Synopsys FPGA Synthesis
User Guide

March 2013

https://solvnet.synopsys.com

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2013 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only.

Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AEON, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, CoMET, CODE V, Design Compiler, DesignWare, EMBED-IT!, Formality, Galaxy Custom Designer, Global Synthesis, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, LightTools, MAST, METeor, ModelTools, NanoSim, NOVeA, OpenVera, ORA, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Sonic Focus, STAR Memory System, Syndicated, Synplicity, the Synplicity logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, ARC, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIMplus, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCSi, VHDL Compiler, VMC, and Worksheet Buffer are trademarks of Synopsys, Inc.

Copyright © 2013 Synopsys, Inc.

Service Marks (sm)

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

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A
March 2013

Contents

Chapter 1: Introduction

Synopsys FPGA and Prototyping Products 24
FPGA Implementation Tools 24
Synopsys FPGA Tool Features 26

Scope of the Document 29
The Document Set 29
Audience 29

Getting Started 30
Starting the Software 30
Getting Help 30
Requesting Technical Support 31

User Interface Overview 32

Chapter 2: FPGA Synthesis Design Flows

Logic Synthesis Design Flow 36

Synplify Premier Synthesis Design Flows 39
Logic Synthesis with Enhanced Optimization 40
Design Plan-Based Logic Synthesis 42
Altera Graph-Based Physical Synthesis 46
Altera Graph-Based Physical Synthesis with Design Planner 49
Design Plan-based Physical Synthesis 51

Altera Physical Synthesis 55
Guidelines for Physical Synthesis in Altera Designs 56
Set up the Altera Physical Synthesis Project 56
Run Logic Synthesis for the Altera Physical Synthesis Flow 60
Validate Logic Synthesis Results for Altera Physical Synthesis 61
Run Altera Physical Synthesis 61
Analyze Results of Altera Physical Synthesis 61

Copyright © 2013 Synopsys, Inc.

Xilinx Physical Plus 64
Physical Plus Design Flows 65
Setting Up I/O and Clock Component Constraints 67
Creating Region Constraints 73
Running Physical Plus Starting from RTL 74
Running Physical Plus Starting from EDIF 78
Input/Output Files and Directory Structure 81
Physical Plus Dependencies 83

Hierarchical Project Management Flows 86
Block-First Development Flow for Hierarchical Projects 88
Top-First Development Flow for Hierarchical Projects 89
Bottom-Up Synthesis for Hierarchical Projects 91
Top-Down Synthesis for Hierarchical Projects 93
Mixed Block Synthesis for Hierarchical Projects 95

Prototyping Flow 97

Chapter 3: Preparing the Input

Setting Up HDL Source Files 100
Creating HDL Source Files 100
Using the Compiler Directives Editor 102
Using the Context Help Editor 108
Checking HDL Source Files 110
Editing HDL Source Files with the Built-in Text Editor 111
Setting Editing Window Preferences 114
Using an External Text Editor 116

Using Mixed Language Source Files 117

Working with Constraint Files 122
When to Use Constraint Files over Source Code 122
Using a Text Editor for Constraint Files (Legacy) 122
Tcl Syntax Guidelines for Constraint Files 123
Checking Constraint Files 125
Generating Constraint Files for Forward Annotation 125

Using Input from Related Tools 127

Chapter 4: Specifying Constraints

Using the SCOPE Editor 130
Creating Constraints in the SCOPE Editor 130

Copyright © 2013 Synopsys, Inc.

Specifying SCOPE Constraints 135
Entering and Editing SCOPE Constraints 135
Setting Clock and Path Constraints 137
Defining Input and Output Constraints 139
Specifying Standard I/O Pad Types 140
Using the TCL View of SCOPE GUI 141
Guidelines for Entering and Editing Constraints 143

Specifying Timing Exceptions 146
Defining From/To/Through Points for Timing Exceptions 146
Defining Multicycle Paths 150
Defining False Paths 151

Finding Objects with Tcl find and expand 152
Specifying Search Patterns for Tcl find 152
Refining Tcl Find Results with -filter 154
Using the Tcl Find Command to Define Collections 155
Using the Tcl expand Command to Define Collections 157
Checking Tcl find and expand Results 158
Using Tcl find and expand in Batch Mode 159

Combining Tcl find with Other Operations 160

Using Collections 161
Creating and Using SCOPE Collections 162
Creating Collections using Tcl Commands 164
Viewing and Manipulating Collections with Tcl Commands 167

Converting SDC to FDC 171

Using the SCOPE Editor (Legacy) 173
Entering and Editing SCOPE Constraints (Legacy) 175
Specifying SCOPE Timing Constraints (Legacy) 176
Defining Input and Output Constraints (Legacy) 187
Defining False Paths (Legacy) 188

Translating XDC Constraints to FDC 190

Translating Altera QSF Constraints 193

Specifying Xilinx Constraints (Legacy) 195
Setting Clock Priority in Xilinx Designs (Legacy) 196

Converting and Using Xilinx UCF Constraints 200
Converting UCF Constraints Without Creating a Project 200
Using Xilinx UCF Constraints in a Logic Synthesis Design 201
Using Xilinx UCF Constraints in a Physical Synthesis Design 203
Support for UCF Conversion 206

Copyright © 2013 Synopsys, Inc.

Chapter 5: Setting up a Logic Synthesis Project

Setting Up Project Files 212
Creating a Project File 212
Opening an Existing Project File 215
Making Changes to a Project 216
Setting Project View Display Preferences 217
Updating Verilog Include Paths in Older Project Files 219

Managing Project File Hierarchy 220
Creating Custom Folders 220
Manipulating Custom Project Folders 223
Manipulating Custom Files 224

Setting Up Implementations 226

Setting Logic Synthesis Implementation Options 228
Setting Device Options 228
Setting Optimization Options 231
Specifying Global Frequency and Constraint Files 234
Specifying Result Options 237
Specifying Timing Report Output 238
Setting Verilog and VHDL Options 239

Specifying Attributes and Directives 245
Specifying Attributes and Directives in VHDL 246
Specifying Attributes and Directives in Verilog 247
Specifying Directives in a CDC File 248
Specifying Attributes Using the SCOPE Editor 250
Specifying Attributes in the Constraints File 253

Working with Hierarchical Projects 254
Creating Hierarchical Subprojects by Exporting Blocks 254
Creating Hierarchical Subprojects by Exporting Instances 257
Creating Nested Subprojects 263
Generating Dependent File Lists 269
Working with Multiple Implementations in Hierarchical Projects 270
Working with Multiple Instances and Parameterized Modules 271
Allocating Resources for Subprojects 275
Setting Initial Timing Budgets for Subprojects 277
Configuring Synthesis Runs for Hierarchical Projects 278
Analyzing Synthesis Results for Hierarchical Projects 282

Searching Files 284
Identifying the Files to Search 285
Filtering the Files to Search 285

Copyright © 2013 Synopsys, Inc.

