3-12
Push/Pop Hierarchy (toolbar icon) 3-91
Redo 3-87
Redo (toolbar icon) 3-89
Remove All ->
  Select Files to Add to Project dialog box 3-40
  Select Projects to Include in Workspace dialog box 3-82
Remove Row 3-87
Remove ->
  Select Files to Add to Project dialog box 3-40
  Select Projects to Include in Workspace dialog box 3-82
Replace 3-21
Replace All 3-21
Reset 3-25
RTL View (toolbar icon) 3-90
Save 3-11, 3-71
Save (toolbar icon) 3-89
Save all (toolbar icon) 3-89
Show Critical Path (toolbar icon) 3-90
Technology View (toolbar icon) 3-90
Text File 3-70
toggle Bookmark (toolbar icon) 3-92
Undo 3-87
Undo (toolbar icon) 3-89
Validate 3-71
Zoom 100% (toolbar icon) 3-91
Zoom Full (toolbar icon) 3-91
Zoom In (toolbar icon) 3-91
Zoom Out (toolbar icon) 3-91
-active command option 4-13
Add icon
License Editor dialog box 3-70
Add Row action button
  SCOPE Attributes dialog box 3-87
Add Source File menu item 3-82
  Project menu 3-30
  Project View popup menu 3-76
Add to Project dialog box zone
  New dialog box 3-8
add_file Tcl command 4-3, 4-4, 4-6, 4-9, 6-20
  examples of use 4-10
Additional Products menu item
Help menu 3-65
Adjusting RGB values (table) 2-14
   Adjusting RGB values in HDL Analyst
  (table) 5-7
Advanced submenu
   Edit menu 3-17
Align menu item
   Format menu 3-51
All -> action button
   Object Query dialog box 3-50
   -all command option 4-11
alsloc attribute (Actel) 7-19
  examples
   sdc file 7-19
   Verilog 7-19
   VHDL 7-20
alspin attribute (Actel) 7-21
  examples
   sdc file 7-21
   Verilog 7-21
   VHDL 7-22
alspreserve attribute (Actel) 7-23
  examples
   sdc file 7-23
   Verilog 7-23
   VHDL 7-23
Alt keyboard shortcut (Text Editor) 2-9, 3-61
Altera B-1
ACEX
   device mapping options B-26
   file format options B-27
   project Tcl command B-28
   set_option Tcl command B-26, B-27
altera_auto_use_eab attribute 7-25
altera_auto_use_esb attribute 7-28
altera_chip_pin_lc attribute 7-31
altera_implement_in_eab attribute 7-33
altera_implement_in_esb attribute 7-35
altera_io_opendrain attribute 7-37
altera_io_powerup attribute 7-39
APEX
   device mapping options B-14
   file format options B-16
   project Tcl command B-16
   set_option Tcl command B-15
APEX II
   device mapping options B-14
   file format options B-16
   project Tcl command B-16
   set_option Tcl command B-15
attributes summary B-36
cliquing B-15
clock enable net assignment 7-53
   constraints B-3
device mapping options
   ACEX B-26
   APEX B-14
   APEX II B-14
   Excalibur B-14
   FLEX B-26
   MAX B-32
   Mercury B-14
   Stratix B-20
device support B-2
directives summary B-36
EDIF and VQM filenames B-9
Excalibur
   device mapping options B-14
   file format options B-16
   project Tcl command B-16
   set_option Tcl command B-15
fanout limits B-17
file format options
   ACEX B-27
   APEX B-16
   APEX II B-16
   Excalibur B-16
   FLEX B-27
   MAX B-33
   Mercury B-16
   Stratix B-21
FLEX
   device mapping options B-26
   file format options B-27
   project Tcl command B-28
   set_option Tcl command B-26
Altera (continued)
forward annotating
  to Max+Plus II 6-39, B-28
  to Quartus II 6-39
forward annotation B-34
  ACEX B-28
  APEX B-17
  APEX II B-17
  Excalibur B-17
  FLEX B-28
  Mercury B-17
  Stratix B-17
I/O insertion B-15
I/O registers power-up mode 7-39
implementation options B-10
inerring
  RAM B-4
  registers B-9
  ROM B-6
instantiating
  LPMS
    in Verilog B-4, B-41
    in VHDL B-4, B-46
  macros/black boxes
    in Verilog B-3
    in VHDL B-3
  special buffers
    in Verilog B-43
    in VHDL B-49
mapping EABs B-40, B-45
mapping to ATOMs B-10
  APEX B-15
  APEX II, Excalibur, Mercury B-15
MAX
  device mapping options B-32
  file format options B-33
  project Tcl command B-34
  set_option Tcl command B-33
Max+Plus II, forward-annotating to 6-39, B-28
Mercury
  device mapping options B-14
  file format options B-16
  project Tcl command B-16
  set_option Tcl command B-15
open-drain mode 7-37

packing registers into I/O cells B-17
pin locations 7-31
project Tcl command
  ACEX B-28
  APEX B-16
  APEX II B-16
  Excalibur B-16
  FLEX B-28
  MAX B-34
  Mercury B-16
  Stratix B-21
Quartus II, forward-annotating to 6-39
RAM inference B-4
reports B-11
ROM inference B-6
simulation (post-synthesis) in
  APEX B-18
specifying I/O locations B-39, B-44
Stratix
  device mapping options B-20
  DSP MAC block inference B-23
  file format options B-21
  I/O insertion B-20
  mapping to ATOMs B-20
  project Tcl command B-21
  RAM inference B-22
  set_option Tcl command B-20
syn_romstyle attribute 7-97
syn_useioff attribute 7-108
timing constraints 6-39
VHDL prepared components B-51
altera_auto_use_eab attribute (Altera) 7-25
  examples
    sdc file 7-25
    Verilog 7-25
    VHDL 7-26
altera_auto_use_esb attribute (Altera) 7-28
  examples
    sdc file 7-28
    Verilog 7-28
    VHDL 7-29
altera_chip_pin_lc attribute (Altera) 7-31
  examples
    sdc file 7-31
    Verilog 7-31
    VHDL 7-32
altera_implement_in_eab attribute (Altera) 7-33
examples
sdc file 7-33
Verilog 7-33
VHDL 7-34
altera_implement_in_esb attribute (Altera) 7-35
examples
sdc file 7-35
Verilog 7-35
VHDL 7-36
altera_io_opendrain attribute (Altera) 7-37
examples
sdc file 7-37
Verilog 7-37
VHDL 7-38
altera_io_powerup attribute (Altera) 7-39
examples
sdc file 7-39
Verilog 7-39
VHDL 7-40
always blocks
Verilog 8-8
combinational logic 8-16
flip-flops 8-20
level-sensitive latches 8-22
always-enabled write enable
VHDL 9-78
amount of logic displayed, changing 5-7
Amplify Physical Optimizer tool
overview 1-5
Analyst toolbar 3-90
Filter on Selected Gates icon 3-90
RTL View icon 3-90
Show Critical Path icon 3-90
Technology View icon 3-90
Analyst toolbar (figure) 3-90
Analyst toolbar icons (table) 3-90
analyzing critical paths in HDL Analyst 5-18
APEX II menu options (table) B-15
APEX II project Tcl command arguments (table) B-17
APEX II set_option Tcl command arguments (table) B-15
APEX menu options (table) B-15
APEX project Tcl command arguments (table) B-17
APEX set_option Tcl command arguments (table) B-15
area, optimizing for 1-19
-area_delay_percent command option 4-16, B-33
-area_delay command option 4-16, E-14
Argument column
SCOPE spreadsheet 6-17
arithmetic operators
Verilog 8-2
VHDL 9-15
Arrange Icons menu item
Window menu 3-62
arranging windows 3-62
Assigning I/O locations with the SCOPE spreadsheet (figure) H-19
assignment operators
Verilog 8-2
VHDL 9-9
assignment statement
combinational logic (Verilog) 8-18
level-sensitive latches (Verilog) 8-21
VHDL 9-34
asynchronous sets and resets
Verilog 8-24
VHDL 9-44
asynchronous state machines
created with a process (VHDL) 9-55
that generate errors (VHDL) 9-55
Verilog 8-32, 8-33
VHDL 9-54
Atmel C-1
device mapping options C-2
I/O pad insertion C-2
product families C-2
set_option Tcl command C-4
Atmel attributes and directives (table) C-5
Atmel set_option command options (table) C-4
ATOM mapping, Altera B-10
Attribute column
SCOPE spreadsheet 7-10
Attribute dialog box zone
SCOPE Attributes dialog box 3-86
attributes 7-1
Actel A-10
alphabetical summary 7-3
alsloc (Actel) 7-19
alspin (Actel) 7-21
alspreserve (Actel) 7-23
Altera B-36
altera_auto_use_eab (Altera) 7-25
altera_auto_use_esb (Altera) 7-28
altera_chip_pin_lc (Altera) 7-31
altera_implement_in_eab 7-33
altera_implement_in_esb 7-35
altera_io_opendrain 7-37
altera_io_powerup 7-39
defining in source code 7-14
din (Lattice) 7-41
dout (Lattice) 7-42
individual descriptions 7-18
loc (Lattice) 7-43
lock (Lattice) 7-44
orca_padtype (Lattice) 7-45
orca_props (Lattice) 7-47
overview 7-2
ql_padtype (QuickLogic) 7-49
ql_placement (QuickLogic) 7-51
specifying 7-9
specifying in SCOPE spreadsheet 7-10
specifying in Verilog 7-15
specifying in VHDL 7-16
syn_direct_enable 7-53
syn_edif_bit_format 7-55
syn_edif_name_length 7-58
syn_edif_scalar_format 7-60
syn_encoding 7-62
syn_forward_io_constraints 7-65
syn_hier 7-67
syn_maxfan 4-25, 7-70, B-17, H-3
syn_multistyle 7-72
DSP MAC inference in Altera
Stratix B-25
syn_netlist_hierarchy 7-74
syn_noarrayports 7-75
syn_noclockbuf 4-25, 7-76
syn_pipeline 7-78
syn_preserve_sr_priority (Actel) 7-81
syn_props (Triscend) 7-83
syn_radhardlevel (Actel) 7-85
syn_ramstyle 7-87, 8-36
syn_reference_clock 6-35, 7-91
syn_replicate 7-92
syn_resources 7-95
syn_romstyle
Altera 7-97
Xilinx 7-100
syn_srlstyle 7-102
syn_tristatetomux 7-104
syn_useenable 7-106
syn_useioff
Altera 7-108
DSP MAC inference in Altera
Stratix B-25
Lattice ORCA 7-110
QuickLogic 7-112
Xilinx 7-114
syntax
.sdc file 7-11
Verilog 7-15
VHDL 7-16
Verilog 8-47
VHDL 9-85
VHDL library package 9-85
VHDL package 7-16
VHDL syntax 7-16
xc_alias (Xilinx) 7-116
xc_clockbuf (Xilinx) 7-117
xc_fast (Xilinx) 4-25, 7-119
xc_fast_auto (Xilinx) 7-121
xc_global_buffers (Xilinx) 7-123
xc_loc (Xilinx) 4-25, 7-124
xc_map (Xilinx) 7-125
xc_nvel_auto_relax (Xilinx) 6-43
xc_nvel_auto_relax (Xilinx) 6-43, 7-126
xc_nodelay (Xilinx) 7-128
xc_padtype (Xilinx) 7-129
xc_props (Xilinx) 7-131
xc_pulldown (Xilinx) 7-132
xc_pullup (Xilinx) 7-132
xc_rloc (Xilinx) 7-134
xc_slow (Xilinx) 7-136
xc_uset (Xilinx) 7-137
Xilinx H-5, H-43
Attributes (table) 7-3
attributes and directives
defining in source code 7-14
overview 7-2
Verilog 8-47
VHDL 9-85
Attributes and directives for Altera technol-
ology (table) B-36
Attributes panel
  Object column 7-9
  Object Type column 7-9
  SCOPE spreadsheet 6-17, 7-9
  attributes wizard 7-12
  -autosm command option 4-16

B
Background Compile menu item
  Options menu, Quartus submenu B-19
  -batch command option 1-13
  batch mode 1-13, 4-4, 4-6, 4-7
  Beep when a job completes check box
  Project View Options dialog box 3-56
  bit enumeration data type (VHDL) 9-6
  black boxes
    See also macros
    Actel A-5
    black_box_pad_pin 7-140
    black_box_tri_pins 7-142
    instantiating in Verilog 8-40
    instantiating in VHDL 9-86
    internal startup box (Xilinx) 7-189
    Lattice E-2
    syn_black_box directive 7-149
    syn_isclock 7-155
    syn_resources 7-95
    syn_tco directive 7-177
    syn_tpd directive 7-180
    syn_tristate directive 7-183
    syn_tsu directive 7-184
    timing constraints in VHDL 9-86
    using Verilog macro libraries (Actel) A-6
    using VHDL macro libraries (Actel) A-6
    Verilog 8-40
    VHDL 9-86
    Xilinx H-7
    black_box_pad_pin directive 7-140
      examples
        Verilog 7-140
        VHDL 7-141
    black_box_tri_pins directive 7-142
      examples
        Verilog 7-142
        VHDL 7-143
  black-box source code directives 6-36
  black-box timing constraints 6-36
  VHDL 9-86
  -block command
    option 4-16, B-16, B-21, B-27, C-4, E-6,
  bookmarks
    deleting 2-8
    inserting 2-8
    removing all 3-92
    using toolbar icons to manipulate 3-92
  boolean enumeration data type (VHDL) 9-6
  bottom-up synthesis 4-25
  browsers 5-12
    traversing hierarchy 5-12
  BUFG H-3
  Build Project menu item 3-11
    File menu 3-6
    Build Workspace dialog box 3-83
    Projects zone 3-83
    Build Workspace dialog box (figure) 3-83
    Build Workspace menu item 3-83
    Project View popup menu 3-74
    built-in gate primitives, Verilog 8-14
    bundle 6-36
    buttons
      See action buttons
  bypass logic 7-89

C
Cancel Job action button
  Job Status dialog box 3-45
  canceling a synthesis run 3-45
  Cascade menu item 3-63
    Window menu 3-62
    cascading windows 3-63
    Cascading windows display (figure) 3-63
    case statement (VHDL) 9-20
    Cells dialog box 3-52
    Cells dialog box (figure) 3-52
    Cells menu item 3-52
      Format menu 3-51
      Certify
        overview 1-6
      Certify SC
        overview 1-6
Check boxes

3D-Buttons 3-28
Beep when a job completes 3-56
Compress Buses 3-60
Cool Look 3-26
Enable 3-86
Enhanced Text Crossprobing 3-60
Horizontal Lines 3-28
Jump Location 3-50
Large Buttons 3-26
Mark Current Column 3-29
Mark Current Row 3-28
Match Case 3-20, 3-21
No Buses in Technology View 3-60
Project view 3-97
Push Tristates across Process/Block boundaries 3-37
Regular expression 3-20, 3-21
Resource Sharing 3-98, 7-172
Run 3-98
Show Cell Interior 3-60
Show Conn Name 3-60
Show Hierarchy Browser 3-60
Show Instance Name 3-60
Show Pin Name 3-60
Show Port Name 3-60
Show Project File Library 3-56
Show Project Library 2-13
Show Sheet Connector Index 3-60
Show Symbol Name 3-60
Show Text 3-60
Show Tips at Startup 3-72
Show Tooltips 3-26
Symbolic FSM Compiler 3-98
Verilog 2001 3-37, 3-78
Vertical Lines 3-28
View project files in folders 2-12, 3-56
Wrap around search 3-20, 3-21
Write Vendor Constraint File 6-39

Checking syntax 2-9
Checking synthesis 2-9
Clear All Bookmarks icon

Change action button
Schematic Options dialog box 3-61
Source File dialog box 3-40
Change File menu item 3-40
Project menu 3-30
Project View popup menu 3-75
Change Implementation Name menu item
Project View popup menu 3-78

Check boxes

3D-Buttons 3-28
Beep when a job completes 3-56
Compress Buses 3-60
Cool Look 3-26
Enable 3-86
Enhanced Text Crossprobing 3-60
Horizontal Lines 3-28
Jump Location 3-50
Large Buttons 3-26
Mark Current Column 3-29
Mark Current Row 3-28
Match Case 3-20, 3-21
No Buses in Technology View 3-60
Project view 3-97
Push Tristates across Process/Block boundaries 3-37
Regular expression 3-20, 3-21
Resource Sharing 3-98, 7-172
Run 3-98
Show Cell Interior 3-60
Show Conn Name 3-60
Show Hierarchy Browser 3-60
Show Instance Name 3-60
Show Pin Name 3-60
Show Port Name 3-60
Show Project File Library 3-56
Show Project Library 2-13
Show Sheet Connector Index 3-60
Show Symbol Name 3-60
Show Text 3-60
Show Tips at Startup 3-72
Show Tooltips 3-26
Symbolic FSM Compiler 3-98
Verilog 2001 3-37, 3-78
Vertical Lines 3-28
View project files in folders 2-12, 3-56
Wrap around search 3-20, 3-21
Write Vendor Constraint File 6-39

Checking syntax 2-9
Checking synthesis 2-9
Clear All Bookmarks icon
Clocks panel wizard
  SCOPE spreadsheet 6-19, 7-14
cloning a window 3-62
Close All menu item
  Window menu 3-62
 - close command option 4-13
Close menu item
  File menu 3-5
Close Project menu item
  File menu 3-6
  Project View popup menu 3-77
Close Workspace menu item
  Project View popup menu 3-77
closed book icon (online help) 3-66
color coding in Text Editor 2-8
color dialog box zone
  Display Settings dialog box 3-29
color preferences 2-14
colors
  changing in HDL Analyst 5-7
  defining for HDL Analyst 2-14
  setting in RTL and Technology views 2-14
combinational logic
  continuous assignments (Verilog) 8-18
Verilog 8-15, 8-18
  examples 8-15
VHDL 9-24
  examples 9-89
command access methods 3-2
Command column
  SCOPE spreadsheet 6-17
command line operation 4-4, 4-6, 4-7
command options
  -_include 4-9
  -active 4-13
  -all 4-11
  -area_delay_percent 4-16, B-33
  -areadelay 4-16, E-14
  -autosm 4-16
  -batch 1-13
  -clear 4-11
  -clique 4-16, B-16, B-27
  -cliquing 4-16, B-16
  -clockgroup 6-25
  -close 4-13
  -compiler_compatible 4-17
  -constraint 4-3, 4-9, 6-20
  -date 4-14
  -default 6-29, 6-31
  -default_enum_encoding 4-17
  -dir 4-13
  -disable 4-10, 6-25, 6-27, 6-28, 6-29, 6-31, 6-32, 6-34
  -disable_io_insertion 4-17, A-9, B-16, B-21
  -domap 4-17
  -duty_pct 6-26
  -enable 4-10
  -fall 6-26
  -fanin_limit 4-16, B-16, B-27, B-33, C-4, E-6, E-11, E-13, F-6, G-3, H-28, H-31, H-35, H-38, H-41
  -new 4-3, 4-12
  -no_time_gated_clock 4-18
  -num_critical_paths 4-18
  -num_startend_points 4-18
  -period 6-25
  -period 6-25