Initiating the Search 286
Search Results 286

Archiving Files and Projects 287
Archive a Project 287
Un-Archive a Project 291
Copy a Project 294

Chapter 6: Setting up a Physical Synthesis Project

Setting up for Physical Synthesis 300

Setting Options for Physical Synthesis 302
Setting Synplify Premier Prototyping Tools Optimizations 302
Creating a Place and Route Implementation 303
Specifying Altera Place-and-Route Options 308
Specifying Xilinx Place-and-Route Options in a Tcl File 311
Specifying Xilinx Place-and-Route Options in an opt File 314
Specifying Xilinx Global Placement Options 318

Setting Constraints for Physical Synthesis 320
Using Design Planner Floorplan Constraints 320
Translating Pin Location Files 321
Setting Physical Synthesis Constraints for Altera 322

Forward-Annotating Physical Synthesis Constraints 323
Forward Annotating Altera Constraints 323
Limitations Using the Physical Analyst and Technology View 325

Backannotating Physical Synthesis Constraints 326
Backannotating Place-and-Route Data 326
Generating a Xilinx Coreloc Placement File 327

Chapter 7: Inferring High-Level Objects

Defining Black Boxes for Synthesis 330
Instantiating Black Boxes and I/Os in Verilog 330
Instantiating Black Boxes and I/Os in VHDL 332
Adding Black Box Timing Constraints 334
Adding Other Black Box Attributes 338

Defining State Machines for Synthesis 340
Defining State Machines in Verilog 340
Defining State Machines in VHDL 341
Specifying FSMs with Attributes and Directives 343

Copyright © 2013 Synopsys, Inc.

Implementing High-Reliability Designs 345
Implementing Distributed TMR 346
Implementing Duplication with Comparison (DWC) 350
Using Redundancy for I/O Connectors 353
Using TMR or ECC for RAMs 356
Specifying Safe FSMs 359
Error Monitoring for High Reliability Features 361

Inferring RAMs 369
Inference Versus Instantiation 369
Basic Guidelines for Coding RAMs 370
Specifying RAM Implementation Styles 375
Implementing Altera RAMs Automatically 376
Implementing Xilinx RAMs Automatically 379
Implementing Altera Stratix Multi-Port RAMs 382
Inferring Altera Stratix III LUTRAMs 383
Inferring Xilinx Block RAMs Using Registered Addresses 385
Inferring Xilinx Block RAMs Using Registered Output 387
Mapping Xilinx ROM to Block RAM 392

Initializing RAMs 394
Initializing RAMs in Verilog 394
Initializing RAMs in VHDL 395
Initializing Xilinx RAM 398

Inferring Shift Registers 405

Working with LPMs 411
Instantiating Altera LPMs as Black Boxes 412
Instantiating Altera LPMs Using VHDL Prepared Components 416
Instantiating Altera LPMs Using a Verilog Library 418

Chapter 8: Specifying Design-Level Optimizations

Tips for Optimization 422
General Optimization Tips 422
Optimizing for Area 423
Optimizing for Timing 424

Pipelining 426
Prerequisites for Pipelining 426
Pipelining the Design 427

Retiming 430
Controlling Retiming 430
Retiming Example 432

Retiming Report 433
How Retiming Works 434
How Retiming Works With Synplify Premier Regions 437

Preserving Objects from Optimization 438
Using syn_keep for Preservation or Replication 439
Controlling Hierarchy Flattening 442
Preserving Hierarchy 442

Optimizing Fanout 444
Setting Fanout Limits 444
Controlling Buffering and Replication 446

Sharing Resources 448

Inserting I/Os 452

Optimizing State Machines 454
Deciding when to Optimize State Machines 454
Running the FSM Compiler 455
Running the FSM Explorer 459

Inserting Probes 462
Specifying Probes in the Source Code 462
Adding Probe Attributes Interactively 463

Chapter 9: Clock Conversion

Working with Gated Clocks 466
Obstacles to Conversion 468
Prerequisites for Gated Clock Conversion 468
Defining Clocks Properly 471
Synthesizing a Gated-Clock Design 475
Accessing the Clock Conversion Report 476
Analyzing the Clock Conversion Report 477
Interpreting Gated Clock Error Messages 479
Disabling Individual Gated Clock Conversions 482
Instantiated Cells 483
Integrated Clock Gating Cells 484
Using Gated Clocks for Black Boxes 489
OR Gates Driving Latches 490
MUX or XOR Logic in Clock Structure 491
Obstructions to Optimization 492
Unsupported Constructs 493
Other Potential Workarounds to Solve Clock-Conversion Issues 494
Restrictions on Using Gated Clocks 494

Copyright © 2013 Synopsys, Inc.

Optimizing Generated Clocks 499
Enabling Generated-Clock Optimization 499
Conditions for Generated-Clock Optimization 499
Generated-Clock Optimization Examples 500

Chapter 10: Synthesizing and Analyzing the Results

Synthesizing Your Design 506
Running Logic Synthesis 506
Running Physical Synthesis 506
Using Up-to-date Checking for Job Management 509

Checking Log Results 514
Viewing the Log File 514
Analyzing Results Using the Log File Reports 517
Using the Watch Window 518

Handling Messages 521
Checking Results in the Message Viewer 521
Filtering Messages in the Message Viewer 523
Filtering Messages from the Command Line 525
Automating Message Filtering with a Tcl Script 526
Log File Message Controls 528
Handling Warnings 530

Continue on Error 531
Compilation Process 531
Enabling the Continue-on-Error Feature 532
Error Reporting 533
Continue-on-Error for Compile Points 535

Validating Results for Physical Synthesis 536

Congestion Analysis After Logic Synthesis 538
Using the Congestion Analysis Report 540
Displaying Congestion in the Physical Analyst 542

Chapter 11: Analyzing with HDL Analyst and FSM Viewer

Working in the Schematic Views 546
Differentiating Between the Views 547
Opening the Views 547
Viewing Object Properties 549
Selecting Objects in the RTL/Technology Views 553
Working with Multisheet Schematics 555
Moving Between Views in a Schematic Window 556

Copyright © 2013 Synopsys, Inc.

Setting Schematic View Preferences 557
Managing Windows 559

Exploring Design Hierarchy 560
Traversing Design Hierarchy with the Hierarchy Browser 560
Exploring Object Hierarchy by Pushing/Popping 561
Exploring Object Hierarchy of Transparent Instances 567

Finding Objects 568
Browsing to Find Objects in HDL Analyst Views 568
Using Find for Hierarchical and Restricted Searches 570
Using Wildcards with the Find Command 573
Combining Find with Filtering to Refine Searches 578
Using Find to Search the Output Netlist 578

Crossprobing 581
Crossprobing within an RTL/Technology View 581
Crossprobing from the RTL/Technology View 582
Crossprobing from the Text Editor Window 584
Crossprobing from the Tcl Script Window 587
Crossprobing from the FSM Viewer 587

Analyzing With the HDL Analyst Tool 589
Viewing Design Hierarchy and Context 590
Filtering Schematics 593
Expanding Pin and Net Logic 595
Expanding and Viewing Connections 599
Flattening Schematic Hierarchy 600
Minimizing Memory Usage While Analyzing Designs 605

Using the FSM Viewer 606

Chapter 12: Floorplanning with Design Planner

Using Design Planner 612
Starting Design Planner 612
Copying Objects in the Design Planner Tool 614
Controlling Pin Display in the Design Plan Editor 615
Creating and Using a Design Plan File for Physical Synthesis 618

Assigning Pins and Clocks 619
Assigning Pins Interactively 619
Assigning Clock Pins 622
Modifying Pin Assignments 623
Using Temporary Pin Assignments 624
Viewing Assigned Pins in Different Views 626
Viewing Pin Assignment Information 627

Copyright © 2013 Synopsys, Inc.

Working with Regions 629
Creating Regions 629
Using Region Tunneling 631
Moving and Sizing Regions 633
Viewing Intellectual Property (IP) Core Areas 634
Assigning Logic to Top-level Chip Regions 634
Assigning Logic to Regions 638
Replicating Logic Manually 638
Checking Utilization 639

Working with Altera Regions 641
Creating Design Planner Regions for Altera Designs 642
Assigning Logic to Altera Design Planner Regions 643

Working with Xilinx SSI Devices 645
Assigning Logic to SLR Regions 645
SLR Dependencies 648

Working with Xilinx Regions 650
Creating Regions for Xilinx Designs 651

Using Process-Level Hierarchy 652

Bit Slicing 652
Using Bit Slicing 652
Bit Slice Examples 657

Chapter 13: Analyzing Designs in Physical Analyst

Analyzing Physical Synthesis Results 662
Analyzing Physical Synthesis Results Using Various Tools 662
Running Multiple Implementations 663
Checking Altera Pre-Placement Physical Synthesis Results 663

Using Physical Analyst 665
Opening the Physical Analyst Interface 665
Zooming in the Physical Analyst 667
Moving Between Views in the Physical Analyst 668
Using the Physical Analyst Context Window 669

Displaying and Selecting Objects 671
Setting Visibility for Physical Analyst Objects 671
Displaying Instances and Sites in Physical Analyst 672
Displaying Nets in Physical Analyst 676
Selecting Objects in Physical Analyst 678

Copyright © 2013 Synopsys, Inc.

Querying Physical Analyst Objects 681
Viewing Properties in Physical Analyst 681
Using Tool Tips to View Properties in Physical Analyst 684

Finding Objects 686
Using Find to Locate Physical Analyst Objects) 686
Finding Physical Analyst Objects by Their Locations 690
Using Markers to Find Physical Analyst Objects 691
Identifying Encrypted IP Objects in Physical Analyst 693

Crossprobing in Physical Analyst 695
Crossprobing from the Physical Analyst View 695
Crossprobing from a Text File to Physical Analyst 698
Crossprobing from the RTL View to Physical Analyst 699
Crossprobing from the Technology View to Physical Analyst 701

Analyzing Netlists in Physical Analyst 703
Filtering the Physical Analyst View 703
Expanding Pin and Net Logic in Physical Analyst 704
Expanding and Viewing Connections in Physical Analyst 709

Using Implementation Maps in Physical Analyst 710
Implementation Maps Controls and Settings 712
Using the Routing Congestion Map 713
Using the Block Component Utilization Map 715
Using the Block Input Utilization Map 716
Using the Slack Distribution Map 717
Crossprobing from Implementation Maps 719

Chapter 14: Analyzing Timing

Analyzing Timing in Schematic Views 722
Viewing Timing Information 722
Annotating Timing Information in the Schematic Views 723
Analyzing Clock Trees in the RTL View 725
Viewing Critical Paths 725
Handling Negative Slack 728

Generating Custom Timing Reports with STA 729

Using Analysis Design Constraints 732
Scenarios for Using Analysis Design Constraints 733
Creating an ADC File 734
Using Object Names Correctly in the adc File 738

Copyright © 2013 Synopsys, Inc.

Using Auto Constraints 739
Results of Auto Constraints 741

Using the Timing Report View 744

Analyzing Timing with Physical Analyst 752
Viewing Critical Paths in Physical Analyst 752
Tracing Critical Paths Forward in Physical Analyst 755
Tracing Critical Paths Backward in Physical Analyst 757

Chapter 15: Optimizing for Specific Targets

Optimizing Altera Designs 760
Working with Altera PLLs 760
Instantiating Special Buffers as Black Boxes in Altera Designs 761
Specifying Altera I/O Locations 763
Packing I/O Cell Registers in Altera Designs 763
Specifying HardCopy and Stratix Companion Parts 765
Specifying Core Voltage in Stratix III Designs 766
Using LPMs in Simulation Flows 767
Improving Altera Physical Synthesis Performance 769
Working with Quartus II 769

Optimizing Lattice Designs 772
Instantiating Lattice Macros 772
Using Lattice GSR Resources 774
Inferring Carry Chains in Lattice XPLD Devices 775
Inferring Lattice PIC Latches 775
Controlling I/O Insertion in Lattice Designs 783
Forward-Annotating Lattice Constraints 784

Optimizing Lattice iCE65/iCE40 Designs 786
Using Sequential Logic 786
Using Combinational Logic 787
Handling Tristates 787
Handling I/Os and Buffers 788

Optimizing Microsemi Designs 789
Using Predefined Microsemi Black Boxes 789
Using Smartgen Macros 790
Working with Radhard Designs 790
Specifying syn_radhardlevel in the Source Code 791

Optimizing Xilinx Designs 794
Designing for Xilinx Architectures 794
Specifying Xilinx Macros 795

Copyright © 2013 Synopsys, Inc.

Specifying Global Sets/Resets and Startup Blocks 797
Inferring Wide Adders 798
Instantiating CoreGen Cores 801
Instantiating Virtex PCI Cores 802
Packing Registers for Xilinx I/Os 804
Specifying Xilinx Register INIT Values 807
Inserting Xilinx I/Os and Specifying Pin Locations 809
Working with Xilinx Buffers 815
Working with Xilinx Regional Clock Buffers 816
Specifying RLOCs 817
Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute 819
Using Clock Buffers in Virtex Designs 820
Working with Clock Skews in Xilinx Virtex-5 Physical Designs 822
Instantiating Special I/O Standard Buffers for Virtex 823
Reoptimizing with EDIF Files 824
Improving Xilinx Physical Synthesis Performance 825
Running Post-Synthesis Simulation 826
Working with Xilinx Place-and-Route Software 827

Chapter 16: Working with Synthesis Output

Passing Information to the P&R Tools 830
Specifying Pin Locations 830
Specifying Locations for Microsemi Bus Ports 831
Specifying Macro and Register Placement 831
Passing Technology Properties 832
Specifying Padtype and Port Information 832

Generating Vendor-Specific Output 834
Targeting Output to Your Vendor 834
Customizing Netlist Formats 835

Invoking Third-Party Vendor Tools 836
Configuring Tool Tags 836
Invoking a Third-Party Tool 837

Chapter 17: Running Post-Synthesis Operations

Running P&R Automatically after Synthesis 842
Integrating Synthesis and Place-and-Route in One Run 842
Running the Integrated Synthesis and Xilinx ISE Flow 843
Releasing the Synthesis License During Place and Route 845

Copyright © 2013 Synopsys, Inc.

Running Altera Quartus II Incrementally 846
Quartus II Incremental Compilation Flow with Fast Fit 846
Quartus II Incremental Compilation Flow 848

Running Xilinx Vivado Place-and-Route 852
Vivado Place-and-Route Design Flow 853
Setting Vivado Environment Variables 855
Running the Vivado Flow 856
Customizing Vivado Place and Route Options 860

Running Vivado Incrementally 864

Running Xilinx ISE Incrementally 869
Xilinx SmartGuide Flow 870
Xilinx Partition Flow 873
Xilinx Partition Flow for Versions Before ISE 12.1 878

Working with the Identify Tools 880
Launching from the Synplify Pro or Synplify Premier Tool 880
Launching from the Synplify Tool 882
Handling Problems with Launching Identify 884
Using the Identify Tool 885
Using Compile Points with the Identify Tool 887

Netlist Editing 889
RTL-Level Flow 889
Diagnosing Problems 891
Netlist Editing Commands 891

VIF Formal Verification Flows 893
Overview of the VIF Flow 893
Generating VIF Files 894
Using VIF with Cadence Conformal 897
Handling Equivalency Check Failures 898