Synplify Reference Manual, April 2002

Index
command options (continued)
- -pipe H-28
- -ref 6-29, 6-31
- -remove 4-14
- -report_path 4-19, A-9
- -resource_sharing 4-19
- -result_file 4-13, 4-19, B-17, B-22, B-28, B-3
- -result_format 4-13, 4-19, B-17, B-28, E-14, H-29,
- -retiming H-28
- -rise 6-26
- -route 1-18, 6-26, 6-27, 6-28, 6-29, 6-30, 6-31
   define_clock 6-26
   define_input_delay 6-29
   define_output_delay 6-31
   define_reg_input_delay 6-27, 6-28
   define_reg_output_delay 6-28
- -run 4-3, 4-12
- -save 4-13
- -soft 4-19
- -soft_buffers 4-19, B-33
- -speed_grade 4-15, A-8, B-15, B-21, B-27, C-4,
   E-6, E-11, F-6, G-3, H-28, H-31, H-35,
   H-38, H-41
- -symbolic_fsm_compiler 4-16
- -technology 4-15, A-8, B-15, B-21, B-27, B-3
   C-4, D-11, E-6, E-11, E-13, F-6, G-3, H
- -through 6-32, 6-34, 7-157
- -time 4-14
- -to 6-32, 6-34
- -top_module 4-19
- -type 4-14
- -use_reset_pin 4-19
- -use_scan_for_test 4-19
- -userstin 4-19, H-38
- -verilog 4-3, 4-9
- -vhdl 4-3, 4-9
- -virtual 6-26
- -write_apr_constraint 4-19
- -write_verilog 4-19
- -write_vhdl 4-19
- -xilinx_m 4-19

commands
- File menu 3-5
- Format menu 3-51
- HDL Analyst menu 3-46

Help menu 3-65
- Options menu 3-54
- Project menu 3-30
- Project View popup menu 3-74, 3-85
- Run menu 3-43
- Tcl command equivalents 3-3
- Tcl constraint file 6-2
- View menu 3-22
- Window menu 3-62

Commands panel
- Customize dialog box 3-26
- Commands panel, Customize dialog box
  (figure) 3-27
- Comment Code menu item
  (edit menu, Advanced submenu) 3-17
- Comment column
  SCOPE spreadsheet 6-7, 6-11, 6-12, 6-13,
  6-15, 6-17, 7-10
- Comment dialog box zone
  SCOPE Attributes dialog box 3-87
- commenting out code (Text Editor) 2-9
- comments 7-10
- SCOPE spreadsheet
  Clocks panel 6-7
  False Paths panel 6-15
  Inputs/Outputs panel 6-11
  Multi-Cycle panel 6-13
  Other panel 6-17
  Registers panel 6-12
- syntax
  (Verilog) 8-10
- comments, syntax
  VHDL 9-34
- Common licensing tasks (table) 1-2
- Compile Only menu item
  Project View popup menu 3-76
  Run menu 3-43
- compiler
  inferring flip-flops with clock
  enables. 7-53
- -compiler_compatible command option 4-17
- compiling design units into libraries
  VHDL 9-11
- completion of keywords, Text Editor 2-8
- component information files 1-30
- component instantiation, VHDL 9-24
components
  creating resource VHDL library 9-15
  instantiating in VHDL 9-15, 9-24, 9-25
  specifying VHDL configurations 9-61
  using VHDL configuration declarations 9-64
  using VHDL vendor macro libraries 9-15
Compress Buses check box
  Schematic Options dialog box 3-60
concurrent signal assignments (VHDL) 9-21
configuration declaration (VHDL) 9-64
specification (VHDL) 9-60
Configure Verilog Compiler menu item 3-55
  Options menu 3-54
Configure VHDL Compiler menu item 3-54
  Options menu 3-54
configuring, Altera Max+Plus II (MAX) B-34
Connect to Visual Elite menu item
  HDL Analyst menu, Visual Elite Cross Probing submenu 3-48
constants
  declaring in VHDL 9-11
  designating using SNS in VHDL 9-27
VHDL 9-11
Constrainable objects (table) 6-16
constraint command option 4-3, 4-9, 6-20
constraint file Tcl commands 6-2
constraint files 1-25, 6-2, 6-20, 7-10, 8-47
  created manually 6-20
  creating 6-20
  manual creation 6-20
  writing guidelines 6-20
constraint types
  SCOPE spreadsheet 6-5
constraint wizards
  SCOPE spreadsheet 6-18
constraint_file Tcl command 4-10
  examples of use 4-11
constraint_file Tcl command options (table) 4-10
constraints
  Altera B-3
  forward-annotating 6-39
  Lattice ORCA E-8
  relaxing with xc_ncf_auto_relax attribute (Xilinx) 7-126
  timing 1-17, 6-2
  Xilinx H-5
contacting Synplicity customer support 1-4
Contents panel
  Help Topics dialog box 3-66
context-sensitive popup menus 3-2, 3-73
continuous assignment
  used for level-sensitive latches (Verilog) 8-21
  continuous assignments for combinational logic, Verilog 8-18
Cool Look check box
  Toolbars dialog box 3-26
CoolRunner, CoolRunner-II CPLDs (Xilinx) H-40
Copy File dialog box 3-79
  Implementation Name zone 3-79
  New File Name zone 3-79
  Original File Name zone 3-79
Copy File dialog box (figure) 3-79
Copy File menu item
  Project View popup menu 3-75
Copy menu item
  Edit menu 3-19
Copy Implementation menu item
  Project View popup menu 3-78
Copy menu item
  Edit menu 3-17, 3-18
Cover Cells menu item
  Format menu 3-51
CPLDs, Xilinx H-40
creating a hierarchical design
  Verilog 8-43
  VHDL 9-56
creating a scalable adder, Verilog example 8-12
creating a scalable design
  using generate statements (VHDL) 9-75
  using generics (VHDL) 9-72
  using unconstrained vector ports (VHDL) 9-71
VHDL 9-71
creating a timing report 6-46
creating an extra initialization state (VHDL) 9-53
creating constraint files 6-20
creating flip-flops and registers using VHDL processes 9-36
creating sequential logic
  Verilog 8-19
  creating Tcl scripts 4-3
critical path information, interpreting 5-19
Critical path reports (table) 6-50
critical paths
  analyzing in HDL Analyst 5-18
  HDL Analyst 5-18
  highlighting 5-18
  maximum number (Actel) A-5
  viewing 5-19
Cross Probing Enabled menu item
  HDL Analyst menu, Visual Elite Cross
  Probing submenu 3-48
crossprobing 5-8
  filtering text objects 3-61
  HDL Analyst 5-8
  ModelSim 5-11
  to trace component cause 5-11
  Visual Elite 5-8
Crossprobing with HDL Analyst views
  (table) 5-8
Ctrl keyboard shortcut (with mouse) 2-11
Ctrl-1 keyboard shortcut 3-93
Ctrl-a keyboard shortcut 3-93
Ctrl-c keyboard shortcut 3-93
Ctrl-f keyboard shortcut 3-94
Ctrl-F2 keyboard shortcut 3-93
  Text Editor 2-8
  Ctrl-F4 keyboard shortcut 2-13, 3-93
  Ctrl-F6 keyboard shortcut 2-13, 3-93
  Ctrl-g keyboard shortcut 3-94
  Ctrl-h keyboard shortcut 3-94
  Ctrl-I (Shift i) keyboard shortcut 3-95
  Ctrl-k keyboard shortcut 3-94, 5-11
  Ctrl-l keyboard shortcut 3-94
  Ctrl-N (Shift n) keyboard shortcut 3-95
  Ctrl-n keyboard shortcut 3-94, 5-15
  Ctrl-o keyboard shortcut 3-94
  Ctrl-P (Shift p) keyboard shortcut 3-95
  Ctrl-p keyboard shortcut 3-94, 5-16
  Ctrl-r keyboard shortcut 3-94
  Ctrl-s keyboard shortcut 3-94, 5-16
  Ctrl-Shift-F2 keyboard shortcut 3-95
  Text Editor 2-9
  Ctrl-Shift-i keyboard shortcut 3-95
  Ctrl-Shift-n keyboard shortcut 3-95
  Ctrl-Shift-p keyboard shortcut 3-95
Ctrl-v keyboard shortcut 3-94
Ctrl-x keyboard shortcut 3-94
Ctrl-y keyboard shortcut 3-95
Ctrl-z keyboard shortcut 3-95
customer support, contacting 1-4
Customize action button
  Toolbars dialog box 3-25
  Customize dialog box 3-26, 3-55
  Commands panel 3-26
  Toolbars panel 3-26
Customize menu item 3-55
  Options menu 3-54
  customizing
display of project files 2-12
  RTL and Technology views 2-14
toolbars 3-55
Cut icon
  Project toolbar 3-89
Cut menu item
  Edit menu 3-17, 3-18
Cycles column
  SCOPE spreadsheet 6-13
Cypress
  D-1
  instantiating LPMs in VHDL D-4
  product families D-2
  set_option Tcl command D-11
Cypress attributes and directives (table) D-12

D

d keyboard shortcut 3-95
data types
  for arithmetic in VHDL 9-7
  VHDL 9-5, 9-34
date and time header and footer variable
  (printing) 3-15
  -date command option 4-14
date header and footer variable
  (printing) 3-15
declaring and assigning objects, VHDL 9-8
default clock group 6-8
  -default command option 6-29, 6-31
  Default Enum Encoding dialog box zone
    Options for implementation dialog box 3-37
default_clkgroup clock group 6-8
  -default_enum_encoding command option 4-17
define_clock Tcl constraint file
command 4-24, 6-6, 6-25
examples of use 6-26
forward annotation
  Altera Max+Plus II 6-39, B-28
  Altera Quartus 6-39, B-18, B-34
  Lattice ORCA 6-41
  Xilinx 6-42, H-8
define_false_path Tcl constraint file
command 6-34
forward annotation
  Altera Quartus 6-39, B-18
  Xilinx 6-42, H-8
syn_keep and -through option 7-157
define_input_delay Tcl constraint file
command 4-24, 5-22, 6-10, 6-29
examples of use 6-29, 6-30
forward annotation
  Altera Max+Plus II 6-39, B-28
  Altera Quartus 6-39, B-18, B-34
  Lattice ORCA 6-41
  with syn_forward_io_constraints 7-65
  Xilinx 6-42, H-8
-route option 6-30
define_multicycle_path Tcl constraint file
command 6-32
examples of use 6-33
forward annotation
  Altera Quartus 6-39, B-18
  Xilinx 6-42, H-8
syn_keep and -through option 7-157
define_output_delay Tcl constraint file
command 4-24, 5-22, 6-10, 6-31
examples of use 6-31
forward annotation
  Altera Max+Plus II 6-39, B-28
  Altera Quartus 6-39, B-18, B-34
  Lattice ORCA 6-41
  with syn_forward_io_constraints 7-65
  Xilinx 6-42, H-8
define_reg_input_delay Tcl constraint file
command 4-24, 5-22, 6-27
examples of use 6-27
-route option 6-27
define_reg_output_delay Tcl constraint file
command 5-22, 6-28
defining an event outside a process (VHDL) 9-39
defining attributes 7-9
Defining clocks with the SCOPE spreadsheet (figure) 6-9
defining directives and attributes 7-14
  in source code 7-14
defining paths/points for timing exceptions 6-15
defining timing constraints 6-2
Delete action button
  Toolbars dialog box 3-25
Delete all bookmarks menu item
  Edit menu 3-17
Delete File menu item
  Project View popup menu 3-76
Delete icon
  License Editor dialog box 3-70
Delete keyboard shortcut 3-95
deleting bookmarks 2-8
Description column
  SCOPE spreadsheet 7-10
deselecting objects in HDL Analyst 5-6
design size, amount displayed on a sheet 2-15
designing with vendor technologies
  Actel A-1
  Altera B-1
  Atmel C-1
  Cypress D-1
  Lattice E-1
  QuickLogic F-1
  Triscend G-1
  Xilinx H-1
detailed clock report 6-49
Detailed clock report (table) 6-49
detecting forgotten assignment to the next state (VHDL) 9-54
detecting reset problems (VHDL) 9-53
device mapping options
  Actel A-3
  Altera
    ACEX B-26
    APEX B-14
    APEX II B-14
    Excalibur B-14
    FLEX B-26
    MAX B-32
device mapping options (continued)

Mercury B-14
Stratix B-20
Atmel C-2
Lattice
  GAL/isp E-11
  MACH E-12
  ORCA E-4
QuickLogic F-3
Triscend G-2
Xilinx
  CoolRunner H-40
  CoolRunner-II CPLDs H-40
  CPLDs H-40
  Spartan H-27
  Virtex H-30
  XC3000, XC5200 H-37
  XC4000 H-34
  XC9500 H-40

Device panel
  Options for implementation dialog box 3-32

Device panel in Options for implementation dialog box (figure) C-3

Device panel, Options for Implementation dialog box (figure) 3-32

dialog boxes 3-2
  Build Workspace 3-83
  Cells 3-52
  Copy File 3-79
  Customize 3-26, 3-55
  Display Settings 3-28
  Edit menu 3-19
  Editor Options 3-56, 3-58
  File menu 3-7
  File Options 3-41, 3-78
  File Options (VHDL) 3-41
  Find 3-19
  Font 3-58
  Format menu 3-52
  Goto 3-21
  HDL Analyst menu 3-49
  Help menu 3-66
  Help Topics 3-66
  Implementation Name 3-79
  Job Status 3-45
  License Editor 3-69
  License Wizard 3-67
  New 3-7

New Project 3-13
Object Query 3-49
Open 3-9, 3-44
Open Project 3-12, 3-83
Options for implementation 3-31, 3-41, 3-54, 3-82
Options menu 3-54
Page Setup 3-15
popup menus, Project View 3-80
Print 3-14
Print Setup 3-16
Project menu 3-31
Project Properties 3-80
Project View Options 2-12, 3-55, 3-84
Project Wizard 3-13
Replace 3-20
Run menu 3-44
Save As 3-10
Save Project As 3-84
Schematic Options 3-59
SCOPE Attributes 3-86, 7-12
Select element list 3-61
Select Files to Add to Project 3-11, 3-38, 3-82
Select Preferred License 3-71
Select Projects to Include in Workspace 3-14, 3-80, 3-81
Sheet Selection 3-27, 5-16
Source File 3-40
Styles 3-52
Syntax Coloring 3-58
Tip of the Day 3-72
Toolbars 3-24
View menu 3-24
different clock frequency goals 4-22
din attribute (Lattice) 7-41
d examples 7-41
-dir command option 4-13
Direction dialog box zone
  Find dialog box 3-20
  Replace dialog box 3-21
directives 7-1
  Actel A-10
  alphabetical summary 7-7
  Altera 8-36
  black box instantiation (VHDL) 9-86
  black_box_pad_pin 7-140
  black_box_tri_pins 7-142
  black-box source code 6-36
defining in source code 7-14
directives (continued)
   full_case 7-144
   include (QuickLogic) F-11
   individual descriptions 7-139
   overview 7-2
   parallel_case (Verilog) 7-147
   specifying 7-14
   specifying in Verilog 7-15
   specifying in VHDL 7-16
   summary 7-7
   syn_black_box 7-149, 8-41
   syn_enum_encoding 7-151
   syn_isclock 7-155
   syn_keep 7-157
      DSP MAC inference in Altera Stratix B-25
   syn_macro 7-161
   syn_noprune 7-162
   syn_preserve 7-167
      DSP MAC inference in Altera Stratix B-25
   syn_sharing 7-172
   syn_state_machine 7-173
   syn_tco 7-177
   syn_tpd 7-180
   syn_tristate 7-183
   syn_tsu 7-184
   syntax
      Verilog 7-15
      VHDL 7-16
   translate_off 7-187
   translate_on 7-187
   Verilog 8-47
   VHDL 9-85
   xc_isgsr (Xilinx) 7-189
   Xilinx H-43
   directives and attributes
      defining in source code 7-14
      overview 7-2
   -disable command
      option 4-10, 6-25, 6-27, 6-28, 6-29, 6-31, 6-32, 6-34
   -disable_io_insertion command
      option 4-17, A-9, B-16, B-21
   disabling I/O packing 7-92
   disabling replication 7-92
   Disconnect to Visual Elite menu item
      HDL Analyst menu, Visual Elite Cross Probing submenu 3-48

display of project files, preferences 2-12
Display Settings dialog box 3-28
   3D-Buttons check box 3-28
   Color zone 3-29
   Horizontal Lines check box 3-28
   Mark Current Column check box 3-29
   Mark Current Row check box 3-28
   Preview zone 3-28
   User Properties zone 3-29
   Vertical Lines check box 3-28
Display Settings dialog box (figure) 3-28
Display Settings dialog box features (table) 3-28
Dissolve Selected Instances menu item 3-85
   Technology View menu item 3-85
document icon (online help) 3-66
   -domap command option 4-17
   dout attribute (Lattice) 7-42
      examples 7-42
DSP MAC block inference
   Altera Stratix B-23
DSP MAC block inference in Altera Stratix (table) B-23
Duty Cycle column
   SCOPE spreadsheet 6-7
   -duty_pct command option 6-26

E

E (Shift-e) keyboard shortcut 3-95
   e keyboard shortcut 3-95
EABs
   altera_auto_use_eab 7-25
EDIF output file
   appending characters 7-56
   appending vector range 7-56
   converting parentheses 7-55
   filenames in Altera B-9
   hierarchy control 7-74
   keeping scalars 7-75
   name format 7-55
   port names in Altera 7-58
   scalar name format 7-60
   syn_edif_bit_format 7-55
   syn_edif_name_length 7-58
   syn_edif_scalar_format 7-60
   using upper case 7-56
Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Edit Attributes menu item
   RTL View popup menu, SCOPE submenu 3-86
   Technology View popup menu, SCOPE submenu 3-86
Edit menu 3-16
   Advanced submenu 3-17
      Comment Code item 3-17
      Lowercase item 3-17
      Uppercase item 3-17
   Clear item 3-18
   commands 3-17
   Copy Image item 3-19
   Copy item 3-17, 3-18
   Cut item 3-17, 3-18
   Delete all bookmarks item 3-17
dialog boxes 3-19
   Fill Down item 3-18
   Find again item 3-19
   Find item 3-17, 3-19
   Goto item 3-17
   Insert Quick item 3-18
   Insert Rows item 3-18
   Insert Wizard item 3-18
   Next bookmark item 3-17
   Paste item 3-17, 3-18
   Previous bookmark item 3-17
   Redo item 3-17, 3-18
   Remove Rows item 3-18
   Replace item 3-17, 3-19
   Select All item 3-17
   Toggle bookmark item 3-17
   Undo item 3-17, 3-18
   Edit Menu commands
      in RTL/Technology views 3-19
      in SCOPE window 3-18
      in Text Editor view 3-17
   Edit menu commands (table) 3-17
   Edit or enter license information received from Synplicity radio button
   Introduction dialog box (License Wizard) 3-67
Edit toolbar 3-92
   Clear All Bookmarks icon 3-92
   Next Bookmark icon 3-92
   Previous Bookmark icon 3-92
   Toggle Bookmark icon 3-92
   Edit toolbar (figure) 3-92
   Edit toolbar icons (table) 3-92
   editor options 3-56

Editor Options dialog box 3-56, 3-58
   ... action button 3-57
   External Editor radio button 3-57
   Internal Editor radio button 3-57
   Options action button 3-57
   Editor Options dialog box (figure) 3-57
   Editor Options dialog box features (table) 3-57
   Editor Options menu item
      Options menu 3-54, 3-56
      effect of encoding styles 7-152
   Enable check boxes
      SCOPE Attributes dialog box 3-86
      -enable command option 4-10
   Enabled column
      SCOPE spreadsheet 6-6, 6-11, 6-12, 6-14, 6-17, 7-9
   ending points
      multicycle and false paths 6-15
   Enhanced Text Crossprobing check box
      Schematic Options dialog box 3-60
      enhanced text crossprobing option 3-61
   Entering licensing information automatically (table) 3-70
   enumerated types
      syn_enum_encoding directive 7-151
tips for simulation (VHDL) 9-53
   enumeration encoding, setting (VHDL) 3-54
   Environment Variable Used dialog box zone
      License Editor dialog box 3-69
   environment variables
      examples, accessing 4-11
      LM_LICENSE_FILE 5-9
      VSH_LIB 5-9, 5-10
      VSH_VISUAL_COMMAND 5-9, 5-10
   ESBs
      altera_auto_use_esb 7-28
      altera_implement_in_esb 7-35
   Esc keyboard shortcut (Text Editor) 2-8
   event, defining outside a process (VHDL) 9-39
   Example: the SCOPE spreadsheet’s Other tab from and to (table) E-8
   examples
      accessing a popup menu 2-11
      accessing environment variables 4-11
      accessing libraries (VHDL) 9-13
      accessing packages (VHDL) 9-13

Synplicity Reference Manual, April 2002
examples (continued)
also attribute
   sdc file  7-19
   Verilog  7-19
   VHDL  7-20
alspin attribute
   sdc file  7-21
   Verilog  7-21
   VHDL  7-22
alspreserve attribute
   sdc file  7-23
   Verilog  7-23
   VHDL  7-23
altera Quartus II, forward annotation 6-40
altera_auto_chip_pin_lc attribute
   sdc file  7-31
   Verilog  7-31
   VHDL  7-32
altera_auto_use_eab attribute
   sdc file  7-25
   Verilog  7-25
   VHDL  7-26
altera_auto_use_esb attribute
   sdc file  7-28
   Verilog  7-28
   VHDL  7-29
altera_implement_in_eab attribute
   sdc file  7-33
   Verilog  7-33
   VHDL  7-34
altera_implement_in_esb attribute
   sdc file  7-35
   Verilog  7-35
   VHDL  7-36
altera_io_opendrain attribute
   sdc file  7-37
   Verilog  7-37
   VHDL  7-38
altera_io_powerup attribute
   sdc file  7-39
   Verilog  7-39
   VHDL  7-40
always blocks, event control (Verilog) 8-17
always blocks, multiple event control arguments (Verilog) 8-8
asynchronous sets and resets (Verilog) 8-24
asynchronous state machines (Verilog) 8-33
asynchronous state machines (VHDL) 9-55
attributes and directives, syntax
   Verilog  7-15
   VHDL  7-16, 7-17
black_box_pad_pin directive
   Verilog  7-140
   VHDL  7-141
black_box_tri_pins directive
   Verilog  7-142
   VHDL  7-143
calculation of relationships between source and destination clocks 6-9
case statement (VHDL) 9-21
clock groups, worst case timing calculation 6-8
color coding in log file 1-28
combinational logic (Verilog) 8-4
combined data, port types (Verilog) 8-4
comma-separated sensitivity list (Verilog) 8-5
conditional signal assignments (VHDL) 9-23
configuration declaration (VHDL) 9-65
configuration specification (VHDL) 9-61
constants (VHDL) 9-11
continuous assignments for combinational logic (Verilog) 8-18
creating a hierarchical design (Verilog) 8-43
creating a hierarchical design (VHDL) 9-59
creating a resource library (VHDL) 9-14
creating a scalable adder (Verilog) 8-12
creating a scalable design (VHDL)
   using generics 9-73
   using unconstrained vector ports 9-71
creating a scalable design using generate statements (VHDL) 9-75
creating a scalable module (Verilog) 8-11
creating flip-flops and registers using processes (VHDL) 9-37
creating flip-flops in VHDL 9-36
creating level-sensitive latches (VHDL) 9-42
examples (continued)
creating unwanted level-sensitive latches (VHDL) 9-43
D flip-flop with asynchronous, active-high reset, set (VHDL) 9-45
D flip-flop with synchronous, active-high reset, set (VHDL) 9-48
default assignment (VHDL) 9-52
defining paths/points for timing exceptions 6-16
delivered with synthesis tool 1-15
detecting forgotten assignment to the next state (VHDL) 9-54
din attribute 7-41
directory 1-15
dout attribute 7-42
false paths 6-15
forward annotation, Altera Quartus II 6-40
FSM coding (Verilog) 8-30
FSM coding style (VHDL) 9-51
FSM default state assignment (Verilog) 8-31
FSM, defining state values (Verilog) 8-32
full_case directive 7-144
generate statement (Verilog 2001) 8-6
generating synchronous read RAMs (VHDL) 9-81
if-then-else statement (VHDL) 9-20
inferring RAM with special write enables (Verilog) 8-35
inferring RAMs, nested write enables (VHDL) 9-80
inferring ROMs (VHDL) 9-82
instantiatiating a technology-vendor I/O (VHDL) 9-88
level-sensitive latches (Verilog) 8-21
level-sensitive latches using concurrent signal assignments (VHDL) 9-41
loc attribute 7-43
lock attribute 7-44
model template (VHDL) 9-34
multicycle paths 6-13
multiple constraints on the same object 6-16
multiplexor (Verilog) 8-17
net buffering report 1-29
object naming 6-23
orca_padtype attribute 7-45
orca_props attribute 7-47
overriding a parameter value with # (Verilog) 8-13
with defparam (Verilog) 8-12
parallel_case directive 7-149
ql_padtype attribute
sdc file 7-49
Verilog 7-49
VHDL 7-50
ql_placement attribute
sdc file 7-51
Verilog 7-51
VHDL 7-51
RAM inference (VHDL) 9-77
RAMs with always-enabled write enables (VHDL) 9-79
RAMs with special write enables (VHDL) 9-78
resource usage report
Altera 8-12
RGB (color) value adjustment in HDL Analyst 2-14, 5-7
rise and fall constraints 6-9
ROM inference (Verilog) 8-38
selected signal assignments (VHDL) 9-22
sequential logic (Verilog) 8-19
signal assignments in always blocks (Verilog) 8-21
signals and ports (VHDL) 9-9
signed signals (Verilog 2001) 8-6
simple component instantiation (VHDL) 9-25
simple signal assignments (VHDL) 9-22
SNS (VHDL)
constants 9-27
functions 9-28
operators 9-29
user-defined functions 9-30
specifying black-box attributes
Verilog 6-38
VHDL 8-37
status bar information, HDL Analyst 5-5
syn_black_box directive 7-149
syn_direct_enable attribute
Verilog 7-53
VHDL 7-54
syn_edif_bit_format attribute
sdc file 7-55
Verilog 7-57
VHDL 7-57
examples (continued)

syn_edif_name_length attribute
  Verilog 7-58
  VHDL 7-59

syn_edif_scalar_format attribute 7-60

syn_encoding attribute
  Verilog 7-63
  VHDL 7-64

syn_enum_encoding directive 7-152

syn_forward_io_constraints attribute
  sdc file 7-65
  Verilog 7-65
  VHDL 7-66

syn_hier attribute 7-68

syn_isclock directive 7-155

syn_keep directive 7-158

syn_macro directive 7-161

syn_maxfan attribute
  sdc file 7-70
  Verilog 7-71
  VHDL 7-71

syn_multistyle attribute
  sdc file 7-72
  Verilog 7-72
  VHDL 7-73

syn_netlist_hierarchy attribute 7-74

syn_noarrayports attribute 7-75

syn_nocheckbuf attribute
  sdc file 7-76
  Verilog 7-76
  VHDL 7-77

syn_noprune directive 7-162

syn_pipeline attribute
  sdc file 7-78
  Verilog 7-78
  VHDL 7-79

syn_preserve directive 7-168

syn_preserve_sr_priority attribute
  sdc file 7-81
  Verilog 7-81
  VHDL 7-82

syn_props attribute 7-83

syn_radhardlevel attribute 7-86

syn_ramstyle attribute 7-89

syn_reference_clock attribute 6-35, 7-91

syn_replicate attribute
  sdc file 7-92
  Verilog 7-92

VHDL 7-93

syn_resources attribute
  sdc file 7-95
  Verilog 7-95
  VHDL 7-96

syn_romstyle attribute
  sdc file 7-97, 7-100
  Verilog 7-97, 7-100
  VHDL 7-98, 7-101

syn_sharing directive 7-172

syn_srlstyle attribute
  Verilog 7-102
  VHDL 7-103

syn_state_machine directive 7-173

syn_tco directive
  Verilog 7-177
  VHDL 7-178

syn_tpd directive
  Verilog 7-180
  VHDL 7-181

syn_translate_off directive
  Verilog 7-187
  VHDL 7-188

syn_translate_on directive
  Verilog 7-187
  VHDL 7-188

syn_tristate directive 7-183

syn_tristatetomux attribute
  sdc file 7-104
  Verilog 7-104
  VHDL 7-105

syn_tsu directive
  Verilog 7-184
  VHDL 7-185

syn_useenables attribute 7-106

syn_useioff attribute
  sdc file 7-108, 7-110, 7-112, 7-114
  Verilog 7-108, 7-110, 7-112, 7-114
  VHDL 7-109, 7-111, 7-113, 7-114

synchronous READ RAMs (Verilog) 8-37

synchronous sets and resets
  (Verilog) 8-26

Tcl scripts 4-21

tooltip information, state machine 5-4

use of Push Tristates across
  Process/Block boundaries
  option 3-37

user-defined enumeration types
  (VHDL) 9-7
examples (continued)

using a scalable architecture  
(VHDL) 9-73
using a scalable module (Verilog) 8-11
using a wait statement inside a process  
(VHDL) 9-39
using Actel macro libraries A-7
using add_file Tcl command 4-10
using constraint_file Tcl command 4-11
using define_clock Tcl constraint file  
command 6-28
using define_input_delay Tcl constraint  
file command 6-30
with -route option 6-30
using define_input_delay Tcl constraint  
file command with -default  
option 6-29
using define_multicycle_path Tcl  
constraint file command 6-33
using define_output_delay Tcl  
constraint file command with  
-default option 6-31
using define_reg_input_delay Tcl  
constraint file command with  
-route option 6-27
using project_file Tcl command 4-15
using projectTcl command 4-13
using Slack Margin to find critical path  
instances with worst-case  
slack 5-19, 5-20
using syn_ramstyle attribute  
Altera 8-5
using syn_romstyle attribute  
Altera 8-7
using synthesis macro (Verilog) 8-45
variables (VHDL) 9-10

Verilog
always blocks, event control 8-17
always blocks, multiple event control  
arguments 8-8
asynchronous sets and resets 8-24
asynchronous state machines 8-33
attribute and directive syntax 7-15
combinational logic 8-15
combined data, port types 8-4
comma-separated sensitivity list 8-5
continuous assignments for combi-
national logic 8-18
creating a hierarchical design 8-43
creating a scalable adder 8-12
creating a scalable module 8-11
FSM coding 8-30
FSM default state assignment 8-31
FSM, defining state values 8-32
generate statement (Verilog 2001) 8-6
inferring RAM with special write  
enables 8-35
level-sensitive latches 8-21
multiplexor 8-17
overriding a parameter value  
with # 8-13
with defparam 8-12
ROM inference 8-38
sequential logic 8-19
signal assignments in always  
blocks 8-21
signed signals (Verilog 2001) 8-6
specifying black-box attributes 6-38
synchronous READ RAMs 8-37
synchronous sets and resets 8-26
using a scalable module 8-11
using synthesis macro 8-45
wildcard (*) in sensitivity list 8-5

VHDL
accessing libraries 9-13
accessing packages 9-13
asynchronous state machines 9-55
attribute and directive  
syntax 7-16, 7-17
case statement 9-21
conditional signal assignments 9-23
configuration declaration 9-65
configuration specification 9-61
constants 9-11
creating a hierarchical design 9-59
creating a resource library 9-14
creating a scalable design using gen-
erate statements 9-75
creating a scalable design using  
generics 9-73
creating a scalable design using un-
constrained vector ports 9-71
creating flip-flops and registers using  
processes 9-37
creating level-sensitive latches 9-42
examples (continued)
creating unwanted level-sensitive
latches 9-43
D flip-flop with asynchronous, ac-
tive-high reset, set 9-45
D flip-flop with synchronous, ac-
tive-high reset, set 9-48
default assignment 9-52
detecting forgotten assignment to the
next state 9-54
FSM coding style 9-51
generating synchronous read
RAMs 9-81
if-then-else statement 9-20
inferring RAMs, nested write
enables 9-80
inferring ROMs 9-82
instantiating a technology-vendor
I/O 9-88
level-sensitive latches using concur-
rent signal assignments 9-41
model template 9-34
RAM inference 9-77
RAMs with always-enabled write
enables 9-79
RAMs with special write enables 9-78
selected signal assignments 9-22
signals and ports 9-9
simple component instantiation 9-25
simple signal assignments 9-22
SNS constants 9-27
SNS functions 9-28
SNS operators 9-29
SNS user-defined functions 9-30
specifying black-box attributes 6-37
user-defined enumeration types 9-7
using a scalable architecture 9-73
using a wait statement inside a
processes 9-39
variables 9-10
wildcard (*) in sensitivity list
(Verilog) 8-5
xc_alias attribute 7-116
xc_clockbuftime attribute 7-117
xc_fast attribute 7-119
xc_fast_auto attribute 7-121
xc_global_buffers attribute 7-123
xc_isgsr directive 7-189
xc_loc attribute 7-124
xc_map attribute 7-125
xc_ncf_auto_relax attribute 7-126
xc_nodelay attribute 7-128
xc_padtype attribute 7-129
xc_props attribute 7-131
xc_pulldown attribute 7-132
xc_pullup attribute 7-132
xc_rloc attribute
sdc file 7-134
Verilog 7-134
VHDL 7-135
xc_slow attribute 7-136
xc_uset attribute 7-137
examples directory 1-15
Excalibur menu options (table) B-15
Excalibur project Tcl command arguments
(table) B-17
Excalibur set_option Tcl command argu-
ments (table) B-15
executing Tcl scripts 4-5
Existing Project action button
Open Project dialog box 3-12
Exit menu item
File menu 3-6
Expand menu item
HDL Analyst menu 3-48
Expand paths menu item
HDL Analyst menu 3-48
Expand to register/port menu item
HDL Analyst menu 3-48
External Cross Probing Engaged menu
item 5-11
HDL Analyst menu 3-48
External Editor radio button
Editor Options dialog box 3-57
F
F1 keyboard shortcut 3-95
F10 keyboard shortcut 3-96
F11 keyboard shortcut 3-96
F12 keyboard shortcut 3-96, 5-17
F2 keyboard shortcut 3-95, 5-13, 5-14
F4 keyboard shortcut 3-96
F5 keyboard shortcut 3-96
F7 keyboard shortcut 3-96
F8 keyboard shortcut 3-96
Fall At column
  SCOPE spreadsheet 6-7
  -fall command option 6-26
false paths 6-14
  constraint points 6-15
  define_false_path timing constraint 6-34
  examples 6-15
False Paths panel
  Comment column 6-15
  Enabled column 6-14
  From column 6-14
  SCOPE spreadsheet 6-14
  Through column 6-15
  To column 6-15
  -fanin_limit command option 4-18, B-33
fanout guide, overriding default 7-70
fanout limits
  Actel
    hard limit A-4
  Altera B-17
  Lattice ORCA E-4
fanout limitsActel guideline A-3
  -fanout_guide command option 4-18, 7-70, A-8
  -fanout_limit command option 7-70
fanouts, Xilinx H-2
  -file command option 4-13
file format options
  Altera
    ACEX B-27
    FLEX B-27
    MAX B-33
    Stratix B-21
  Altera APEX B-16
  Altera APEX II B-16
  Altera Excalibur B-16
  Altera Mercury B-16
File Location dialog box zone
  New dialog box 3-8
File menu
  Build Project item 3-6, 3-11
  Close item 3-5
  Close Project item 3-6
  commands 3-5
dialog boxes 3-7
  Exit item 3-6
  New item 3-5
  New Project item 3-7
  New Workspace item 3-7, 3-14
  Open item 3-5
  Open Project item 3-6
  Page Setup item 3-6, 3-15
  Print Image item 3-7
  Print item 3-6
  Print Preview item 3-6, 3-14
  Print Setup item 3-6, 3-15
  recent files list 3-6
  Recent Projects item 3-6
  Save All item 3-6
  Save As item 3-6
  Save item 3-5
  Save Project item 3-7
File menu commands (table) 3-5
File Name dialog box zone
  New dialog box 3-8
File name dialog box zone 3-81
  Open dialog box 3-9
  Save As dialog box 3-11
  Select Files to Add to Project dialog box 3-39
  Select Projects to Include in Workspace dialog box 3-81
  Source File dialog box 3-40
file name header and footer variable (printing) 3-15
File Options dialog box 3-41, 3-78
  Absolute Path radio button 3-78
  File Path zone 3-78
  File type zone 3-78
  Library Name zone 3-78
  Relative to Project radio button 3-78
  Save file zone 3-78
  Verilog 2001 check box 3-78
  Verilog Standard zone 3-78
File Options dialog box (figure) 3-79
File Options dialog box (VHDL) 3-41
File Options menu item
  Project View popup menu 3-74, 3-75, 3-76
  Project view popup menu 3-78
File Path dialog box zone
  File Options dialog box 3-78
  File Time radio button 3-15
  File Type dialog box zone
    New dialog box 3-8
  File type dialog box zone
    File Options dialog box 3-78
files
Index