Simulating with the VCS Tool 899

Using the VCD/Identify-Analyst Integration Tool 904
VCD-Analyst Integration 904
Identify-Analyst Integration 921
VCS Test Bench Extraction 927

Chapter 18: Working with IP Input

Generating IP with SYNCore 934
Specifying FIFOs with SYNCore 934
Specifying RAMs with SYNCore 939

Specifying Byte-Enable RAMs with SYNCore 946
Specifying ROMs with SYNCore 952
Specifying Adder/Subtractors with SYNCore 957
Specifying Counters with SYNCore 964

The Synopsys FPGA IP Encryption Flow 970
Overview of the Synopsys FPGA IP Flow 970
Encryption and Decryption 971

Working with Encrypted IP 975
Encrypting Your IP 975
Encrypting IP with the encryptP1735.pl Script 976
Encrypting IP with the encryptIP Script 978
Specifying the Script Output Method 979
Preparing the IP Package 981

Using DesignWare IP 988
Using DesignWare Building Blocks 988
DW_Foundation_Arith Package 990

Working with Synenc-encrypted IP 992

Using Hyper Source 993
Using Hyper Source for Prototyping 993
Using Hyper Source for IP Designs 993
Threading Signals Through the Design Hierarchy of an IP 994

Working with Altera IP 998
Using Altera LPMs or Megafunctions in Synthesis 998
Implementing Megafunctions with Clearbox Models 1002
Implementing Megafunctions with Grey Box Models 1012
Including Altera MegaCore IP Using an IP Package 1019
Including Altera Processor Cores Generated in SOPC Builder 1023

Working with SOPC Builder Components 1029
Setting up the SOPC Builder Synthesis Project 1030
Defining SOPC Components as Black Boxes and White Boxes 1031

Importing Projects from Quartus 1033
Importing Quartus Projects 1033
Importing Quartus Designs with Megacore IPs 1038
Importing Quartus Designs with Megafunctions/LPMs 1039
Troubleshooting Imported Quartus Designs 1040

Working with Lattice IP 1042

Importing Vivado IP 1043

Copyright © 2013 Synopsys, Inc.

Working with Xilinx IP Cores 1046
Xilinx Cores 1046
Secure and Non-secure Cores 1047
Including Xilinx Cores for Logic and Physical Synthesis 1049

Converting Xilinx Projects with ise2syn 1051
Converting Designs with the ise2syn Utility 1051
The ise2syn Conversion Process 1056
Specifying EDK Cores as White Boxes 1059

Chapter 19: Fast Synthesis

About Fast Synthesis 1062

Using Fast Synthesis 1063

Fast Synthesis and Other Synthesis Options 1066

Chapter 20: Activity Analysis for Power Optimization

Activity Analysis Flow 1068
Assign Activity Analysis Constraints 1068
Generate the Report (SAIF File) 1069

About the SAIF File 1071

Specifying Activity Analysis Constraints 1073
syn_state1_prob 1074
syn_trans_prob 1075
State and Transition Probability Relationship 1075
Clock and Reset 1075

Specifying SAIF Parameters 1077

Technology View (project_saif.srm) 1078

Activity Analysis for Power Improvement Flow 1079
Power Improvement Flow 1079
Xilinx Power Analysis 1080
XPower Report 1081
Xilinx Place and Route using SAIF 1084
Xilinx Place and Route from Synplify Premier 1085

Chapter 21: Working with Compile Points

Compile Point Basics 1088
Advantages of Compile Point Design 1088
Automatic and Manual Compile Points 1090

Copyright © 2013 Synopsys, Inc.

Nested Compile Points 1091
Compile Point Types 1092

Compile Point Synthesis Basics 1098
Compile Point Constraint Files 1098
Interface Logic Models 1100
Interface Timing for Compile Points 1101
Compile Point Synthesis 1104
Incremental Compile Point Synthesis 1107
Forward-annotation of Compile Point Timing Constraints 1108

Synthesizing Compile Points 1109
The Automatic Compile Point Flow 1109
The Manual Compile Point Flow 1113
Creating a Top-Level Constraints File for Compile Points 1116
Defining Manual Compile Points 1117
Setting Constraints at the Compile Point Level 1119
Analyzing Compile Point Results 1121
Using Automatic and Manual Compile Points Together 1123

Using Compile Points with Other Features 1124
Combining Compile Points with Fast Synthesis 1124
Combining Compile Points with Multiprocessing 1125

Resynthesizing Incrementally 1126
Resynthesizing Compile Points Incrementally 1126
Synthesizing Incrementally with Other Tools 1129

Chapter 22: Optimizing Processes for Productivity

Using Batch Mode 1132
Running Batch Mode on a Project File 1132
Running Batch Mode with a Tcl Script 1133
License Queuing 1134

Working with Tcl Scripts and Commands 1139
Using Tcl Commands and Scripts 1139
Generating a Job Script 1140
Setting Number of Parallel Jobs 1140
Creating a Tcl Synthesis Script 1141
Using Tcl Variables to Try Different Clock Frequencies 1143
Using Tcl Variables to Try Several Target Technologies 1144
Running Bottom-up Synthesis with a Script 1145

Automating Flows with synhooks.tcl 1146

Using Revision Control Tools 1149

Copyright © 2013 Synopsys, Inc.

Chapter 23: Using Multiprocessing

Multiprocessing With Compile Points 1156
Setting Maximum Parallel Jobs 1156
License Utilization 1157

Chapter 24: Verifying Results with Formality

Overview of the Formality Application 1160

The Design Flow 1162
Verifying the Design with Formality Equivalence Checking 1162
Generating VIF Files 1165
The svf File 1168

Finite State Machine Design Example 1169
Verifying a Finite State Machine Design Example 1169
RTL Code Example for a Finite State Machine 1170
FSM Tcl File Example 1173
FSM svf File Example 1173

Tips and Guidelines 1174
Guidelines for Successful Verification 1174
Writing RTL for Verification used in ASIC Prototyping 1175
Limitations to Verification with Formality 1178
Verification Mode Dependencies 1179

Copyright © 2013 Synopsys, Inc.

CHAPTER 1

Introduction

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

- Synopsys FPGA and Prototyping Products, on page 24
- Scope of the Document, on page 29
- Getting Started, on page 30
- User Interface Overview, on page 32

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

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA and Prototyping Products

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

FPGA Implementation Tools

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

Synplify Pro Product

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

Synplify Premier Product

The Synplify Premier solution is a superset of the Synplify Pro product functionality and is the ultimate FPGA implementation and debug environment. It provides a comprehensive suite of tools and technologies for advanced FPGA designers, as well as ASIC prototypers targeting single FPGA-based prototypes. The Synplify Premier software is a technology independent solution that addresses the most challenging aspects of FPGA design including timing closure, logic verification, IP usage, ASIC compatibility, DSP implementation, debug, and tight integration with FPGA vendor back-end tools.

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

Features exclusively supported in the Synplify Premier tool are the following:

- Fast and Enhanced Synthesis Modes
- Physical Synthesis
- Design Planning (Optional)
- DesignWare Support
- Integrated RTL Debug (Identify Tool Set)
- Power Switching Activity (SAIF Generation)

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Tool Features

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

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Performance				
Behavior Extracting Synthesis Technology® (BEST™)	x	x	x	x
Vendor-Generated Core/IP Support (certain technologies)		x	x	x
FSM Compiler	x	x	x	x
FSM Explorer		x	x	x
Gated Clock Conversion		x	x	x
Register Pipelining		x	x	x
Register Retiming		x	x	x
Code Analysis				
SCOPE® Spreadsheet	x	x	x	x
HDL Analyst®	Option	x	x	x
Timing Analyzer – Point-to-point		x	x	x
FSM Viewer		x	x	x
Crossprobing		x	x	x
Probe Point Creation		x	x	x
Physical Design				
Design Plan File				x
Logic Assignment to Regions				x
Area Estimation and Region Capacity				x
Pin Assignment				x

Copyright © 2013 Synopsys, Inc.