HDL Analyst menu, Technology
submenu 3-47
Flattened View menu item
HDL Analyst menu
RTL submenu 3-47
Technology submenu 3-47
FLEX and ACEX menu options (table) B-26
FLEX and ACEX project Tcl command arguments (table) B-28
FLEX set_option Tcl command arguments (table) B-27
flip-flops
using always blocks (Verilog) 8-20
using VHDL wait statement 9-39
Verilog 8-20
VHDL 9-36
Flip-flops with synchronous sets and resets (figure) 8-27, 9-47
Floating License Usage menu item
Help menu 3-65
folders for organizing project files 2-12
following objects across design levels and views 5-12
Font dialog box 3-58
Font dialog box zone
Schematic Options dialog box 3-61
Font Options dialog box zone
Schematic Options dialog box 3-59
Footer dialog box zone 3-15
-force_gsr command option 4-17, E-6, H-28, H-35, H-38
Foreground Compile menu item
Options menu, Quartus submenu (Altera) B-19
Format menu 3-51
Align item 3-51
Cells item 3-51, 3-52
commands 3-51
Cover Cells item 3-51
dialog boxes 3-52
Remove Covering item 3-51
Resize Columns item 3-51
Resize Rows item 3-51
Style item 3-51
Styles item 3-51, 3-52
Format menu commands (table) 3-51
forward annotation 6-39
Altera B-34
APEX B-17
APEX II B-17
Excalibur B-17
FLEX B-28
Max+Plus II 6-39
Mercury B-17
Quartus II 6-39
Stratix B-17
I/O timing constraints 6-42
Lattice ORCA 6-41
Max+Plus II (Altera) 6-39, B-28
ORCA Foundry 6-41
properties (Triscend) 7-83
properties (Xilinx) 7-131
Quartus II (Altera) 6-39, B-28
relaxing Xilinx constraints 6-43
syn_forward_i_o_constraints 7-65
Xilinx 6-42, H-8
-frequency command option 6-25
Frequency (MHz) dialog box zone
Project view 3-98
Frequency column
SCOPE spreadsheet 6-6
-frequency command option 4-17, 6-24, 6-25
From column
SCOPE spreadsheet 6-12, 6-14
-from command option 6-32, 6-34
fse file 1-30
FSM Compiler
enabling and disabling with GUI 1-22
FSM Compiler operations (table) 1-22
FSM Compiler, enabling and disabling with-
-symbolic_fsm_compiler command option 4-16, 4-23
FSM encoding file 1-30
FSM encoding style
syn_encoding attribute 7-62
FSMs
encoding style (Verilog) 8-30
encoding style (VHDL) 9-50
optimization 1-21
Verilog 8-29
Full Path dialog box zone
New dialog box 3-8
Full View menu item
View menu 3-24
full_case directive 7-144
examples 7-144
functions (VHDL)
  selected name support 9-28
  selected name support for user-defined 9-30

G
gate primitives, Verilog 8-14
General Options dialog box zone
  Schematic Options dialog box 3-59
generate statement
Verilog 2001, examples 8-6
VHDL 9-75
Generated technology-specific RAMs
  (table) 8-34, 9-77
generating a timing report 6-46
get_env Tcl command 4-11
get_option Tcl command 4-12
Getting help with Synplicity products
  (table) 1-3
global buffers
  xc_global_buffers attribute 7-123
Xilinx H-3
global set/reset 7-189
Xilinx H-4
 glue logic 7-89
disabling creation in Altera Stratix B-22
Goto dialog box 3-21
Goto dialog box (figure) 3-21
Goto Line Number dialog box zone
  Goto dialog box 3-21
Goto menu item
  Edit menu 3-17
Goto Net Driver menu item
  View menu 3-23
-grade command option 4-16
graphical user interface 1-8
GSR resource, Lattice ORCA E-5
GUI, overview 1-8, 2-1

H
HDL Analyst 5-1, 5-2, 8-38, 9-82
  analyzing critical paths 5-18
  changing amount of logic displayed 5-7
  changing the colors used 5-7
  color adjustment examples 2-14, 5-7
  color preferences 2-14
  commands 5-3
critical paths 5-18
crossprobing 5-8
  ModelSim 5-11
crossprobing to Visual Elite 5-8
filtering schematics 5-17
finding objects 5-5, 5-6
getting started 5-4
handling negative slack 5-21
Hierarchy Browser 5-12
interpreting critical path information 5-19
multi-sheet schematics 5-15
opening windows 5-2
Push/Pop mode 5-13
ROM table viewer 8-38, 9-82
selecting objects 5-6
setting colors 2-14, 5-7
setup/licensing 5-4
sheet connectors 5-15
status bar 5-5
tooltip information 5-4
unselecting objects 5-6
viewing critical paths 5-18
viewing other sheets 5-15
views 5-2
HDL Analyst menu 3-45
  command summary 3-46
  commands 3-46
dialog boxes 3-46
  Expand item 3-48
  Expand path item 3-48
  Expand to register/port item 3-48
  External Cross Probing Engaged item 3-48, 5-11
  Filter Schematic item 3-47, 5-17, 5-18, 5-20
  Find item 3-48, 3-49
  Flatten current schematic item 3-47
  Flatten current schematic to gates item 3-47
  RTL submenu 3-46, 3-47
    Flattened View item 3-47
    Hierarchical View item 3-47
  Set Slack Margin item 3-47
  Show Critical Path item 3-47
  Show Timing Information item 3-47
Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

HDL Analyst menu (continued)
  Technology submenu 3-46, 3-47
  Flattened Critical Path item 3-47
  Flattened to Gates View item 3-47
  Flattened View item 3-47
  Hierarchical Critical Path item 3-47
  Hierarchical View item 3-47
  Visual Elite Cross Probing
     submenu 3-45, 3-48
     Connect to Visual Elite item 3-48
     Cross Probing Enabled item 3-48
     Disconnect to Visual Elite item 3-48
HDL Analyst menu commands (table) 3-47
header and footer variables (printing) 3-15
Header dialog box zone 3-15
header, timing report 6-46
help (Synplicity customer support) 1-4
Help menu 3-64
  About this program item 3-65
  Additional Products item 3-65
  commands 3-65
dialog boxes 3-66
  Floating License Usage item 3-65
  Help item 3-65, 3-66
  How to Use Help item 3-65, 3-66
  License Agreement item 3-65
  License Editor item 3-65, 3-69
  License Wizard item 3-65
  Online Documents item 3-65
  Preferred License Selection 3-65
  Preferred License Selection item 3-65
  Synplify vs. Synplify Pro item 3-65
  TCL Help item 3-65, 3-66, 4-2
  Tip of the Day item 3-65, 3-72
Help menu commands (table) 3-65
Help menu item 3-66
  Help menu 3-65
  Help Topics dialog box 3-66
     Contents panel 3-66
     Find panel 3-67
     Index panel 3-67
  Help Topics dialog box, Contents panel
     (figure) 3-66
  Help Topics dialog box, Index panel
     (figure) 3-67
  Hierarchical Critical Path menu item
  HDL Analyst menu, Technology
     submenu 3-47
  hierarchical designs, VHDL 9-56
Hierarchical View menu item
  HDL Analyst menu, RTL submenu 3-47
  HDL Analyst menu, Technology
     submenu 3-47
hierarchy
  Verilog 8-43
  VHDL 9-56
  viewing 3-91
Hierarchy Browser
  RTL view 2-4
  Technology view 2-5
highlight, flattening (syn_hier attribute) 7-67
Highlight Wildcard (?) dialog box zone
  Object Query dialog box 3-50
Highlighted dialog box zone
  Object Query dialog box 3-50
highlighting critical path 5-18
Horizontal Lines check box, Display Settings
dialog box 3-28
Horizontally tiled check box, Display Settings
dialog box 3-63
How syn_preserve retains registers during
  optimization (figure) 7-168
How to Use Help menu item 3-66
  Help menu 3-65

i
  i keyboard shortcut 3-96
I/O insertion
     Actel A-4
     Altera B-15
     Stratix B-20
     Xilinx, automatic insertion H-22
     Xilinx, manual H-23
     Xilinx, VHDL H-24
I/O locations in Altera B-39
I/O packing, disabling 7-92
I/O properties (table) 7-48
I/O registers, power-up mode (Altera) 7-39
I/O timing constraints, forward
  annotation 6-42
I/Os
  See also ports
     instantiating in Verilog 8-40
     instantiating in VHDL 9-87
iconified windows, arranging 3-62
icons
Analyst toolbar 3-90
Edit toolbar 3-92
Project toolbar 3-88
View toolbar 3-91
IEEE 1364 Verilog 95 standard 1-25
ieee library (VHDL) 9-12
if-then-else statement (VHDL) 9-19
ignored language constructs
Verilog 8-3
VHDL 9-4
Implementation Name dialog box 3-79
Implementation Name dialog box (figure) 3-79
Implementation Name dialog box zone
Copy File dialog box 3-79
implementation options in Altera B-10
Implementation Options menu item
Project Menu 3-30
Project View popup menu 3-78
Implementation Results panel
Options for implementation dialog box 1-26, 3-33
Implementation Results panel, Options for Implementation dialog box (figure) 3-34
implementations
creating 3-41
naming 3-33
Implementations dialog box zone
Project Properties dialog box 3-80
-improve command option 1-18, 5-21
include directive (QuickLogic) F-11
Include Path Order dialog box zone
Options for implementation dialog box 3-36
-include_path command option 4-17
including command files in a project 4-5
indenting text (Text Editor) 2-8
Index panel
Help Topics dialog box 3-67
Inferring RAM
Xilinx H-10
inferring RAM
Altera B-4
inferring registers
Altera B-9
Lattice ORCA E-6
Xilinx H-14
inferring ROM
Altera B-6
inferring SRL tables (Xilinx) H-15
info file 1-30, 8-38
ini file 2-14, 5-7
input delays, removing (Xilinx) 7-128
Inputs/Outputs panel
Clock Edge column 6-11
Comment column 6-11
Enabled column 6-11
Port column 6-11
Route column 6-11
SCOPE spreadsheet 6-10, 6-11
Type column 6-11
Value column 6-11
Inputs/Outputs panel wizard, SCOPE spreadsheet 6-19, 7-14
-insert command option 4-13
Insert Project menu item 3-80
Project View popup menu 3-77
Insert Quick menu item
Edit menu 3-18
Insert Rows menu item
Edit menu 3-18
Insert Wizard dialog box (Inputs/Outputs panel) (figure) 6-18
Insert Wizard menu item
Edit menu 3-18
Insert Wizard Step 1 dialog box (figure) 7-12
Insert Wizard Step 2 dialog box (figure) 7-13
inserting bookmarks 2-8
instance name syntax
Verilog 6-21
VHDL 6-22
Instances menu item
View menu, Select All submenu 3-23
Instances panel
Object Query dialog box 3-49
instances, preserving with syn_noprune directive 7-162
instantiating a technology-vendor I/O
Verilog 8-41
VHDL 9-87
instantiating black boxes
Altera
Verilog B-3
VHDL B-3
instantiating black boxes (continued)
  Lattice E-2
  Verilog 8-40, 8-41
  VHDL 9-86, 9-87
instantiating components, VHDL 9-15
instantiating LPMs
  Altera
    Verilog B-4, B-41
    VHDL B-4, B-46
  Cypress
    VHDL D-4
instantiating macros
  Altera
    Verilog B-3
    VHDL B-3
  Lattice E-2
  Xilinx H-7
instantiating special buffers in Altera
  Verilog B-43
  VHDL B-49
instprop Tcl command (QuickLogic) F-10
integer data type (VHDL) 9-7
interface information, timing report 6-48
Interface report (table) 6-48
Internal Editor radio button
  Editor Options dialog box 3-57
interpreting critical path 5-19
Introduction dialog box
  Edit or enter license information
    received from Synplicity radio button 3-67
  Fill out a form to request a trial license
    from Synplicity radio button 3-67
Introduction dialog box (figure) 3-68
ispDesignExpert (Lattice) E-2

J

Job Command dialog box zone
  Job Status dialog box 3-45
Job Name dialog box zone
  Job Status dialog box 3-45
Job Status dialog box 3-45
  Cancel Job action button 3-45
  Job Command zone 3-45
  Job Name zone 3-45
  Run Time zone 3-45
  Job Status dialog box (figure) 3-45
Job Status menu item
  Run menu 3-44, 3-45
  Jump Location check box
    Object Query dialog box 3-50

K

-k command option (Xilinx) H-33
keyboard shortcuts 3-2, 3-3, 3-93
  Alt (Text Editor) 2-9, 3-61
  Ctrl (with mouse) 2-11
  Ctrl-I 3-93
  Ctrl-a 3-93
  Ctrl-c 3-93
  Ctrl-f 3-94
  Ctrl-F2 3-93
  Text Editor 2-8
  Ctrl-F4 2-13, 3-93
  Ctrl-F6 2-13, 3-93
  Ctrl-g 3-94
  Ctrl-h 3-94
  Ctrl-l (Shift i) 3-95
  Ctrl-k 3-94, 5-11
  Ctrl-I 3-94
  Ctrl-n 3-94, 5-15
  Ctrl-N (Shift n) 3-95
  Ctrl-o 3-94
  Ctrl-p 3-94, 5-16
  Ctrl-P (Shift p) 3-95
  Ctrl-r 3-94
  Ctrl-s 3-94, 5-16
  Ctrl-Shift-F2 3-95
  Text Editor 2-9
  Ctrl-Shift-i 3-95
  Ctrl-Shift-n 3-95
  Ctrl-Shift-p 3-95
  Ctrl-v 3-94
  Ctrl-x 3-94
  Ctrl-y 3-95
  Ctrl-z 3-95
d 3-95
Delete 3-95
e 3-95
E (Shift-e) 3-95
Esc (Text Editor) 2-8
F1 3-95
F10 3-96
F11 3-96
F12 3-96, 5-17
F2 3-95, 5-13, 5-14
F4 3-96
keyboard shortcuts (continued)
F5 3-96
F7 3-96
F8 3-96
i 3-96
p 3-96
Shift (scrolling with mouse) 2-11
Shift-e 3-95
Shift-F11 3-97
Shift-F2 3-96
Shift-F5 3-96
Shift-F7 3-96
Shift-F8 3-96
Shift-Left Arrow 3-97
Shift-Right Arrow 3-97
Shift-Tab (Text Editor) 2-8
Tab (Text Editor) 2-8
Keyboard shortcuts (table) 3-93
Keyboard shortcuts indicated in menus
(figure) 3-93
keyword completion, text editing 2-8

L
-l command option (Xilinx) H-33
language constructs
Verilog 8-2
    ignored 8-3
    supported 8-2
    unsupported 8-3
VHDL 9-5
    ignored 9-4
    miscellaneous 9-3
    supported 9-2
    unsupported 9-4
language guidelines
Verilog 8-8
VHDL 9-33
Large Buttons check box
   Toolbars dialog box 3-26
latches
always blocks (Verilog) 8-22
concurrent signal assignment
   (VHDL) 9-40
continuous assignments (Verilog) 8-21
    creating in VHDL 9-40
error message (VHDL) 9-43
level-sensitive 8-22, 8-23
process blocks (VHDL) 9-41
Xilinx H-15
Lattice E-1
black boxes E-2
device mapping options
   GAL/isp E-11
   MACH E-12
   ORCA E-4
din attribute 7-41
dout attribute 7-42
GAL/isp
device mapping options E-11
   set_option Tcl command E-11
loc attribute 7-43
lock attribute 7-44
MACH
   project Tcl command E-14
   set_option Tcl command E-13
MACH device mapping options E-12
macros E-2
ORCA
   device mapping options E-4
   fanout control E-4
   forward annotation 6-41
   forward-annotation 6-41, E-8
   GSR resource E-5
   I/O properties 7-47
   inferring registers E-6
   pad types 7-45
   pin locations 7-43
   set_option Tcl command E-6
   syn_useioff attribute 7-110
   timing constraints 6-41
orca_padtype attribute 7-45
orca_props attribute 7-47
pin locations 7-44
product families E-2
Lattice attributes and directives (table) E-16
Lattice GAL/isp set_option command options
   (table) E-11
Lattice MACH set_option command options
   (table) E-13
launching synthesis from Visual Elite 5-10
level-sensitive latches 8-22, 8-23
using always blocks
   Verilog 8-22
using concurrent signal assignments
   VHDL 9-40
Index

Index-30

Synpify Reference Manual, April 2002
mapper output file 1-27
mapping EABs in Altera B-40, B-45
mapping to ATOMs, Altera
APEX B-15
APEX II, Excalibur, Mercury B-15
Stratix B-20
Mark All action button
Find dialog box 3-20
Mark Current Column check box
Display Settings dialog box 3-29
Mark Current Row check box
Display Settings dialog box 3-28
Match Case check box
Find dialog box 3-20
Replace dialog box 3-21
MAX menu options (table) B-32
MAX project Tcl command arguments (table) B-34
MAX set_option Tcl command arguments (table) B-33
Max+Plus II
ACEX, configuring B-29
configuring for area B-31
configuring for speed B-31
FLEX10K, configuring B-29
forward annotation 6-39, B-28
MAX
accessing LMF B-35
configuring B-34
selecting synthesis style B-35
subdirectories for EDIF B-9
subdirectories for VQM B-9
-max_terms_per_macrocell command option 4-18
-maxfan command
option 4-18, E-6, F-6, H-28, H-31, H-35, H-38, H-41
-maxfan_hard command option 4-18, A-9
-maxfanin command option 4-18, E-14
Maximum Filtered Instances dialog box zone
Schematic Options dialog box 3-60
Maximum Instance Ports dialog box zone
Schematic Options dialog box 3-60
Maximum Instances dialog box zone
Schematic Options dialog box 3-60
Maximum Sheet Pins dialog box zone
Schematic Options dialog box 3-60
-maxterms command option 4-18, E-14
menubar 3-2
menus 3-2
Advanced 3-17
Edit 3-16
Format 3-51
HDL Analyst 3-45
Help 3-64
menubar 3-4
Options 3-53
popup 3-73
Project view 3-73
RTL view 3-85
SCOPE window 3-84
Technology view 3-85
Text Editor window 3-84
Project 3-29
RTL 3-46, 3-47
Run 3-42
SCOPE 3-86
Select All 3-23
Technology 3-46, 3-47
View 3-22
Visual Elite Cross Probing 3-45, 3-48
Window 3-62
Mercury menu options (table) B-15
Mercury project Tcl command arguments (table) B-17
Mercury set_option Tcl command arguments (table) B-15
Microsoft Windows platform setup 5-9
Minimum Specifications for Implementing Inferred ROM (Altera) (table) B-5, B-6
miscellaneous language constructs
VHDL 9-3
mixed-level schematics, Xilinx H-6
model template, VHDL 9-34
ModelSim, crossprobing with 5-11
module name syntax, Verilog 6-21
module template, Verilog 8-9
modules, Verilog 8-9
mouse 2-10
buttons (selecting, popup menu) 2-11
wheel (scrolling and zooming) 2-11
-move command option 4-14
Move Down icon
License Editor dialog box 3-70
Move Up icon
  License Editor dialog box 3-70
Multicycle constraint example (figure) 6-14
multicycle paths 6-12
  constraint points 6-15
  examples 6-13
  syn_reference_clock attribute 7-91
Multicycle paths example (figure) 6-13
Multi-Cycle Paths panel
  Comment column 6-13
  Cycles column 6-13
  Enabled column 6-12
  From column 6-12
  SCOPE spreadsheet 6-12
  Through column 6-13
  To column 6-12
multiple constraints
  examples 6-16
  multipliers, style 7-72