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Physical Synthesis Optimizations				x
Graph-based Physical Synthesis			x	x
Physical Analyst			x	x
Prototyping			x	x
Synopsys DesignWare Foundation Library			x	x
Runtime Advantages				
Enhanced Optimization			x	x
Fast Synthesis			x	x
Team Design				
Mixed Language Design		x	x	x
Compile Points		x	x	x
True Batch Mode (Floating licenses only)		x	x	x
GUI Batch Mode (Floating licenses)	x	x	x	x
Batch Mode Post-synthesis P&R Run	-	x	x	x
Back-annotation of P&R Data	-	-	-	x
Formal Verification Flow		x	x (Physical synthesis disabled)	x (Physical synthesis disabled)
Identify Integration	Limited	x	x	x
Back-annotation of P&R Data				x
Design Environment				
Technical Resource Center	x	x	x	x
Text Editor View	x	x	x	x

Copyright © 2013 Synopsys, Inc.

	Synplify	Synplify Pro	Synplify Premier	Synplify Premier DP
Watch Window		x	x	x
Message Window		x	x	x
Tcl Window		x	x	x
Multiple Implementations		x	x	x
Vendor Technology/Family Support	x	x	Limited	Limited

Scope of the Document

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

The Document Set

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

- The user guide only explains the options needed to do the typical tasks described in the manual. It does not describe every available command and option. For complete descriptions of all the command options and syntax, refer to the User Interface Overview chapter in the *Synopsys FPGA Synthesis Reference Manual*.
- The user guide contains task-based information. For a breakdown of how information is organized, see *Getting Help*, on page 30.

Audience

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

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

Copyright © 2013 Synopsys, Inc.

Getting Started

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

- Starting the Software, on page 30
- Getting Help, on page 30
- Requesting Technical Support, on page 31

Starting the Software

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

     ```
     synplify
     synplify_pro
     synplify_premier
     synplify_premier_dp
     ```

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

Getting Help

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

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

Requesting Technical Support

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

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

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

Copyright © 2013 Synopsys, Inc.

User Interface Overview

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

Synplify Pro and Synplify Premier Standard Interface

Synplify Interface

The following figure shows you the Synplify interface.

Copyright © 2013 Synopsys, Inc.

CHAPTER 2

FPGA Synthesis Design Flows

This describes the following tool flows:

- Logic Synthesis Design Flow, on page 36
- Synplify Premier Synthesis Design Flows, on page 39
- Altera Physical Synthesis, on page 55
- Xilinx Physical Plus, on page 64
- Hierarchical Project Management Flows, on page 86
- Prototyping Flow, on page 97

Copyright © 2013 Synopsys, Inc.

Logic Synthesis Design Flow

The Synopsys FPGA tools synthesize logic by first compiling the RTL source, and then doing logical mapping and optimizations. After logic synthesis, you get a vendor-specific netlist and constraint file that you use as inputs to the place-and-route (P&R) tool.

The following figure shows the phases and the tools used for logic synthesis and some of the major inputs and outputs. You can use the Synplify, Synplify Pro, or Synplify Premier synthesis software for this flow. The interactive timing analysis, physical analysis, and backannotation steps that are shown in gray are optional. Although the flow shows the vendor constraint files as direct inputs to the P&R tool, you should add these files to the synthesis project for timing black boxes.

Logic Synthesis Procedure

For a design flow with step-by-step instructions based on specific design data, download the tutorial from the website. The following steps summarize the process, which is also illustrated in the figure that follows.

1. Create a project.
2. Add the source files to the project.
3. Set attributes and constraints for the design.
4. Set options for the implementation in the Implementation Options dialog box.
5. If you are running Synplify Premier in logic synthesis mode, other modes can be set. See *Fast Synthesis*, on page 1061 and *Logic Synthesis with Enhanced Optimization*, on page 40.
6. Click Run to run logic synthesis.
7. Analyze the results, using the log file, the HDL Analyst schematic views, the Message window and the Watch Window.

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

Copyright © 2013 Synopsys, Inc.

The following figure lists the main steps in the flow:

SYNPLIFY & SYNPLIFY PRO

SYNPLIFY PREMIER

Synplify Premier Synthesis Design Flows

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

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

Logic Synthesis Flows	
Logic Synthesis Design Flow, on page 36	Same as Synplify Pro logic synthesis
Logic Synthesis with Fast Synthesis	Synplify Premier logic synthesis with fast synthesis runtimes. For details about this flow, see *Fast Synthesis*, on page 1061.
Logic Synthesis with Enhanced Optimization, on page 40	Synplify Premier logic synthesis includes additional optimizations during logic synthesis and provides an output netlist with better QoR than when running basic logic synthesis. Enhanced Optimization is turned on by default.
Design Plan-Based Logic Synthesis, on page 42	Synplify Premier logic synthesis with placement constraints from a floorplan file (needs Design Planner option)
Physical Synthesis Flows	

Copyright © 2013 Synopsys, Inc.

Logic Synthesis Flows	
Altera Graph-Based Physical Synthesis, on page 46	*Altera only* Automated physical synthesis. This includes the enhanced logic synthesis optimizations.
Altera Graph-Based Physical Synthesis with Design Planner, on page 49	*Altera only* Automated physical synthesis that uses floorplan file constraints (needs Design Planner option)
Xilinx Physical Plus, on page 64	*Xilinx only* Modular flow that allows you to separate the logical and physical stages of the design process, using the Process view GUI to guide you through the steps that can or must be executed based on updates to the input files.

Logic Synthesis with Enhanced Optimization

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

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

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

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

Running Logic Synthesis with Enhanced Optimization

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

1. Create a Synplify Premier project.
2. Add the source files to the project.
3. Set attributes and constraints for the design.
4. Set options for the implementation in the Implementation Options dialog box.

Copyright © 2013 Synopsys, Inc.

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

   - Make sure that Fast Synthesis is disabled, either in the Project view or on the Options tab of the Implementation Options dialog box. The two options are contradictory and if both options are enabled, Fast Synthesis takes priority.
   - Enable Enhanced Optimization in the Device tab of the Implementation Options dialog box. When you enable the Physical Synthesis or Physical Plus option, the software automatically uses enhanced optimizations as part of that flow, so you do not have to specifically enable it.
6. Click Run to run logic synthesis.
7. Analyze the results, using the log file, the HDL Analyst schematic views, the Message window and the Watch Window.

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

Design Plan-Based Logic Synthesis

This flow lets you use a floorplan to guide logic synthesis; you do not have to run physical synthesis. To do this, you require the Synplify Premier software with the Design Planner option (see Chapter 12, *Floorplanning with Design Planner* for information about using this tool).

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

The Design Plan region constraints are used in the following flows:

- For Altera designs, to run with logic synthesis or the graph-based physical synthesis flow. See *Design Plan-Based Logic Synthesis*, on page 42 and *Altera Graph-Based Physical Synthesis with Design Planner*, on page 49 for details.
- For Xilinx designs, to guide placement in the Physical Plus flow. For more information, see *Xilinx Physical Plus*, on page 64.

Copyright © 2013 Synopsys, Inc.

Running Logic Synthesis with a Design Plan

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

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

1. Set up the project and compile the design in logic synthesis mode.
   - Set up the design as described in *Set up the Altera Physical Synthesis Project*, on page 56. Set up a P&R implementation.
   - Disable the Physical Synthesis or Physical Plus option to run the tool in logic synthesis mode.
   - Compile the design.

2. Analyze timing results.
   - Analyze timing.
   - Determine which components you want to assign to regions.
3. Launch the Design Planner tool ( ) and do the following:
   - Create regions for the critical paths and interactively assign the critical paths to regions of the chip. See *Working with Regions*, on page 629 and *Working with Altera Regions*, on page 641 for details.
   - Obtain a size estimation for each RTL block in the design. See *Checking Utilization*, on page 639 for details.
   - For multiple clocks, assign critical logic associated with each clock domain (that does not meet design requirements) to a unique region to avoid resource contention.
   - If you have any black boxes in your design, assign them to a region. Designate this region as an IP block, so that the Synplify Premier software can instantiate the black box in the `vqm` file. However, you must provide the content for the black box so that the place-and-route tool can run successfully.

     For details about using Design Planner, see *Floorplanning with Design Planner*, on page 611. Consult the following for more information on how to complete the Design Plan file (`sfp`): *Creating and Using a Design Plan File for Physical Synthesis*, on page 618, *Working with Regions*, on page 629, and *Assigning Pins and Clocks*, on page 619.
   - Save the design plan file (`sfp`) and add it to your project.
4. Run logic synthesis.
   - Make sure the Physical Synthesis or Physical Plus switch is disabled, but that the project includes the physical constraints file (`sfp`).
   - Set up the project to automatically run place-and-route after synthesis completes. Alternatively, you can run the P&R tool in standalone mode.

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

Copyright © 2013 Synopsys, Inc.

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

5. Analyze the timing in the Synplify Premier tool, using the log file and analysis tools. See *Checking Log Results*, on page 514, *Analyzing Timing in Schematic Views*, on page 722, and *Generating Custom Timing Reports with STA*, on page 729 for details.

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

Altera Graph-Based Physical Synthesis

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

Physical synthesis does not require a design plan or place-and-route implementation. If you want to use a design plan file with this flow, see *Altera Graph-Based Physical Synthesis with Design Planner*, on page 49. For graph-based physical synthesis, the tool automatically performs placement with backannotation during the physical synthesis run. It absorbs the core files into the Synplify Premier database for timing, placement and optimizations.

Design Phases in Graph-based Physical Synthesis

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

Logic Synthesis Validation

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

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

For specific explanations of the steps shown here, see *Altera Physical Synthesis*, on page 55.

Physical Synthesis

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

Copyright © 2013 Synopsys, Inc.

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

For specific explanations of the steps shown here, see *Altera Physical Synthesis*, on page 55.

Altera Graph-Based Physical Synthesis with Design Planner

Like the graph-based flow (*Altera Graph-Based Physical Synthesis*, on page 46) this is a push-button, fully automated flow that produces a synthesized design with detailed placement, but it uses a design plan file to specify physical constraints for guiding global placement. Use this flow to improve performance.

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

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

Copyright © 2013 Synopsys, Inc.

SYNPLIFY PREMIER
RTL
Compile RTL
SFP
Run Design Planner
Vendor netlist
SDC
Synthesize Logic
Vendor constraints
Vendor constraints
Run Initial Placement
Run Physical Synthesis
Synthesized netlist
Synthesis constraints
Route
Post P&R Back-annotation
Interactive Timing Analysis
TCL
TAH
Physical Analysis
SRM
LEF
DEF

Design Plan-based Physical Synthesis

This is an interactive flow that where you can specify physical constraints before running physical synthesis. It requires the Design Planner option, a tool that improves performance through physical constraints. This flow supports certain Altera technologies. See *Running Physical Synthesis with a Design Plan*, on page 52 for a procedure using this flow. For information about using Design Planner, see Chapter 12, *Floorplanning with Design Planner*.

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

Copyright © 2013 Synopsys, Inc.

Running Physical Synthesis with a Design Plan

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

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

1. Run logic synthesis.
   - Set up the project for your target technology. See *Set up the Altera Physical Synthesis Project*, on page 56 for details.
   - Synthesize the design in logic synthesis mode, using timing constraints and no physical constraints.

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

2. Analyze timing results. See *Validating Results for Physical Synthesis*, on page 536 for details.

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

3. Determine the critical paths from the P&R results; these are the candidates for logic assignments to regions.

4. Bring up the Design Planner ( ) and do the following:
   - Create regions for the critical paths and interactively assign the critical paths to regions of the chip. See *Working with Regions*, on page 629 and *Working with Altera Regions*, on page 641 for details.
   - Obtain a size estimation for each RTL block in the design. See *Checking Utilization*, on page 639 for details.
   - For multiple clocks, assign critical logic associated with each clock domain (that does not meet design requirements) to a unique region to avoid resource contention.
   - If you have any black boxes in your design, assign them to a region. Designate this region as an IP block, so that the Synplify Premier software can instantiate the black box in the `vqm` file. However, you must provide the content for the black box so that the place-and-route tool can run successfully.

   For details about using Design Planner, see *Floorplanning with Design Planner*, on page 611.

   You can also open Physical Analyst to view the design and check critical path placement. Consult the following sections for more information on how to complete the Design Plan file (`sfp`): *Creating and Using a Design Plan File for Physical Synthesis*, on page 618, *Working with Regions*, on page 629, and *Assigning Pins and Clocks*, on page 619.

5. Save the design plan file (`sfp`) and add it to your project.

6. Run physical synthesis. Use the same project file that you created in step 1 above. This time enable the Physical Synthesis switch and include the physical constraints file (`sfp`). This phase also includes running the netlist through place-and-route after synthesis completes.

Copyright © 2013 Synopsys, Inc.

7. Analyze the timing in the Synplify Premier tool. Use the log file and graphical analysis tools. See *Analyzing Physical Synthesis Results*, on page 662 for details.

   If the target is met, you can continue to the next design phase. If not, you should re-evaluate timing and placement. You might find there is a new critical path or the one that is already assigned to a region that needs tweaking. See *Improving Altera Physical Synthesis Performance*, on page 769 for more suggestions.

Altera Physical Synthesis

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

	For details, see...
Basic logic synthesis	Logic Synthesis Design Flow, on page 36
Logic synthesis with enhanced optimization	Logic Synthesis with Enhanced Optimization, on page 40
Logic synthesis with fast synthesis	Fast Synthesis, on page 1061
Logic synthesis with design plan	Design Plan-Based Logic Synthesis, on page 42
Physical synthesis with design plan	Design Plan-based Physical Synthesis, on page 51
Graph-based synthesis with a design plan	Altera Graph-Based Physical Synthesis with Design Planner, on page 49

Altera Graph-based Physical Synthesis

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

- Guidelines for Physical Synthesis in Altera Designs, on page 56
- Set up the Altera Physical Synthesis Project, on page 56
- Run Logic Synthesis for the Altera Physical Synthesis Flow, on page 60
- Validate Logic Synthesis Results for Altera Physical Synthesis, on page 61
- Run Altera Physical Synthesis, on page 61
- Analyze Results of Altera Physical Synthesis, on page 61

Copyright © 2013 Synopsys, Inc.