N
  -name command option 4-13, 4-14, 6-25
  naming examples 6-23
  naming objects
    SCOPE spreadsheet 6-23
    VHDL 9-8
  negative slack, handling 5-21
  nested write enable (VHDL) 9-79
  net buffering report 1-29
    examples 1-29
  net name syntax
    Verilog 6-21
    VHDL 6-22
  netlist file, Verilog 1-26
  netlist file, VHDL 1-26
  netlists
    structural Verilog 1-26
    structural VHDL 1-26
  nets
    preserving in Actel designs 7-23
    preserving with syn_keep directive 7-157
  Nets panel
    Object Query dialog box 3-49
  New action button
    Toolbars dialog box 3-25
    -new command option 4-3, 4-12
    New Constraint file (SCOPE) icon
      Project toolbar 3-89
      New dialog box 3-7
        ... action button 3-8
      Add to Project zone 3-8
      File Location zone 3-8
      File Name zone 3-8
      File Type zone 3-8
      Full Path zone 3-8
      New dialog box (figure) 3-7
      New dialog box features (table) 3-8
      New File Name dialog box zone
        Copy File dialog box 3-79
      New HDL file icon
        Project toolbar 3-89
      New Implementation menu item 3-41, 3-82
        Project menu 3-30
      Project View popup menu 3-76
      New menu item
        File menu 3-5
      New Project action button
        Open Project dialog box 3-12
        New Project dialog box 3-13
        Select Project Type zone 3-13
        New Project dialog box (figure) 3-13
        New Project menu item
        File menu 3-7
        Project View popup menu 3-74
      New Window menu item
        Window menu 3-62
      New Workspace action button
        Open Project dialog box 3-12
        New Workspace menu item 3-14
        File menu 3-7
      Next Bookmark icon
      Edit toolbar 3-92
      Next bookmark menu item
      Edit menu 3-17
      Next Error menu item
      Run menu 3-44
      Next Sheet icon
      View toolbar 3-91
      Next Sheet menu item
      View menu 3-23
      Next Tip action button
        Tip of the Day dialog box 3-72
      No Buses in Technology View check box
      Schematic Options dialog box 3-60
      -no_time_gated_clock command option 4-18
Synplify Reference Manual, April 2002

Index

Normal View menu item
- View menu 3-24
- -num_critical_paths command option 4-18
- -num_startend_points command option 4-18
numeric_bit package (VHDL) 9-2
numeric_std package (VHDL) 9-2

O
Object column
- SCOPE spreadsheet 7-9
object information, HDL Analyst 5-4
object naming examples 6-23
object naming syntax
- Verilog 6-21
- VHDL 6-22
Object Query dialog box 3-49
- action button 3-50
- All action button 3-50
- action button 3-50
All -> action button 3-50
Highlight Wildcard (^?) zone 3-50
Highlighted zone 3-50
Instances panel 3-49
- Jump Location check box 3-50
Nets panel 3-49
- Ports panel 3-49
Symbols panel 3-49
Unhighlight Wildcard (^?) zone 3-50
Unhighlighted zone 3-49
Object Query dialog box (figure) 3-49
Object Query dialog box features (table) 3-49
Object Type column
- SCOPE spreadsheet 7-9
objects, declaring in VHDL 9-8
Online Documents menu item
- Help menu 3-65
Open action button
- Open dialog box 3-10
Open as read only check box
- Open dialog box 3-10
Open as Text menu item
- Project View popup menu 3-75, 3-76, 3-77
open book icon (online help) 3-66
Open dialog box 3-9, 3-44
- File name zone 3-9
- Files of type zone 3-10
- Look in zone 3-9
- Open action button 3-10
Open as read only zone 3-10
Open dialog box (figure) 3-9, 3-44
Open dialog box features (table) 3-9
Open icon
- Project toolbar 3-89
Open menu item
- File menu 3-5
- Project View popup menu 3-74, 3-75, 3-76
Open Project dialog box 3-12, 3-83
- Existing Project action button 3-12
- New Project action button 3-12
- New Workspace action button 3-12
- Project Wizard action button 3-12
- Recent Projects zone 3-12
Open Project dialog box (figure) 3-12
Open Project dialog box features (table) 3-12
Open project icon
- Project toolbar 3-89
Open Project menu item 3-83
- File menu 3-6
- Project View popup menu 3-74
open-drain mode (Altera) 7-37
Opening a SCOPE constraint file (figure) 6-4
opening constraint file with SCOPE spreadsheet 6-4
opening windows, HDL Analyst 5-2
operators
- selected name support (VHDL) 9-29
sharing in VHDL CASE statements 9-23
VHDL 9-15
optimization
- for area 1-19
- state machines 1-21
options
- Project view 3-97
- Resource Sharing 7-172
text editor 3-56
See also command options
Options- > Quartus submenu items
- (table) B-19
Options action button
- Editor Options dialog box 3-57
Options for implementation dialog box 3-31, 3-41, 3-54, 3-82
Device panel 3-32
Implementation Results panel 1-26, 3-33
Options/Constraints panel 1-17, 3-33
Timing Report panel 3-35, 6-46
Options for implementation dialog box (continued)
Verilog panel 3-36
   Include Path Order zone 3-36
   Push Tristates across Process/Block boundaries check box 3-37
Verilog 2001 check box 3-37
VHDL panel 3-36
   Default Enum Encoding zone 3-37
   Push Tristates across Process/Block boundaries check box 3-37

Options for implementation dialog box (figure) 3-31, 3-42
Options menu 3-53
   commands 3-54
      Configure Verilog Compiler item 3-54, 3-55
      Configure VHDL Compiler item 3-54
      Customize item 3-54, 3-55
dialog boxes 3-54
Editor Options item 3-54, 3-56
Project View Options item 3-54, 3-55
Quartus submenu (Altera) 3-54
   Background Compile item B-19
   Foreground Compile item B-19
   Set Options item B-19
Schematic Options item 3-54, 3-59
Xilinx item 3-54
Options menu commands (table) 3-54
Options/Constraints panel
   Options for implementation dialog box 1-17, 3-33
Options/Constraints panel, Options for Implementation dialog box (figure) 3-33
ORCA Foundry (Lattice) E-2
orca_padtype attribute (Lattice) 7-45
   examples 7-45
orca.props attribute (Lattice) 7-47
   examples 7-47
organizing views 2-13
Original File Name dialog box zone
   Copy File dialog box 3-79
Other panel
   Argument column 6-17
   Command column 6-17
   Comment column 6-17
   Enabled column 6-17
   SCOPE spreadsheet 6-17
Other panel (table) 6-17
Other VHDL operators (table) 9-17
output drivers
   slowing transition times 7-136
   speeding transition time 7-119
output files
   See files, output

P
p keyboard shortcut 3-96
 -package command
   option 4-16, A-9, B-15, B-21, B-27, E-11
packages, VHDL 9-11
   packing registers into I/O cells, Altera B-17
   pad locations
   See also pin locations
   QuickLogic 7-51
   pad types
      Lattice ORCA 7-45
      QuickLogic 7-49
      Xilinx 7-129
page number header and footer variable
   (printing) 3-15
Page Setup dialog box 3-15
Page Setup dialog box (figure) 3-15
Page Setup header and footer variables
   (table) 3-15
Page Setup menu item 3-15
   File menu 3-6
   Pan Center menu item
      View menu 3-23
   Pan menu item
      View menu 3-23
   panels
      Attributes 6-17, 7-9
      Clocks 6-6
      Contents 3-66
      Customize 3-26
      Device 3-32
      False Paths 6-14
      Find 3-67
      Implementation Results 1-26, 3-33
      Index 3-67
      Inputs/Outputs 6-10, 6-11
      Instances 3-49
      Multi-Cycle Paths 6-12
      Nets 3-49
      Options/Constraints 1-17, 3-33

Index-34
Synpify Reference Manual, April 2002
panels (continued)
  Other 6-17
  Ports 3-49
  Registers 6-11
  Symbols 3-49
  Timing Report 3-35, 6-46
  Toolbars 3-26
  Verilog 3-36
  VHDL 3-36
parallel_case directive (Verilog) 7-147
example 7-147
-part command
  option 4-15, A-8, B-15, B-21, B-27, B-3
  3, C-4,
  E-6, E-11, E-13, F-6, G-3, H-28, H-31, H-
  35, H-38,
  H-41
Paste icon
  Project toolbar 3-89
Paste menu item
  Edit menu 3-17, 3-18
Performance summary (table) 6-47
performance summary, timing report 6-47
Period column
  SCOPE spreadsheet 6-6
-period command option 6-25
physical synthesis
  See Amplify
pin locations
  Actel 7-21
  Altera 7-31
  Lattice 7-43, 7-44
  Xilinx, automatic insertion H-22
-pipe command option H-28
pipelining
  syn_pipeline attribute 7-78
platform setup
  Microsoft Windows 5-9
  Unix 5-10
popup (tooltip) information, HDL Analyst 5-4
popup menu dialog boxes, Project View 3-80
popup menus 3-2, 3-73
  Project view 3-73
  RTL view 3-85
  SCOPE window 3-84
  Technology view 3-85
  Text Editor window 3-84
Port column
  SCOPE spreadsheet 6-11
port definitions, VHDL 9-9
port locations, Xilinx 7-124
port name syntax
  Verilog 6-21
  VHDL 6-22
portprop Tcl command (QuickLogic) F-9
ports and signals, VHDL 9-9
Ports panel
  Object Query dialog box 3-49
predefined enumeration types, VHDL 9-5
predefined packages, VHDL 9-12
Predefined VHDL libraries and packages
  (table) 9-12
preferences
  colors in HDL Analyst 2-14
  display of project files 2-12
  Project view display 2-12
  Text Editor window 2-14
  preferred license selection 1-2
Preferred License Selection menu item 3-71
Help menu 3-65
PREP benchmarks
  Verilog 8-42
  VHDL 9-91
preserving instances with syn_noprune
  directive 7-162
preserving macros with syn_macro
  directive 7-161
preserving nets in Actel designs 7-23
preserving nets with syn_keep directive 7-157
preserving registers for simulation with
  syn_preserve directive 7-167
Preview dialog box zone, Display Settings dia-
  log box 3-28
Previous Bookmark icon
  Edit toolbar 3-92
Previous bookmark menu item
  Edit menu 3-17
Previous Error menu item
  Run menu 3-44
Previous Sheet icon
  View toolbar 3-91
Previous Sheet menu item
  View menu 3-23
Previous Tip action button
  Tip of the Day dialog box 3-72
primitives, Verilog 8-14
Print dialog box 3-14
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Print dialog box (figure) | 3-14 |
| Print icon |
| Project toolbar | 3-89 |
| Print Image menu item |
| File menu | 3-7 |
| Print menu item |
| File menu | 3-6 |
| Print Preview menu item | 3-14 |
| File menu | 3-6 |
| Print Setup dialog box | 3-16 |
| Print Setup dialog box (figure) | 3-16 |
| Print Setup menu item | 3-16 |
| File menu | 3-6 |
| printing a region of a graphic view | 3-7 |
| prj file | 1-23 |
| process template |
| modeling combinational logic |
| (VHDL) | 9-18 |
| VHDL | 9-36 |
| process VHDL keyword | 9-18 |
| processes, VHDL | 9-33 |
| project command options (table) | 4-12 |
| Project File Name Display dialog box zone | 2-13 |
| Project Name zone | 3-80 |
| Project Properties dialog box (figure) | 3-80 |
| project Tcl command | 4-3, 4-12 |
| Altera |
| ACEX | B-28 |
| APEX | B-16 |
| APEX II | B-16 |
| Excalibur | B-16 |
| FLEX | B-28 |
| MAX | B-34 |
| Mercury | B-16 |
| Stratix | B-21 |
| examples of use | 4-13 |
| Lattice |
| MACH | E-14 |
| Xilinx |
| CoolRunner | H-41 |
| CoolRunner-II CPLDs | H-41 |
| CPLDs | H-41 |
| Spartan | H-28 |
| Virtex | H-31 |
| XC3000, XC5200 | H-38 |
| XC4000 | H-35 |
| XC9500 | H-41 |
| Project toolbar | 3-88 |
| Cut icon | 3-89 |
| Find icon | 3-89 |
| New Constraint file (SCOPE) icon | 3-89 |
| New HDL file icon | 3-89 |
| Open icon | 3-89 |
| Open project icon | 3-89 |
| Paste icon | 3-89 |
| Print icon | 3-89 |
| Redo icon | 3-89 |
| Save all icon | 3-89 |
| Save icon | 3-89 |
| Undo icon | 3-89 |
| Project toolbar (figure) | 3-88 |
| Project toolbar icons (table) | 3-89 |
| Project view |
| 2-2 |
| action buttons | 3-97 |
| dialog boxes | 3-78 |
| Frequency (MHz) zone | 3-98 |
| options | 3-97 |
| Resource Sharing check box | 3-98 |
| Run check box | 3-98 |
| setting up | 3-55 |
| Symbolic FSM Compiler check box | 3-98 |
| Project view (figure) | 2-3 |

Index-36

Synplify Reference Manual, April 2002
Index

Project view action buttons (figure) 3-97
Project view action buttons and options (table) 3-98
Project view menubar (figure) 3-2
Project View Options dialog box 2-12, 3-55, 3-84
   Beep when a job completes check box 3-56
   Project File Name Display zone 3-55
   Project Window zone 3-55
   Show file name only radio button 3-56
   Show full file path radio button 3-56
   Show Project File Library check box 3-56
   Show relative file path radio button 3-56
   View project files in folders check box 3-56
Project View Options dialog box (figure) 3-55
Project View Options dialog box features (table) 3-56
Project View Options menu item
   Options menu 3-54, 3-55
   Project View popup menu 3-74
   Project View popup menu 3-84
Project View popup menu
   Add Source File item 3-76, 3-82
   Build Workspace item 3-74, 3-83
   Change File item 3-75
   Change Implementation Name item 3-78
   Close Project item 3-77
   Close Workspace item 3-77
   command summary 3-85
   commands 3-74
   Compile Only item 3-76
   Copy File item 3-75
   Copy Implementation item 3-78
   Delete File item 3-76
   File Options item 3-74, 3-75, 3-76, 3-78
   Implementation Options item 3-78
   Insert Project item 3-77, 3-80
   New Implementation item 3-76, 3-82
   New Project item 3-74
   Open as Text item 3-75, 3-76, 3-77
   Open item 3-74, 3-75, 3-76
   Open Project item 3-74, 3-83
   Project Options item 3-76, 3-80
   Project View Options item 3-74, 3-84
   Refresh item 3-74
Remove File from Project item 3-75
Remove Implementation item 3-78
Remove Project From Workspace item 3-77
Run All Projects item 3-77
Run item 3-78
Save Project item 3-77
Save Workspace item 3-77, 3-84
Syntax Check item 3-74
Synthesis Check item 3-75
Synthesize item 3-76
Workspace Options item 3-77, 3-80
Project view popup menu commands, constraint file selected (table) 3-75
Project view popup menu commands, HDL source file selected (table) 3-74
Project view popup menu commands, implementation selected (table) 3-78
Project view popup menu commands, nothing selected (table) 3-74
Project view popup menu commands, project selected (table) 3-76
Project view popup menu commands, results file selected (table) 3-76
Project view popup menu commands, workspace selected (table) 3-77
Project window 2-2
Project Wizard action button
   Open Project dialog box 3-12
   Project Wizard dialog box 3-13
project_file command options (table) 4-14
project_file Tcl command 4-14
   examples of use 4-15
Projects dialog box zone
   Build Workspace dialog box 3-83
Properties menu item
   View menu 3-24
   properties, SCOPE spreadsheet display 3-28
Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Push Tristates across Process/Block boundaries check box
Options for implementation dialog box 3-37
Push/Pop Hierarchy icon
View toolbar 3-91
Push/Pop Hierarchy menu item
View menu 3-23
Push/Pop mode, HDL Analyst 5-12, 5-13

set_option Tcl command F-6
syn_useioff attribute 7-112
synthesis command file 7-112
synthesis commands F-9
QuickLogic attributes and directives (table) F-7
QuickLogic set_option command options (table) F-6
quitting a synthesis run 3-45

Radiation level 7-85
radio buttons
Absolute Path 3-78
Edit or enter license information received from Synplicity 3-67
External Editor 3-57
Fill out a form to request a trial license from Synplicity radio button 3-67
Internal Editor 3-57
Relative to Project 3-78
Show file name only 2-13, 3-56
Show full file path 2-13, 3-56
Show relative file 2-13
Show relative file path 3-56

RAM
additional components 8-36
altera_auto_use_eab attribute 7-25
altera_auto_use_esb 7-28
implementation in Altera B-5
inference in Verilog 8-34
limited resources (Verilog) 8-34
syn_ramstyle attribute 7-87
with special write enable 9-78
VHDL 9-78
with special write enables
Verilog 8-35
RAM inference
additional logic (VHDL) 9-81
Altera B-4
Stratix B-22
synchronous RAM (VHDL) 9-81
VHDL 9-77
working with limited RAM resources (VHDL) 9-80
Xilinx H-10
RAM inference in Altera Stratix (table) B-22
ReadOnly menu item
View menu 3-24