Guidelines for Physical Synthesis in Altera Designs

Follow these guidelines for physical synthesis:

- Include the entire design – black boxes cannot be present. However, Altera LPMs (Library of Parameterized Modules) or Megafunctions are supported. See *Using Altera LPMs or Megafunctions in Synthesis*, on page 998 if more information is needed.
- Use the appropriate methodology defined for Altera IPs or Nios II cores in the design. For more information, see the following:
  - Including Altera MegaCore IP Using an IP Package, on page 1019
  - Implementing Megafunctions with Grey Box Models, on page 1012
  - Including Altera Processor Cores Generated in SOPC Builder, on page 1023.
- Assign realistic, accurate timing constraints. Do not over-constrain the tool. (See *Improving Altera Physical Synthesis Performance*, on page 769 for tips.)
- Use the top-down design methodology. (A bottom-up flow is not supported.)
- Do not use compile points with graph-based physical synthesis.
- Install the recommended version of the Altera Quartus II place-and-route tool.

Set up the Altera Physical Synthesis Project

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

1. Make sure you follow these requirements for this flow:
   - Make sure the design is properly constrained. (See *Improving Altera Physical Synthesis Performance*, on page 769 for tips.)
   - Make sure to specify I/O pin location constraints for all pins in the design for physical synthesis.
   - Make sure to include any I/O constraints from the Quartus settings file (`qsf`) as necessary. You can translate the I/O constraints and I/O

standards to `sdc` format with a utility. See *Translating Altera QSF Constraints*, on page 193.

- If you have the Design Planner option, you can use a design plan file (`sfp`) for physical synthesis.
- If you are using one of the graph-based physical synthesis flows, make sure you select a target technology that is supported. See *Altera Graph-based Physical Synthesis*, on page 55.
- Keep the guidelines described in *Guidelines for Physical Synthesis in Altera Designs*, on page 56 in mind.

2. Create the project. If you are using a design plan file, make sure to add it to the project.

   See *Setting Up HDL Source Files*, on page 100 and *Setting Up Project Files*, on page 212 for details.

3. Set constraints.
   - Compile the design. Open the SCOPE interface and set constraints. Timing constraints specify performance goals and describe the design environment. See *Using the SCOPE Editor*, on page 130 and *Specifying SCOPE Constraints*, on page 135 for details.
   - Add physical constraints.
   - Translate constraints from the Quartus settings file (QSF) and combine them with the timing constraints into a single `sdc` constraint file. See *Translating Altera QSF Constraints*, on page 193 for details.
   - Check your constraints with Run->Constraint Check.
   - Save the constraints file and save the project file.

4. Specify the implementation options for synthesis.
   - Click the Implementation Options button.

Copyright © 2013 Synopsys, Inc.

– Set implementation options on the various tabs of the dialog box as shown in the following table. For details about setting implementation options, refer to *Setting Logic Synthesis Implementation Options*, on page 228.

Device	Set technology and device mapping options. Make sure the Disable I/O Insertion option is disabled, because Synplify Premier physical synthesis requires this setting. See *Setting Device Options*, on page 228.
Options	Set optimization switches for synthesis. See *Setting Optimization Options*, on page 231.
Constraints	Set an overall target frequency for the design. Select the constraint files you want to use. See *Specifying Global Frequency and Constraint Files*, on page 234.
Implementation Results	Specify the output results directory and output file options. See *Specifying Result Options*, on page 237 for details.

Timing Report	Specify the number of critical paths and start/end points to display in the timing report. See *Specifying Timing Report Output*, on page 238.
Verilog/VHDL	Specify the HDL options. See *Setting Verilog and VHDL Options*, on page 239.
Netlist Restructure	Specify options for any necessary netlist optimizations, and the netlist restructure file (`nrf`) for which bit slicing might have been performed. See *Setting Synplify Premier Prototyping Tools Optimizations*, on page 302 for descriptions.

– Click OK to apply the implementation options. Save the project file.

5. Create a place-and-route implementation to automatically run the Altera Quartus II place and route tool from the synthesis UI after synthesis.
   – Make sure you have the correct version of the P&R tool and that you have set the environment variables for the tool.
   – Create a P&R implementation. See *Creating a Place and Route Implementation*, on page 303 for details.

Copyright © 2013 Synopsys, Inc.

- Specify the Place & Route Job Name. Make sure the Run Place and Route following synthesis switch is enabled and click OK.
- For Altera devices, you can alternatively choose to Use placement constraints from physical synthesis, otherwise Quartus determines how it handles physical constraints.
- Specify the place-and-route options file. The tool automatically uses default options located in *installDirectory*\lib\altera\altera_par.tcl. For more information, see *Specifying Altera Place-and-Route Options*, on page 308. Select other options for backannotation and for forward-annotation of constraints and click OK.
- Go to Implementation Options->Place and Route and enable the place-and-route implementation that you want to use for your project.

- Save the project file.

Run Logic Synthesis for the Altera Physical Synthesis Flow

If this is the first time you are running synthesis on the design, you must run logic synthesis mode. This means that you run synthesis with the Physical Synthesis switch disabled. You can choose to run logic synthesis in the

Synplify Premier tool with the Enhanced Optimization mode or the standard logic synthesis mode for certain Altera devices. When the Enhanced Optimization mode is:

- Enabled (box is checked in the Project view or on the Implementation Options - Device tab) — Additional optimizations are used to achieve QoR results that exceed the standard logic synthesis QoR. This is the default.
- Disabled (box is unchecked in the Project view or on the Implementation Options - Device tab) — The standard logic synthesis QoR can be achieved.

This initial synthesis run is to determine if there are any problems that need to be addressed before going on to the physical synthesis stage. See the first step of *Running Physical Synthesis*, on page 506 for details of running logic synthesis as part of a physical synthesis flow.

Validate Logic Synthesis Results for Altera Physical Synthesis

After doing an initial run of logic synthesis, check the results and fix any errors you find. See *Validating Results for Physical Synthesis*, on page 536 for details.

Run Altera Physical Synthesis

Once you have validated the logic synthesis run and set up the physical constraints, you can run physical synthesis. Make sure to enable the Physical Synthesis switch and to enable the place-and-route implementation before clicking Run. If you are using a design plan file, you must also enable this file. For a detailed procedure, see *Running Physical Synthesis*, on page 506.

Analyze Results of Altera Physical Synthesis

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

Copyright © 2013 Synopsys, Inc.

Log file that includes the default timing report (`srr` or `htm`)	See *Checking Log Results*, on page 514.
HDL Analyst	Consists of schematic views that help you analyze the design. See Chapter 4, *Specifying Constraints*.
RTL View ( )	Select HDL Analyst->RTL->Hierarchical View or Flattened View to display the compiled view of the design. See Chapter 11, *Analyzing with HDL Analyst and FSM Viewer*.
Technology View ( )	Select HDL Analyst ->Technology->Hierarchical View, or ->Flattened View to display the mapped view of the design. See Chapter 11, *Analyzing with HDL Analyst and FSM Viewer*.
Physical Analyst ( )	The Physical Analyst provides a visual display of the device, and design placement of instances and nets. Select HDL Analyst->Physical Analyst. See Chapter 13, *Analyzing Designs in Physical Analyst*.
Timing Analyst ( )	The stand-alone timing analyzer produces timing reports (`ta`) for specific reporting requirements. See *Generating Custom Timing Reports with STA*, on page 729.

Use these guidelines to analyze the results:

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

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

Copyright © 2013 Synopsys, Inc.