Index-38
Synplify Reference Manual, April 2002
<table>
<thead>
<tr>
<th>Page</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-6</td>
<td>recent files list, File menu</td>
</tr>
<tr>
<td>3-12</td>
<td>Recent Projects dialog box zone</td>
</tr>
<tr>
<td>3-6</td>
<td>Open Project dialog box</td>
</tr>
<tr>
<td>3-6</td>
<td>Recent Projects menu item</td>
</tr>
<tr>
<td>3-6</td>
<td>File menu</td>
</tr>
<tr>
<td>3-6</td>
<td>recommended reading</td>
</tr>
<tr>
<td>1-25</td>
<td>Verilog</td>
</tr>
<tr>
<td>1-24</td>
<td>VHDL</td>
</tr>
<tr>
<td>3-87</td>
<td>SCOPE Attributes dialog box</td>
</tr>
<tr>
<td>3-89</td>
<td>Redo icon</td>
</tr>
<tr>
<td>3-17</td>
<td>Edit menu</td>
</tr>
<tr>
<td>3-18</td>
<td>-ref command option</td>
</tr>
<tr>
<td>3-74</td>
<td>Refresh menu item</td>
</tr>
<tr>
<td>3-19</td>
<td>Select Files to Add to Project dialog box</td>
</tr>
<tr>
<td>3-40</td>
<td>Select Projects to Include in Workspace dialog box</td>
</tr>
<tr>
<td>3-78</td>
<td>Remove All -&gt; action button</td>
</tr>
<tr>
<td>4-14</td>
<td>Remove command option</td>
</tr>
<tr>
<td>3-51</td>
<td>Remove Covering menu item</td>
</tr>
<tr>
<td>3-75</td>
<td>Remove File from Project menu item</td>
</tr>
<tr>
<td>3-77</td>
<td>Remove Row action button</td>
</tr>
<tr>
<td>3-78</td>
<td>Remove Rows menu item</td>
</tr>
<tr>
<td>3-82</td>
<td>Remove Projects from Workspace menu item</td>
</tr>
<tr>
<td>6-12</td>
<td>SCOPE spreadsheet</td>
</tr>
<tr>
<td>6-12</td>
<td>Register column</td>
</tr>
<tr>
<td>6-12</td>
<td>SCOPE spreadsheet (table)</td>
</tr>
<tr>
<td>6-12</td>
<td>Replace dialog box (figure)</td>
</tr>
<tr>
<td>6-12</td>
<td>Replace dialog box features (table)</td>
</tr>
<tr>
<td>6-19</td>
<td>Replace dialog box (figure)</td>
</tr>
<tr>
<td>6-21</td>
<td>Replace with dialog box</td>
</tr>
<tr>
<td>7-108</td>
<td>Altera syn_useioff attribute</td>
</tr>
<tr>
<td>7-110</td>
<td>Lattice ORCA syn_useioff attribute</td>
</tr>
<tr>
<td>7-112</td>
<td>QuickLogic syn_useioff attribute</td>
</tr>
<tr>
<td>7-114</td>
<td>Xilinx syn_useioff attribute</td>
</tr>
<tr>
<td>6-12</td>
<td>Registers panel</td>
</tr>
<tr>
<td>6-12</td>
<td>Enabled column</td>
</tr>
<tr>
<td>6-12</td>
<td>Register column</td>
</tr>
<tr>
<td>6-12</td>
<td>Route column</td>
</tr>
<tr>
<td>6-11</td>
<td>SCOPE spreadsheet</td>
</tr>
<tr>
<td>6-12</td>
<td>Type column</td>
</tr>
<tr>
<td>7-14</td>
<td>Relative to Project radio button</td>
</tr>
<tr>
<td>6-43</td>
<td>relaxing forward-annotated constraints</td>
</tr>
<tr>
<td>4-14</td>
<td>Remove All -&gt; action button</td>
</tr>
<tr>
<td>4-14</td>
<td>Remove command option</td>
</tr>
<tr>
<td>3-40</td>
<td>Select Files to Add to Project dialog box</td>
</tr>
<tr>
<td>3-82</td>
<td>Remove Projects from Workspace dialog box</td>
</tr>
<tr>
<td>3-75</td>
<td>Remove Files From Project menu item</td>
</tr>
<tr>
<td>3-77</td>
<td>Remove Row action button</td>
</tr>
<tr>
<td>3-78</td>
<td>Remove Rows menu item</td>
</tr>
<tr>
<td>3-19</td>
<td>Match Case check box</td>
</tr>
<tr>
<td>3-21</td>
<td>Replace action button</td>
</tr>
<tr>
<td>3-21</td>
<td>Replace All action button</td>
</tr>
<tr>
<td>3-21</td>
<td>Replace file options dialog box</td>
</tr>
<tr>
<td>9-36</td>
<td>registers, VHDL</td>
</tr>
<tr>
<td>3-20</td>
<td>related clocks</td>
</tr>
<tr>
<td>9-36</td>
<td>Regular expression check box</td>
</tr>
<tr>
<td>6-16</td>
<td>relational operators, VHDL</td>
</tr>
<tr>
<td>7-19</td>
<td>relative location</td>
</tr>
<tr>
<td>7-125</td>
<td>alsloc (Actel)</td>
</tr>
<tr>
<td>7-134</td>
<td>xc_map (Xilinx)</td>
</tr>
<tr>
<td>7-137</td>
<td>xc_rloc (Xilinx)</td>
</tr>
<tr>
<td>7-142</td>
<td>xc_uset (Xilinx)</td>
</tr>
<tr>
<td>7-21</td>
<td>replication, disabling</td>
</tr>
<tr>
<td>4-19</td>
<td>-report_path command option</td>
</tr>
<tr>
<td>A-9</td>
<td>-report_path command option</td>
</tr>
<tr>
<td>H-14</td>
<td>(table)</td>
</tr>
<tr>
<td>H-25</td>
<td>relative position, Xilinx components</td>
</tr>
<tr>
<td>Reports</td>
<td>VHDL 9-82</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Actel  A-3</td>
<td>ROM tables 8-38, 9-82</td>
</tr>
<tr>
<td>Altera  B-11</td>
<td>HDL Analyst viewer 8-38, 9-82</td>
</tr>
<tr>
<td>Lattice  E-10</td>
<td>Verilog 8-38</td>
</tr>
<tr>
<td>net buffering 1-29</td>
<td>rom.info file 8-38</td>
</tr>
<tr>
<td>QuickLogic  F-5</td>
<td>Route column</td>
</tr>
<tr>
<td>resource usage 1-29</td>
<td>SCOPE spreadsheet 6-7, 6-11, 6-12</td>
</tr>
<tr>
<td>Xilinx  H-9, H-32</td>
<td>-route command</td>
</tr>
<tr>
<td>Reset action button</td>
<td>option 1-18, 6-26, 6-27, 6-28, 6-29, 6-30, 6-31</td>
</tr>
<tr>
<td>Toolbars dialog box 3-25</td>
<td>define_clock 6-26</td>
</tr>
<tr>
<td>resets, Verilog 8-24, 8-26</td>
<td>define_input_delay 6-29, 6-30</td>
</tr>
<tr>
<td>Resize Columns menu item</td>
<td>define_output_delay 6-31</td>
</tr>
<tr>
<td>Format menu 3-51</td>
<td>define_reg_input_delay 6-27, 6-28</td>
</tr>
<tr>
<td>Resize Rows menu item</td>
<td>define_reg_output_delay 6-28</td>
</tr>
<tr>
<td>Format menu 3-51</td>
<td>RTL submenu</td>
</tr>
<tr>
<td>resource sharing</td>
<td>HDL Analyst menu 3-46, 3-47</td>
</tr>
<tr>
<td>operators in VHDL CASE</td>
<td>RTL view 2-4, 4-22, 5-2</td>
</tr>
<tr>
<td>statements 9-23</td>
<td>customizing 2-14</td>
</tr>
<tr>
<td>syn_sharing 7-172</td>
<td>file 1-27</td>
</tr>
<tr>
<td>VHDL 9-23</td>
<td>popup menu 3-85</td>
</tr>
<tr>
<td>Resource Sharing check box</td>
<td>primitives (VHDL) 9-46</td>
</tr>
<tr>
<td>Project view 3-98, 7-172</td>
<td>Verilog primitives 8-26</td>
</tr>
<tr>
<td>resource usage report 1-29</td>
<td>RTL view (figure) 2-4</td>
</tr>
<tr>
<td>examples</td>
<td>RTL view dialog boxes 3-86</td>
</tr>
<tr>
<td>Altera  B-12</td>
<td>RTL View icon</td>
</tr>
<tr>
<td>-resource_sharing command option 4-19</td>
<td>Analyst toolbar 3-90</td>
</tr>
<tr>
<td>-result_file command</td>
<td>RTL View popup menu</td>
</tr>
<tr>
<td>option 4-13, 4-19, B-17, B-22, B-28, B-34, H-29, H-31, H-36, H-39, H-42</td>
<td>Dissolve Selected Instances item 3-85</td>
</tr>
<tr>
<td>-result_format command</td>
<td>SCOPE submenu 3-86</td>
</tr>
<tr>
<td>option 4-13, B-17, B-28, E-14, H-29, H-31, H-36, H-39, H-42</td>
<td>Edit Attributes item 3-86</td>
</tr>
<tr>
<td>results</td>
<td>View FSM Info File item 3-86</td>
</tr>
<tr>
<td>faster 1-17</td>
<td>Run All Projects menu item</td>
</tr>
<tr>
<td>smaller 1-19</td>
<td>Project View popup menu 3-77</td>
</tr>
<tr>
<td>-retiming command option H-28</td>
<td>Run check box</td>
</tr>
<tr>
<td>RGB values, defining for HDL Analyst 2-14</td>
<td>Project view 3-98</td>
</tr>
<tr>
<td>rise and fall timing constraints 6-9</td>
<td>-run command option 4-3, 4-12</td>
</tr>
<tr>
<td>Rise At column</td>
<td>Run menu 3-42</td>
</tr>
<tr>
<td>SCOPE spreadsheet 6-7</td>
<td>commands 3-43</td>
</tr>
<tr>
<td>-rise command option 6-26</td>
<td>Compile Only item 3-43</td>
</tr>
<tr>
<td>ROM</td>
<td>dialog boxes 3-44</td>
</tr>
<tr>
<td>altera_auto_use_eab attribute 7-25</td>
<td>Job Status item 3-44, 3-45</td>
</tr>
<tr>
<td>altera_auto_use_esb 7-28</td>
<td>Next Error item 3-44</td>
</tr>
<tr>
<td>syn_romstyle (Xilinx) 7-100</td>
<td>Previous Error item 3-44</td>
</tr>
<tr>
<td>syn_romstyle attribute (Altera) 7-97</td>
<td>Run Tcl Script item 3-43, 3-44, 4-7</td>
</tr>
<tr>
<td>ROM inference</td>
<td>Syntax Check item 3-43</td>
</tr>
<tr>
<td>Altera  B-6</td>
<td>Synthesis Check item 3-43</td>
</tr>
<tr>
<td>Verilog 8-38</td>
<td>Synthesize item 3-43</td>
</tr>
<tr>
<td>Run menu commands (table) 3-43</td>
<td></td>
</tr>
</tbody>
</table>
Run menu item
  Project View popup menu 3-78
  Tools menu (Visual Elite) 5-10
  Run Synplify dialog box (Visual Elite) 5-10
  Run Tcl Script menu item 3-44, 4-7
  Run menu 3-43
  Run Time dialog box zone
    Job Status dialog box 3-45
    running Tcl scripts 4-5
    running the synthesis tool 1-14

S
Save action button
  License Editor dialog box 3-71
  Save As dialog box 3-11
  Select Preferred License dialog box 3-71
Save all icon
  Project toolbar 3-89
Save All menu item
  File menu 3-6
  Save As dialog box 3-10
    File name zone 3-11
    Save action button 3-11
    Save as type zone 3-11
    Save in zone 3-11
  Save As dialog box (figure) 3-10
  Save As dialog box features (table) 3-11
Save As menu item
  File menu 3-6
  Save as type dialog box zone
    Save As dialog box 3-11
    -save command option 4-13
  Save file dialog box zone
    File Options dialog box 3-78
Save icon
  Project toolbar 3-89
Save in dialog box zone
  Save As dialog box 3-11
Save menu item
  File menu 3-5
  Save Project As dialog box 3-84
Save Project menu item
  File menu 3-7
  Project View popup menu 3-77
Save Workspace menu item 3-84
  Project View popup menu 3-77
scalable architecture, using (VHDL) 9-73
scalable designs
  VHDL 9-71
scaling by overriding a parameter value
  with # (Verilog example) 8-13
  with defparam (Verilog example) 8-12
Schematic Options dialog box 3-59
  Change action button 3-61
  Compress Buses check box 3-60
  Enhanced Text Crossprobing check box 3-60
  Font Options dialog box zone 3-59
  Font zone 3-61
  General Options zone 3-59
  Maximum Filtered Instances zone 3-60
  Maximum Instance Ports zone 3-60
  Maximum Instances zone 3-60
  Maximum Sheet Pins zone 3-60
  No Buses in Technology View check box 3-60
  Sheet Size Options zone 3-59
  Show Cell Interior check box 3-60
  Show Conn Name check box 3-60
  Show Hierarchy Browser check box 3-60
  Show Instance Name check box 3-60
  Show Pin Name check box 3-60
  Show Port Name check box 3-60
  Show Sheet Connector Index check box 3-60
  Show Symbol Name check box 3-60
  Show Text check box 3-60
  Size zone 3-61
  Text Label Options zone 3-59
  Schematic Options dialog box (figure) 3-59
  Schematic Options dialog box features (table) 3-60
  Schematic Options menu item 3-59
  Options menu 3-54
schematics
  filtering 5-17
  multi-sheet 5-15
  page size 2-15
SCOPE Attribute fields (table) 7-9
SCOPE Attributes dialog box 3-86, 7-12
  Add Row action button 3-87
  Attribute zone 3-86
  Comment zone 3-87
  Enable check boxes 3-86
  Redo action button 3-87
SCOPE Attributes dialog box (continued)
  Remove Row action button 3-87
  Undo action button 3-87
  Value zone 3-86
SCOPE Attributes dialog box (figure) 3-86
SCOPE Attributes dialog box features
  (table) 3-86
SCOPE Attributes panel (figure) 7-10
SCOPE Attributes wizard 7-12
SCOPE Clocks panel (table) 6-6
SCOPE False Paths panel (table) 6-14
SCOPE Inputs/Outputs panel (table) 6-11
SCOPE Multi-Cycle Paths panel (table) 6-12
SCOPE panel constraints (table) 6-5
SCOPE panel wizards (table) 6-19, 7-14
SCOPE Registers panel (table) 6-12
SCOPE spreadsheet 6-3
  Attributes panel 6-17, 7-9
    Attribute column 7-10
    Comment column 7-10
    Description column 7-10
    Enabled column 7-9
    Object column 7-9
    Object Type column 7-9
    Val Type column 7-10
    Value column 7-10
    wizard 7-12
  changing column headings 3-52
  changing display properties 3-28
Clocks panel 6-6
  Clock column 6-6
  Clock Group column 6-6
  Comment column 6-7
  Duty Cycle column 6-7
  Enabled column 6-6
  Fall At column 6-7
  Frequency column 6-6
  Period column 6-6
  Rise At column 6-7
  Route column 6-7
  Virtual Clock column 6-7
  wizard 6-19, 7-14
constraint types 6-5
constraint wizards 6-18
False Paths panel 6-14
  Comment column 6-15
  Enabled column 6-14
  From column 6-14
  Through column 6-15
  To column 6-15
file conversions 6-17
Inputs/Outputs panel 6-10, 6-11
  Clock Edge column 6-11
  Comment column 6-11
  Enabled column 6-11
  Port column 6-11
  Route column 6-11
  Type column 6-11
  Value column 6-11
multicycle paths 6-12
Multi-Cycle Paths panel 6-12
  Comment column 6-13
  Cycles column 6-13
  Enabled column 6-12
  From column 6-12
  Through column 6-13
  To column 6-12
naming objects 6-23
object naming 6-21
opening constraint file 6-4
Other panel 6-17
  Argument column 6-17
  Command column 6-17
  Comment column 6-17
  Enabled column 6-17
panel wizards 6-19, 7-14
panels 6-5
popup menu 3-84
Registers panel 6-11
  Comment column 6-12
  Enabled column 6-12
  Register column 6-12
  Route column 6-12
  Type column 6-12
setting up 3-51
specifying attributes 7-9, 7-10
starting 6-4
timing constraints 6-3
wizards 6-19, 7-13
SCOPE submenu
  RTL View popup menu 3-86
  Technology View popup menu 3-86
SCOPE window 6-3
Script menu item
  Synthesis menu (Visual Elite) 5-10
sdc file 1-25, 6-2, 6-20, 8-47
Select All menu item
  Edit menu 3-17
Select All submenu
  View menu 3-23
Select Editor dialog box zone 3-56
Select element list dialog box 3-61
Select Files to Add to Project dialog box 3-11, 3-38, 3-82
  <- Add action button 3-40
  <- Add All action button 3-40
File name zone 3-39
Files of type zone 3-40
Files to Add to Project zone 3-40
Look in zone 3-39
Remove All -> action button 3-40
Remove -> action button 3-40
Select Files to Add to Project dialog box (figure) 3-39
Select Files to Add to Project dialog box features (table) 3-39
Select From menu item 3-61
Select Net Driver menu item
  View menu 3-23
Select Net Instances menu item
  View menu 3-23
Select Preferred License dialog box 3-71
  License Type dialog box zone 3-71
  Save action button 3-71
Select Preferred License dialog box (figure) 3-71
Select Project Type dialog box zone
  New Project dialog box 3-13
Select Projects to Include in Workspace dialog box 3-14, 3-80, 3-81
  <- Add action button 3-82
  <- Add All action button 3-82
File name zone 3-81
Files of type zone 3-82
Files To Add To Project zone 3-82
Look in zone 3-81
Remove All -> action button 3-82
Remove -> action button 3-82
Select Projects to Include in Workspace dialog box (figure) 3-81
Select Projects to Include in Workspace dialog box features (table) 3-81
selected name support (SNS) 9-26
Selected Name Support, VHDL 9-26
selecting a text column (Text Editor) 2-9
selecting objects, HDL Analyst 5-6
selecting synthesis style, Max+Plus II (MAX) B-35
sensitivity list (VHDL) 9-18
sequential logic
  Verilog 8-19, 8-20, 8-21, 8-24, 8-26
  examples 8-19
  VHDL 9-24
  examples 9-90
sequential optimization, preventing with syn_preserve directive 7-167
sequential shift components
  syn_srlstyle attribute 7-102
  Verilog H-17
  VHDL H-15
set and reset signals, VHDL 9-44
Set Options menu item
  Options menu, Quartus submenu
    (Altera) B-19
Set Slack Margin menu item
  HDL Analyst menu 3-47
Set VHDL Library menu item 3-41
  Project menu 3-30
set/reset priority 7-81
set_option and get_option Tcl command options (table) 4-16
set_option Tcl command 4-3, 4-4, 4-6, 4-15
Actel A-8
Altera
  ACEX B-26, B-27
  APEX B-15
  APEX II B-15
  Excalibur B-15
  FLEX B-26
  MAX B-33
  Mercury B-15
  Stratix B-20
Atmel C-4
Cypress D-11
Lattice
  GAL/isp E-11
  MACH E-13
  ORCA E-6
QuickLogic F-6
Triscend G-3
set_option Tcl command (continued)
  Xilinx
    CoolRunner H-40
    CoolRunner-II CPLDs H-40
    CPLDs H-40
    Spartan H-27
    Virtex H-30
    XC3000, XC5200 H-37
    XC4000 H-34
    XC9500 H-40

set_option Tcl command options: technology, part, speed grade (table) 4-15

sets and resets
  Verilog 8-24, 8-26
  VHDL 9-44

setting colors, HDL Analyst 5-7

setting options and timing constraints 4-23

setting Project view display preferences 2-12
setting Text Editor window preferences 2-14

setting up HDL Analyst 5-4

Sheet connector (figure) 5-15

Sheets, viewing other 5-15

Shift keyboard shortcut (scrolling with mouse) 2-11

Shift operators, VHDL 9-17

Shift-Alt-F2 keyboard shortcut 3-95
Shift-Alt-i keyboard shortcut 3-95
Shift-Alt-n keyboard shortcut 3-95
Shift-Alt-p keyboard shortcut 3-95
Shift-e (E) keyboard shortcut 3-95
Shift-F11 keyboard shortcut 3-97
Shift-F2 keyboard shortcut 3-96
Shift-F5 keyboard shortcut 3-96
Shift-F7 keyboard shortcut 3-96
Shift-F8 keyboard shortcut 3-96
Shift-Left Arrow keyboard shortcut 3-97
Shift-Right Arrow keyboard shortcut 3-97
Shift-Tab keyboard shortcut (Text Editor) 2-8

shortcuts, keyboard
  See keyboard shortcuts

Show Cell Interior check box
  Schematic Options dialog box 3-60

Show Conn Name check box
  Schematic Options dialog box 3-60

Show Critical Path icon 5-18, 5-19

Analyst toolbar 3-90

Show Critical Path menu item
  HDL Analyst menu 3-47

Show file name only radio button 2-13

  Project View Options dialog box 3-56

Show full file path radio button 2-13

  Project View Options dialog box 3-56

Show Hierarchy Browser check box
  Schematic Options dialog box 3-60

Show Instance Name check box
  Schematic Options dialog box 3-60

Show Line Numbers menu item
  View menu 3-24

Show Pin Name check box
  Schematic Options dialog box 3-60

Show Port Name check box
  Schematic Options dialog box 3-60

Show Project File Library check box
  Project View Options dialog box 3-56

Show Project Library check box 2-13

Show relative file path radio button 2-13

  Project View Options dialog box 3-56

Show Sheet Connector Index check box
  Schematic Options dialog box 3-60

Show Symbol Name check box
  Schematic Options dialog box 3-60

Show Text check box
  Schematic Options dialog box 3-60

Show Timing Information menu item
  HDL Analyst menu 3-47

Show Tips at Startup check box
  Tip of the Day dialog box 3-72

Show Tooltips check box
  Toolbars dialog box 3-26

signal assignments
  concurrent (VHDL) 9-21
  conditional (VHDL) 9-23
  selected (VHDL) 9-22
  simple (VHDL) 9-22
  VHDL 9-9

signals and ports, VHDL 9-9

signed arithmetic 9-7

signed signal examples, Verilog 2001 8-6

simple component instantiation, VHDL 9-25

Simple gates (table) 8-14
simple gates, Verilog 8-14
simulation (post-synthesis) in Altera APEX 8-18
simulation using enumerated types, VHDL 9-53
Size dialog box zone
  Schematic Options dialog box 3-61
slack 6-47
  analysis 5-18
  handling 5-21
  on registers 6-27
  setting margins 5-20
SNS
  constants (VHDL) 9-27
  demand loading (VHDL) 9-32
  functions and operators (VHDL) 9-28
  user-defined function support (VHDL) 9-30
  -soft command option 4-19
  -soft_buffers command option 4-19, B-33
Software Authorization form, Synplicity 3-70
source code black-box directives 6-36
Source File dialog box 3-40
  Change action button 3-40
  File name zone 3-40
  Files of type zone 3-40
  Look in zone 3-40
Source File dialog box (figure) 3-41
source files, adding to VHDL design library 9-11
Spartan (Xilinx) H-27
Spartan project Tcl command options (table) H-29
Spartan set_option Tcl command options (table) H-28
specifying attributes 7-9
Specifying constraint files (figure) 7-11
specifying directives and attributes 7-14
Specifying false paths (figure) 6-15
Specifying I/O locations in Altera B-44
specifying timing constraints 6-2
  -speed_grade command
srm file 1-27
srr file 1-27, 4-23
srs file 1-27
standard package (VHDL) 9-12
starting in batch mode
  using a project file 4-4, 4-6
  using a Tcl script 4-7
starting points, multicycle and false paths 6-15
starting the Text Editor 2-7
startup block, Xilinx H-4
state machine guidelines
  Verilog 8-29
  VHDL 9-49
state machines
  asynchronous
    See asynchronous state machines
  encoding file 1-30
  optimization 1-21
  setting enumerated type in VHDL 9-52
  Verilog 8-29, 8-31, 8-32
  VHDL 9-49
state values (FSM), Verilog 8-32
Status Bar menu item
  View menu 3-22
status bar, HDL Analyst 5-5
std library (VHDL) 9-12
std package (VHDL) 9-2
std_logic enumeration data type (VHDL) 9-6
std_logic_1164 IEEE package (VHDL) 9-2
std_logic_1164 package (VHDL) 9-12
std_logic_arith package (VHDL) 9-2, 9-13
std_logic_signed package (VHDL) 9-2, 9-13
std_logic_unsigned package (VHDL) 9-2, 9-13
stopping a synthesis run 3-45
Stratix menu options (table) B-20
Stratix project Tcl command arguments (table) B-22
Stratix set_option Tcl command arguments (table) B-21
structural designs, Verilog 8-43
structural Verilog netlist file 1-26
structural VHDL netlist file 1-26
Style menu item 3-51
Styles dialog box 3-52
Styles dialog box (figure) 3-53
Styles menu item 3-52
  Format menu 3-51
submenus
  Advanced 3-17
  RTL 3-46, 3-47
  SCOPE 3-86
  Select All 3-23
  Technology 3-46, 3-47
  Visual Elite Cross Probing 3-45, 3-48
supported language constructs
  Verilog 8-2
  VHDL 9-2
supported standards
  Verilog 1-25
  VHDL 1-23
Supported Verilog 2001 features (table) 8-4
Symbolic FSM Compiler check box
  Project view 3-98
  symbolic_fsm_compiler command option 4-16
Symbols panel
  Object Query dialog box 3-49
  syn_black_box directive 7-149, 8-41
  examples 7-149
  syn_direct_enable attribute 7-53
  examples
    Verilog 7-53
    VHDL 7-54
  syn_direct_enable attribute (table) 7-53
  syn_edif_bit_format attribute 7-55
  examples
    sdc file 7-55
    Verilog 7-57
    VHDL 7-57
  syn_edif_name_length attribute 7-58
  examples
    Verilog 7-58
    VHDL 7-59
  syn_edif_scalar_format attribute 7-60
  examples 7-60
  syn_encoding attribute 7-62
  examples
    Verilog 7-63
    VHDL 7-64
  FSM encoding style (Verilog) 8-30
  FSM encoding style (VHDL) 9-50
  versus syn_enum_encoding directive 7-154
  syn_enum_encoding (figure) 7-152
  syn_enum_encoding directive 7-151
examples 7-152
  not for FSM encoding 9-50
  versus syn_encoding attribute 7-154
  syn_forward_io_constraints attribute 6-40, 6-42, 7-65
  examples
    sdc file 7-65
    Verilog 7-65
    VHDL 7-66
  syn_hier attribute 7-67
  examples 7-68
  using with fanout guides 7-70
  syn_isclock directive 7-155
  examples 7-155
  syn_keep directive 7-157
  compared with syn_preserve and syn_noprune 7-158
  DSP MAC inference in Altera Stratix B-25
  examples 7-158
  syn_keep directive used to prevent sharing (figure) 7-158
  syn_macro directive 7-161
  examples 7-161
  syn_maxfan attribute 4-25, 7-70, B-17, H-3
  examples
    sdc file 7-70
    Verilog 7-71
    VHDL 7-71
  syn_multstyle attribute 7-72
  DSP MAC inference in Altera Stratix B-25
  examples
    sdc file 7-72
    Verilog 7-72
    VHDL 7-73
  syn_netlist_hierarchy attribute 7-74
  examples 7-74
  syn_noarrayports attribute 7-75
  examples 7-75
  syn_noclockbuf attribute 4-25, 7-76
  examples
    sdc file 7-76
    Verilog 7-76
    VHDL 7-77
  using with fanout guides 7-70
  syn_noprune directive 7-162
  compared with syn_keep and syn_preserve 7-163
syn_noprune directive used to prevent instance optimization (figure) 7-162
examples
- sdc file 7-78
- Verilog 7-78
- VHDL 7-79

syn_pipeline attribute 7-78
examples
- sdc file 7-78
- Verilog 7-78
- VHDL 7-79

syn_preserve directive 7-167
- compared with syn_keep and syn_noprune 7-168
- DSP MAC inference in Altera Stratix B-25
examples 7-168

syn_preserve_sr_priority attribute
examples
- sdc file 7-81
- Verilog 7-81
- VHDL 7-82

syn_preserve_sr_priority attribute
(Actel) 7-81

syn_props attribute (Triscend) 7-83
examples 7-83

syn_radhardlevel attribute (Actel) 7-85
examples 7-86

syn_ramstyle attribute 7-87, 8-36
See also RAM inference
examples 7-89
examples of use
- Altera B-5

syn_ramstyle support (table) 7-87
syn_reference_clock attribute 6-35, 7-91
examples 6-35, 7-91

syn_replicate attribute 7-92
examples
- sdc file 7-92
- Verilog 7-92
- VHDL 7-93

syn_resources attribute 7-95
examples
- sdc file 7-95
- Verilog 7-95
- VHDL 7-96

syn_romstyle attribute
- Altera 7-97
examples
- sdc file 7-97, 7-100
- Verilog 7-97, 7-100

VHDL 7-98, 7-101
examples of use
- Altera B-7
- Xilinx 7-100

syn_sharing directive 7-172
examples 7-172

syn_srlstyle attribute 7-102
examples
- Verilog 7-102
- VHDL 7-103

syn_state_machine directive 7-173
examples 7-173

syn_state_machine directive example
(figure) 7-173

syn_tco directive 7-177
black-box timing constraint 6-36
examples
- Verilog 7-177
- VHDL 7-178

syn_tpd directive 7-180
black-box timing constraint 6-36
examples
- Verilog 7-180
- VHDL 7-181

syn_translate_off directive
examples
- Verilog 7-187
- VHDL 7-188

syn_translate_on directive
examples
- Verilog 7-187
- VHDL 7-188

syn_tristate directive 7-183
examples 7-183

syn_tristatetomux attribute 7-104
examples
- sdc file 7-104
- Verilog 7-104
- VHDL 7-105

syn_tsu directive 7-184
black-box timing constraint 6-36
examples
- Verilog 7-184
- VHDL 7-185

syn_useenables attribute 7-106
examples 7-106
syn_useioff attribute
Altera 7-108
  DSP MAC inference in Altera
    Stratix B-25
examples
  sdc file 7-108, 7-110, 7-112, 7-114
  Verilog 7-108, 7-110, 7-112, 7-114
  VHDL 7-109, 7-111, 7-113, 7-114
Lattice ORCA 7-110
QuickLogic 7-112
Xilinx 7-114
synchronous FSM from concurrent assignment statement (VHDL) 9-55
synchronous READ RAM
  Verilog 8-37
  VHDL 9-81
synchronous sets and resets
  Verilog 8-26
  VHDL 9-46
Synplicity customer support, contacting 1-4
Synplicity Software Authorization form 3-70
SYNPLICITY_LICENSE_FILE environment variable 3-69
Synplify ASIC synthesis tool, overview 1-6
  synplify command-line command 1-13
Synplify Pro synthesis tool, overview 1-5
Synplify synthesis tool, overview 1-5
Synplify Tool GUI (figure) 1-9
Synplify vs. Synplify Pro menu item
  Help menu 3-65
Syntax Check menu item
  Project View popup menu 3-74
  Run menu 3-43
syntax checking 2-9
Syntax Coloring and Font dialog boxes
  (figure) 3-58
Syntax Coloring dialog box 3-58
synthesis
  guidelines, Verilog 8-8
  macro, Verilog 8-45
  running 1-14
Synthesis and Actel-specific attributes and directives (table) A-10
Synthesis and Lattice-specific attributes and directives (table) E-16
Synthesis and Xilinx attributes and directives (table) H-43
synthesis attributes
  individual descriptions 7-18
  overview 7-1
synthesis attributes and directives
  Verilog 8-47
  VHDL 9-85
Synthesis Check menu item
  Project View popup menu 3-75
  Run menu 3-43
synthesis checking 2-9
Synthesis Constraints Optimization Environment (SCOPE) 6-3
synthesis directives
  individual descriptions 7-139
  overview 7-1
Synthesis directives (table) 7-7
synthesis examples, VHDL 9-89
synthesis guidelines
  Verilog 8-7
  VHDL 8-7, 9-33
synthesis log file 1-27, 4-23
synthesis macro, Verilog 8-45
Synthesis menu (Visual Elite) 5-10
Synthesis Script window (Visual Elite) 5-10
Synthesize menu item
  Project View popup menu 3-76
  Run menu 3-43
System Time radio button 3-15

T
Tab keyboard shortcut (Text Editor) 2-8
tabbed panels
  See panels
tabs
  See panels
Tcl (Tool Command Language) 4-2
Tcl commands 3-3, 4-1
  Actel A-8
    add_file 4-3, 4-4, 4-6, 4-9, 6-20
    constraint_file 6-24
    constraint_file 4-10
    define_clock 6-6, 6-25
    define_false_path 6-34
    define_input_delay 6-10, 6-29
    define_multicycle_path 6-32
    define_output_delay 6-10, 6-31
    define_reg_input_delay 6-27
    define_reg_output_delay 6-28
Tcl commands (continued)
   for constraint files 6-2
   for synthesis 4-9
   get_env 4-11
   get_option 4-12
   instprop (QuickLogic) F-10
   portprop (QuickLogic) F-9
   project 4-3, 4-12
       Altera ACEX B-28
       Altera APEX B-16
       Altera APEX II B-16
       Altera Excalibur B-16
       Altera FLEX B-28
       Altera MAX B-34
       Altera Mercury B-16
       Altera Stratix B-21
       Lattice MACH E-14
       Xilinx CoolRunner H-41
       Xilinx CoolRunner-II CPLDs H-41
       Xilinx CPLDs H-41
       Xilinx Spartan H-28
       Xilinx Virtex H-31
       Xilinx XC3000, XC5200 H-38
       Xilinx XC4000 H-35
       Xilinx XC9500 H-41
   project_file 4-14
   set_option 4-3, 4-4, 4-6, 4-15
       Actel A-8
       Altera ACEX B-26
       Altera APEX B-15
       Altera APEX II B-15
       Altera Excalibur B-15
       Altera FLEX B-26
       Altera MAX B-33
       Altera Mercury B-15
       Altera Stratix B-20
       Atmel C-4
       Cypress D-11
       Lattice GAL/isp E-11
       Lattice MACH E-13
       Lattice ORCA E-6
       QuickLogic F-6
       Triscend G-3
       Xilinx CoolRunner H-40
       Xilinx CoolRunner-II CPLDs H-40
       Xilinx CPLDs H-40
       Xilinx Spartan H-27
       Xilinx Virtex H-30
       Xilinx XC3000, XC5200 H-37
       Xilinx XC4000 H-34
       Xilinx XC9500 H-40
       timing constraints 6-24
   Tcl constraint file commands 6-2
   Tcl constraint files 6-20
   Tcl conventions 4-2
   TCL Help menu item 3-66
       Help menu 3-65
   Tcl scripts 4-1, 4-3, 4-7
       creating 4-3
       examples 4-21
       running 3-44, 4-5
   -technology command
       option 4-15, A-8, B-15, B-21, B-27, B-3
       3, C-4,
   Technology Schematic view 2-5
   Technology submenu
       HDL Analyst menu 3-46, 3-47
       Technology view 2-5, 4-22, 5-2
       amount of logic displayed 5-7
       customizing 2-14
       dialog boxes 3-86
       file 1-27
       popup menu 3-85
       Technology Schematic view 2-5
   Technology view (figure) 2-6
   Technology View icon
       Analyst toolbar 3-90
   Technology View popup menu
       Dissolve Selected Instances item 3-85
       SCOPE submenu 3-86
       Edit Attributes item 3-86
       View FSM Info File item 3-86
   Technology View toolbar icon 2-5
   text crossprobing option 3-61
   Text Editor
       features 2-8
       options 3-56
       popup menu 3-84
       removing all bookmarks 3-92
       selecting text columns 2-9
Index

Text Editor (continued)
  starting 2-7
  view 2-7
Text Editor features (table) 2-8
Text Editor view (figure) 2-7
Text File action button
  License Editor dialog box 3-70
Text Label Options dialog box zone
  Schematic Options dialog box 3-59
Through column
  SCOPE spreadsheet 6-13, 6-15
  -through command option 6-32, 6-34, 7-157
  through points, multicycle and false paths 6-15
Tile Horizontally menu item 3-63
  Window menu 3-62
Tile Vertically menu item 3-64
  Window menu 3-62
  -time command option 4-14
  time header and footer variable
    (printing) 3-15
timing analysis of critical paths in HDL
  Analyst 5-18
timing constraints 6-1, 6-2
  Altera 6-39
  black-box 6-36
  define_clock 4-24, 6-6, 6-25
  define_false_path 6-34
  define_input_delay 4-24, 5-22, 6-10, 6-29
  define_multicycle_path 6-32
  define_output_delay 4-24, 5-22, 6-10, 6-31
  define_reg_input_delay 4-24, 5-22, 6-27
  define_reg_output_delay 5-22, 6-28
  forward-annotating 6-39
  Lattice ORCA 6-41
  rise and fall 6-9
  role in optimization 1-17
  SCOPE spreadsheet 6-3
  specifying 6-2
  syn_forward_io_constraints 7-65
  Tcl commands 6-24
  Xilinx 6-42
timing exceptions 5-21
  See multicycle paths, false paths
  defining paths/points 6-15
  specifying 6-12
timing failures 5-21
timing report 6-46
  clock relationships 6-48
detailed clock report 6-49
generating 6-46
  header 6-46
  interface information 6-48
  performance summary 6-47
  Xilinx 6-32
Timing Report header (figure) 6-46
Timing Report panel
  Options for implementation dialog box 3-35, 6-46
Timing Report panel, Options for implementation dialog box (figure) 3-35
Tip of the Day dialog box 3-72
  Next Tip action button 3-72
  Previous Tip action button 3-72
  Show Tips at Startup check box 3-72
  Tip of the Day dialog box (figure) 3-72
  Tip of the Day menu item
  Help menu 3-65, 3-72
To column
  SCOPE spreadsheet 6-12, 6-15
  -to command option 6-32, 6-34
  Toggle Bookmark icon
    Edit toolbar 3-92
  Toggle bookmark menu item
    Edit menu 3-17
  Tool Command Language (Tcl) 4-7
  Toolbar name dialog box zone
    Toolbars dialog box 3-25
toolbars 3-2, 3-88
  Analyst 3-90
  customizing 3-55
  Edit 3-92
  overview 3-88
  Project 3-88
  View 3-91
  Toolbars dialog box 3-24
  Cool Look check box 3-26
  Customize action button 3-25
  Delete action button 3-25
  Large Buttons check box 3-26
  New action button 3-25
  Reset action button 3-25
  Show Tooltips check box 3-26
  Toolbar name zone 3-25
  Toolbars zone 3-25
  Toolbars dialog box (figure) 3-25
  Toolbars dialog box features (table) 3-25

Index-50

Synpify Reference Manual, April 2002
Synplify Reference Manual, April 2002
variables (continued)

file name for header and footer (printing) 3-15
for page header and footer (printing) 3-15
LM_LICENSE_FILE 3-69, 5-9
long date and time for header and footer (printing) 3-15
long date for header and footer (printing) 3-15
page number for header and footer (printing) 3-15
SYNPLICITY_LICENSE_FILE 3-69
time for header and footer (printing) 3-15

VHDL 9-10
VSH_LIB 5-9, 5-10
VSH_VISUAL_COMMAND 5-9, 5-10
vendor I/O instantiation, Verilog 8-41
VENDOR_STRING license value 3-70
Vendor-specific attributes (table) 7-3
vendor-specific Tcl commands 4-20
Vendor-specific Tcl commands (table) 4-20
Verilog
always blocks 8-8
for combinational logic 8-16
assignment operators 8-2
asynchronous sets and resets 8-24
asynchronous state machines 8-32, 8-33
attribute syntax 7-15
attributes and directives 8-47
attributes specified as C-style comments 7-15
black box instantiation 8-40
built-in gate primitives 8-14
combinational logic 8-15, 8-18
comments, syntax 8-10
continuous assignments 8-18, 8-21
creating a hierarchical design 8-43
creating a scalable adder (example) 8-12
creating a scalable module 8-11
creating combinational logic 8-15
creating sequential logic 8-19
directives 8-47
examples
always blocks, event control 8-17
asynchronous sets and resets 8-24
asynchronous state machines 8-33
combinational logic 8-15
continuous assignments for combinational logic 8-18
creating a hierarchical design 8-43
creating a scalable adder 8-12
creating a scalable module 8-11
FSM coding 8-30
FSM default state assignment 8-31
FSM, defining state values 8-32
inferring RAM with special write enables 8-35
level-sensitive latches 8-21
multiplexor 8-17
overriding a parameter value
with # 8-13
with defparam 8-12
ROM inference 8-38
sequential logic 8-19
signal assignments in always blocks 8-21
synchronous READ RAMs 8-37
synchronous sets and resets 8-26
using a scalable module 8-11
using synthesis macro 8-45
flip-flops using always blocks 8-20
FSM 8-29
full_case directive 7-144
gate primitives 8-14
general synthesis guidelines 8-7
hierarchy 8-43
I/O instantiation 8-40
ignored language constructs 8-3
instance name syntax 6-21
instantiating a technology-vendor I/O 8-41
instantiating black boxes 8-40, 8-41
instantiating gate primitives 8-14
language constructs 8-2
language guidelines 8-8
language support 8-1
level-sensitive latches
using always blocks 8-22
using continuous assignments 8-21
limited RAM resources 8-36
module name syntax 6-21
module template 8-9
net name syntax 6-21
netlist file 1-26
object naming syntax 6-21
Verilog (continued)
operators
  arithmetic 8-2
  assignment 8-2
parallel_case directive 7-147
port name syntax 6-21
PREP Benchmarks 8-42
PREP benchmarks 8-42
primitives 8-14
RAM inference 8-34
RAM with special write enables 8-35
recommended reading 1-25
ROM inference 8-38
ROM tables 8-38
scalable modules 8-11
scaling by overriding a parameter value
  with # (example) 8-13
  with defparam (example) 8-12
sequential logic 8-19, 8-20, 8-21, 8-22, 8-24, 8-26
  examples 8-19
sequential logic examples 8-19
sequential shift components H-17
sets and resets 8-24
simple gates 8-14
state machine guidelines 8-29
state machines 8-29
state values (FSM) 8-32
statements, assignment 8-18
structural netlist file 1-26
structural Verilog 8-43
supported language constructs 8-2
supported standards 1-25
synchronous READ RAM 8-37
synchronous sets and resets 8-26
synthesis attributes and directives 8-47
synthesis guidelines 8-7
synthesis macro 8-45
tristate gates 8-14
unsupported language constructs 8-3
using macro libraries (Actel) A-6
using scalable modules 8-11
Verilog 2001 1-25
  generate statement examples 8-6
  signed signals examples 8-6
Verilog 2001 check box
  File Options dialog box 3-78
  Options for implementation dialog box 3-37
Verilog 95 1-25
-verilog command option 4-3, 4-9
Verilog panel
  Options for implementation dialog box 3-36
Verilog panel features (table) 3-36
Verilog Standard dialog box zone
  File Options dialog box 3-78
Vertical Lines check box, Display Settings dialog box 3-28
Vertically tiled windows display (figure) 3-64
VHDL
  accessing packages 9-13
  adding source files to design library 9-11
  always-enabled write enable 9-78
  assignment operators 9-9
  assignments 9-34
  asynchronous FSM created with a process 9-55
  asynchronous sets and resets 9-44
  asynchronous state machines 9-54
  that generate error messages 9-55
attributes 9-85
  syntax 7-16
  attributes package 7-16, 9-85
  bit enumeration data type 9-6
  black boxes 9-86
  black-box timing constraints 9-86
  boolean enumeration data type 9-6
  case statement 9-20
  clock edge representation in wait statements 9-40
  clock edges 9-38
  combinational logic 9-24
  examples 9-89
  comments, syntax 9-34
  compiling design units into libraries 9-11
  component instantiation 9-24
  concurrent signal assignments 9-21
  conditional signal assignments 9-23
  configuration declaration 9-64
  specification 9-60
  constants 9-11
  designating with SNS 9-27
  SNS 9-27
  creating a hierarchical design 9-56
VHDL (continued)
creating a scalable design 9-71
using generate statements 9-75
using generics 9-72
using unconstrained vector ports 9-71
creating an extra initialization state 9-53
creating flip-flops and registers using processes 9-36
data types 9-5, 9-34
for signed and unsigned arithmetic 9-7
integer data type 9-7
signed arithmetic 9-7
unsigned arithmetic 9-8
declaring and assigning objects 9-8
defining an event outside a process 9-39
demand loading 9-32
design libraries 9-11
design unit syntax 6-22
detecting forgotten assignment to the next state 9-54
detecting reset problems 9-53
directives 9-85
event outside a process block 9-39
examples
accessing libraries 9-13
accessing packages 9-13
asynchronous state machines 9-55
case statement 9-21
conditional signal assignments 9-23
configuration declaration 9-65
configuration specification 9-61
constants 9-11
creating a hierarchical design 9-59
creating a resource library 9-14
creating a scalable design using generate statements 9-75
creating a scalable design using generics 9-73
creating a scalable design using unconstrained vector ports 9-71
creating flip-flops and registers using processes 9-37, 9-39
creating level-sensitive latches 9-42
creating unwanted level-sensitive latches 9-43
D flip-flop with asynchronous, active-high reset, set 9-45
D flip-flop with synchronous, active-high reset, set 9-48
default assignment 9-52
detecting forgotten assignment to the next state 9-54
FSM coding style 9-51
generating synchronous read RAMs 9-81
if-then-else statement 9-20
inerring RAMs, nested write enables 9-80
inerring ROMs 9-82
instantiating a technology-vendor I/O 9-88
level-sensitive latches using concurrent signal assignments 9-7
model template 9-34
RAM inference 9-77
RAMs with always-enabled write enables 9-79
RAMs with special write enables 9-78
selected signal assignments 9-22
signals and ports 9-9
simple component instantiation 9-25
simple signal assignments 9-22
SNS constants 9-27
SNS functions 9-28
SNS operators 9-29
SNS user-defined functions 9-30
user-defined enumeration types 9-7
using a scalable architecture 9-73
variables 9-10
flip-flops, example of creating 9-36
generics for scalable designs 9-72
hierarchical designs 9-56
if-then-else statement 9-19
ignored language constructs 9-4
inerring RAM 9-77
instance name syntax 6-22
instantiating a technology-vendor I/O 9-87
instantiating black boxes 9-86, 9-87
instantiating components 9-15, 9-24
integer data type 9-7
language constructs 9-2, 9-5
supported 9-2
Index

VHDL (continued)
  language guidelines 9-33
  language support 9-1
  latch error 9-43
  level-sensitive latches
    concurrent signal assignment 9-40
    process blocks 9-41
  libraries 1-24, 9-11
    setting up 3-41
  library and package rules 9-14
  library packages
    accessing 9-13
    accessing standard package 9-14
    attributes package 9-85
    IEEE supported packages 9-2
    predefined 9-12
  library statement 9-13
  limited RAM resources 9-80
  miscellaneous language constructs 9-3
  model template 9-34
  naming objects 9-8
  nested write enable 9-79
  net name syntax 6-22
  object naming syntax 6-22
  objects 9-8
  operators 9-15
    arithmetic 9-15
    logical 9-16
    miscellaneous 9-17
    relational 9-16
    shift 9-17
  packages 9-11
  port definitions 9-9
  port name syntax 6-22
  ports and signals 9-9
  predefined enumeration types 9-5
  predefined packages 9-12
  PREP benchmarks 9-91
  prepared components (Altera) B-51
  process keyword 9-18
  process template 9-36
    to model combinational logic 9-18
  processes 9-33
    creating flip-flops and register 9-36
  RAM inference 9-77
  RAM with special write enable 9-78
  recommended reading 1-24
  registers 9-36
  resource sharing 9-23

  ROM inference 9-82
  RTL view primitives 9-46
  scalable architecture, using 9-73
  scalable designs 9-71
    generate statement 9-75
    generics 9-72
    unconstrained vector ports 9-71
  Selected Name Support 9-26
  selected signal assignments 9-22
  sensitivity list 9-18
  sequential logic 9-24
    examples 9-90
  sequential shift components H-15
  sequential statements 9-19
  set and reset signals 9-44
    asynchronous 9-44
    synchronous 9-46
  setting enumeration encoding 3-54
  sharing operators in case statements 9-23
  signal assignments 9-9
    conditional 9-23
    selected 9-22
    simple 9-22
  signals and ports 9-9
  simple component instantiation 9-25
  simple signal assignments 9-22
  simulation using enumerated types 9-53
  SNS 9-26
    constants 9-27
    demand loading 9-32
    functions and operators 9-28
    user-defined function support 9-30
  state machines 9-49
    guidelines 9-49
  statements
    case 9-20
    generate 9-75
    if-then-else 9-19
    library 9-13
    use 9-13
    WAIT 9-40
    wait 9-39
  std_logic enumeration data type 9-6
  structural netlist file 1-26
  supported language constructs 9-2
  supported standards 1-23
VHDL (continued)
synchronous FSM from concurrent assignment statement 9-55
synchronous READ RAM 9-81
synchronous sets and resets 9-46
synthesis attributes and directives 9-85
synthesis examples 9-89
synthesis guidelines 9-33
unsupported language constructs 9-4
for configuration declaration 9-69
for configuration specification 9-63
use statement 9-13
user-defined enumeration data types 9-7
using enumerated types for state values 9-52
using macro libraries (Actel) A-6
variables 9-10
wait statement 9-39
wait statement inside a process 9-39
VHDL and Verilog panels, Options for Implementation dialog box (figure) 3-36
VHDL arithmetic operators (table) 9-15
-vhdl command option 4-3, 4-9
VHDL logical operators (table) 9-16
VHDL panel
  Options for implementation dialog box 3-36
VHDL panel features (table) 3-37
VHDL relational operators (table) 9-16
VHDL shift operators (table) 9-17
vhm file 1-26
View FSM Info File menu item
  RTL View popup menu 3-86
  Technology View popup menu 3-86
View Log File menu item
  View menu 3-22
  commands 3-22
dialog boxes 3-24
Full View item 3-24
Goto Net Driver item 3-23
Next Sheet item 3-23
Normal View item 3-24
Pan Center item 3-23
Pan item 3-23
Previous Sheet item 3-23
Properties item 3-24
Push/Pop Hierarchy item 3-23
ReadOnly item 3-24
Select All submenu 3-23
  Instances item 3-23
  Nets item 3-23
  Ports item 3-23
Select Net Driver item 3-23
Select Net Instances item 3-23
Show Line Numbers item 3-24
Status Bar item 3-22
Toolbars item 3-22
Unselect All item 3-23
View Log File item 3-22
View Result File item 3-22
View Sheets item 3-23
Workbook Mode item 2-13, 3-22
Zoom In item 3-23, 3-24
Zoom Lock item 3-24
Zoom Out item 3-23, 3-24
View Menu commands
  all views 3-22
  in RTL/Technology views 3-23
  in SCOPE window 3-24
View menu commands (table) 3-22
View project files in folders check box 2-12
  Project View Options dialog box 3-56
View Result File menu item
  View menu 3-22
View Sheets menu item
  View menu 3-23
View toolbar 3-91
  Next Sheet icon 3-91
  Previous Sheet icon 3-91
  Push/Pop Hierarchy icon 3-91
Zoom In 100% icon 3-91
Zoom Full icon 3-91
Zoom In icon 3-91
Zoom Out icon 3-91
View toolbar (figure) 3-91
View toolbar icons (table) 3-91
viewing critical paths, HDL Analyst 5-18
viewing objects across design levels and views 5-12
viewing other sheets 5-15
views 2-2
  arranging 3-62
  managing 2-13
Project 2-2
  RTL 2-4, 4-22, 5-2
  Technology 2-5, 4-22, 5-2, 5-7
Virtex (Xilinx) H-30
Virtex project Tcl command options  
(table) H-31
Virtex set_option Tcl command options  
(table) H-31
Virtual Clock column  
SCOPE spreadsheet 6-7
-virtual command option 6-26
Visual Elite  
crossprobing 5-8
 crossprobing to trace component cause 5-11
 launching synthesis from 5-10
 Tools menu 5-10
Visual Elite Cross Probing submenu  
HDL Analyst menu 3-45, 3-48
vma file 1-26
VQM filenames in Altera B-9
VSH_LIB environment variable 5-9, 5-10
VSH_VISUAL_COMMAND environment variable 5-9, 5-10

W
 wait statement inside a process (VHDL) 9-39
Window menu 3-62
 Arranged Icons item 3-62
 Cascade item 3-62, 3-63
 Close All item 3-62
 commands 3-62
 New Window item 3-62
 Tile Horizontally item 3-62, 3-63
 Tile Vertically item 3-62, 3-64
 Window menu commands (table) 3-62
windows 2-2
 arranging 3-62
 arranging horizontally 3-63
 arranging vertically 3-64
 cascading 3-63
 cloning 3-62
 copying 3-62
 HDL Analyst 5-2
 Project 2-2
Windows (Microsoft) platform setup 5-9
 wires, preserving with syn_keep  
directive 7-157
Workbook Mode menu item 2-13
 View menu 3-22
Workspace Options menu item 3-80
 Project View popup menu 3-77
Worst case edge-to-edge timing (figure) 6-8
Worst case posedge-to-posedge timing  
(figure) 6-8
Wrap around search check box  
Find dialog box 3-20, 3-21
Write Vendor Constraint File check box 6-39
-write_apr_constraint command option 4-19
-write_verilog command option 4-19
-write_vhdl command option 4-19

X
xc_alias attribute (Xilinx) 7-116
 examples 7-116
xc_clockbuftype attribute (Xilinx) 7-117
 examples 7-117
xc_fast attribute (Xilinx) 4-25, 7-119
 examples 7-119
xc_fast_auto attribute (Xilinx) 7-121
 examples 7-121
xc_global_buffers attribute (Xilinx) 7-123
 examples 7-123
xc_isgsr directive (Xilinx) 7-189
 examples 7-189
xc_loc attribute (Xilinx) 4-25, 7-124
 examples 7-124
xc_map attribute (Xilinx) 7-125
 examples 7-125
xc_ncf_auto_relax attribute  
(Xilinx) 6-43, 7-126
 examples 7-126
xc_nodelay attribute (Xilinx) 7-128
 examples 7-128
xc_padtype attribute (Xilinx) 7-129
 examples 7-129
xc_props attribute (Xilinx) 7-131
 examples 7-131
xc_pulldown attribute (Xilinx) 7-132
 examples 7-132
xc_pullup attribute (Xilinx) 7-132
 examples 7-132
xc_rloc attribute (Xilinx) 7-134
 examples
 sdc file 7-134
 Verilog 7-134
 VHDL 7-135
xc_slow attribute (Xilinx) 7-136
 examples 7-136
xc_uset attribute (Xilinx) 7-137
examples 7-137
XC3000 and XC5200 project Tcl command options (table) H-39
XC3000 and XC5200 set_option Tcl command options (table) H-38
XC3000, XC5200 (Xilinx) H-37
XC4000 (Xilinx) H-34
XC4000 project Tcl command options (table) H-36
XC4000 set_option Tcl command options (table) H-35
XC9500 (Xilinx) H-40
Xilinx H-1
attributes H-5, H-43
black boxes H-7
BUFG H-3
bus port names H-26
clock enable net assignment 7-53
constraints H-5
CoolRunner
device mapping options H-40
project Tcl command H-41
set_option Tcl command H-40
CoolRunner-II CPLDs
device mapping options H-40
project Tcl command H-41
set_option Tcl command H-40
CPLDs H-40
device mapping options H-40
project Tcl command H-41
set_option Tcl command H-40
Design Manager H-32
directives H-43
dynamic srl H-15
EDIF port names H-25
fanouts H-2
forward annotation 6-42, H-8
global set/reset H-4
I/O insertion
automatic H-22
manual H-23
inferring registers H-14
interfacing with place and route H-32
level-sensitive latches H-15
macros H-7, H-18
MAP guidelines H-33
mixed-level schematics H-5
packing IOBs using
syn_useioff 7-114, F-3, H-7
pad types 7-129
PAR guidelines H-33
pin locations for automatically inserted I/Os H-22
port locations 7-124
RAM inference H-10
relaxing timing constraints 6-43
resource usage report H-32
SelectRAM H-11
Spartan
device mapping options H-27
project Tcl command H-28
set_option Tcl command H-27
startup block H-4
support for D flip-flops H-2
supported families H-2
syn_maxfan attribute H-3
syn_romstyle attribute 7-100
syn_useioff attribute 7-114
timing constraints 6-42
Virtex
device mapping options H-30
project Tcl command H-31
set_option Tcl command H-30
xc_alias attribute 7-116
xc_clockbuftype attribute 7-117
xc_fast attribute 7-119
xc_fast_auto attribute 7-121
xc_globalBuffers attribute 7-123
xc_isg directive 7-189
xc_loc attribute 7-124
xc_map attribute 7-125
xc_ncf_auto_relax attribute 6-43, 7-126
xc_nodelay attribute 7-128
xc_padtype attribute 7-129
xc_props attribute 7-131
xc_pulldown attribute 7-132
xc_pullup attribute 7-132
xc_rloc attribute 7-134
xc_slow attribute 7-136
xc_uset attribute 7-137
XC3000, XC5200
device mapping options H-37
project Tcl command H-38
set_option Tcl command H-37
XC4000
   device mapping options H-34
   project Tcl command H-35
   set_option Tcl command H-34
XC9500
   device mapping options H-40
   project Tcl command H-41
   set_option Tcl command H-40
Xilinx CPLDs project Tcl command options
   (table) H-42
Xilinx CPLDs set_option Tcl command options (table) H-41
Xilinx menu item
   Options menu 3-54
Xilinx startup block assignment, by family
   (table) H-6
   -xilinx_m1 command option 4-19

Z
   Zoom 100% icon
     View toolbar 3-91
   Zoom Full icon
     View toolbar 3-91
   Zoom In icon
     View toolbar 3-91
   Zoom In menu item
     View menu 3-23, 3-24
   Zoom Lock menu item
     View menu 3-24
   Zoom Out icon
     View toolbar 3-91
   Zoom Out menu item
     View menu 3-23, 3-24
   zooming 3-91