Xilinx Physical Plus

Physical Plus replaces the monolithic Physical Synthesis design flow with a modular flow. Physical Plus:

- Supports Xilinx Virtex families (Virtex-5 and later devices).
- Lets you to import UCF or NCF/XDC constraints previously applied during Xilinx place and route (PAR).
- Separates the logical and physical stages of the design process.
- Is set up easily after you have run Synplify Premier logic synthesis with Enhanced Optimization and Xilinx PAR.
- Lets you create region constraints in the Design Planner.
- Gives you the flexibility to choose the part of the design process for which you can implement netlist or timing changes. To help you do this, use the Process View in the implementation results panel of the Project view. For more information about the Process View, see the *Process View*, on page 77.
- Can help alleviate congestion for unroutable nets in a design.
- Lets you to use the Implementation Maps in the Physical Analyst tool to diagnose congestion and utilization issues in your design. For more information, see *Using Implementation Maps in Physical Analyst*, on page 710.

Topics include:

- Physical Plus Design Flows
- Setting Up I/O and Clock Component Constraints
- Creating Region Constraints
- Running Physical Plus Starting from RTL
- Running Physical Plus Starting from EDIF
- Input/Output Files and Directory Structure
- Physical Plus Dependencies

Copyright © 2013 Synopsys, Inc.

Physical Plus Design Flows

You can run Physical Plus starting from RTL or EDIF. The following figures show these flows.

Physical Plus Starting from RTL

Copyright © 2013 Synopsys, Inc.

Physical Plus Starting from EDIF

Setting Up I/O and Clock Component Constraints

Physical Plus requires that all I/O and clock generating components be locked to a placement location. Clock generating components can include:

- DCM/PLL/MMCM
- BUFG
- BUFGCTRL
- ISERDES/OSERDES
- IODELAY
- IDELAY/ODELAY
- GT
- BUFR
- BUFH
- TEMAC

If you start Physical Plus from RTL, you must provide these pre-placed components in the SDC/FDC file. When starting Physical Plus from EDIF, you must provide these pre-placed components in the PDC file.

To set I/O and clock generating components, you can:

- Manually enter these placement constraints using the syn_loc attribute. For a description of this attribute, see *syn_loc*, on page 1388.
- Use the Import Xilinx Constraints utility. See *Using Import Xilinx Constraints*, on page 67.
- Use an automatic placement option (Virtex-7 devices only). For more information, see

Using Import Xilinx Constraints

The Import Xilinx Constraints utility helps guide you to generate the prerequisite placement constraints for Physical Plus. This utility uses the

- Xilinx Design Check Point (DCP) database file generated after Vivado place and route has been run. You can use either the post-place or post-route DCP file.

Copyright © 2013 Synopsys, Inc.

- Xilinx NDC file after you have run synthesis and ISE place and route. The NCD file contains all placement, timing, and potential routing information. You can use either the post-map or final NCD file. Use the post-map NCD file if your design has routing failures.

Routed designs generate both the post-place and post-route result files.

To import Xilinx constraints, see:

- Import Xilinx Constraints (Vivado), on page 68
- Import Xilinx Constraints (ISE), on page 70

Import Xilinx Constraints (Vivado)

You should use Import Xilinx Constraints for Virtex-7 designs.

To do this, use the Import->Import Xilinx Constraints option from the Project view. Select the Vivado tab on the Import Xilinx Constraints dialog box:

- Add all the input EDIF files to be used in Physical Plus.
- Select the Vivado post-place or post-route design checkpoint (DCP) file from the place-and-route directory.

  The DCP file contains the placement and routing information from Vivado place and route.
- Physical Plus requires that I/O and clock generating components be locked to a placement location. The Import I/O and Clock Component Placement option is enabled by default. You can either:
  - Enable Use for Physical Plus to select the PDC file for the imported placement constraints.
  - Enable Use for Logical Synthesis to select the logic synthesis FDC file for the imported XDC constraints, if you want to use the I/O and clock component placement at the RTL level. You might first need to convert the SDC file to an FDC file.
- You can also chose to import RAM and DSP placements. If so, enable the Include RAM & DSP Placement option.
- Specify the location of the logic synthesis constraints (FDC) file or physical design constraints (PDC) file to which the translated constraints are written in the Project view.

Copyright © 2013 Synopsys, Inc.

- Click on the Import button. A log file opens. Verify that import Xilinx constraints ran successfully.

At this time, the output files *designName*_xdc_placement.PDC and *designName*_xdc_timing.PDC files are generated and added to your project. It is highly recommended that you inspect the constraints in the PDC files and review all warnings in the log file.

Copyright © 2013 Synopsys, Inc.

Import Xilinx Constraints (ISE)

Use Import Xilinx Constraints for Virtex-5 and Virtex-6 designs.

To do this, use the Import->Import Xilinx Constraints option from the Project view. Select the ISE tab on the Import Xilinx Constraints dialog box:

- Add all the input EDIF files to be used in Physical Plus.
- Optionally add any user-generated UCF constraint files to the project (not synplicity.ucf). Physical Plus requires accurate constraints for good results. All constraints should be provided during synthesis.
- Specify the location of the physical design constraints (PDC) file to which the translated UCF constraints are written.
- Physical Plus requires I/O and clock component placement. If this information is not provided in the SDC or UCF file, add the NCD file from the Xilinx PAR job that was run after logical synthesis. The I/O and component placement are extracted from this NCD file.
- Enable Use for Physical Plus to select the PDC file for the imported NCD constraints.

  Note: Enable Use for Logical Synthesis to select the logical synthesis SDC file for the imported NCD constraints, if you want to use the I/O and clock component placement at the RTL level.
- The Import I/O and Clock Component Placement option is enabled by default.
- You can also chose to import RAM and DSP placements. If so, enable the Include RAM & DSP Placement option.
- Click on the Import button. A log file opens. Verify that import Xilinx constraints ran successfully.

At this time, the output files *designName*_ncd.PDC and *designName*_ucf.PDC files are generated and added to your project. It is highly recommended that you inspect the constraints in the PDC files and review all warnings in the log file.

In some cases, a synplicity_unapplied.ucf file is created for UCF constraints that cannot be imported. Manually import any constraints that are critical for timing and placement to the PDC file for the project.

Automatic Placement Option

Xilinx Virtex-7 devices

You can automatically generate the prerequisite I/O and clock component placement. To conveniently do this:

- From the Constraints tab on the Implementation Options dialog box, select the Generate IO & Clock Generating Component Placement for Physical Plus option.

  The Tcl equivalent is

  ```
  set_option -run_xilinx_io_clock_place 1 | 0
  ```


Copyright © 2013 Synopsys, Inc.

- The run_vivado_phys.tcl script runs in the background and randomly selects the I/O and clock generating component placements for the design to use as input for Physical Plus.

Creating Region Constraints

You can interactively create regions and assign logic to them for Xilinx devices in the Design Planner. For information about using the Design Planner, see Chapter 12, *Floorplanning with Design Planner*.

To manually create regions, you can use the syn_assign_to_region attribute. For details, see *syn_assign_to_region*, on page 1230.

These region constraints are forward-annotated to Physical Plus from either the *design*_synplify.fdc or user-specified PDC file. The following examples show how region constraints are formatted:

FDC (Block Region Assignment)

```
define_attribute {i:rgn1.flop} {syn_assign_to_region}
{SLICE_X54Y468:SLICE_X295Y558 RAMB36_X1Y94:RAMB36_X4Y110
RAMB18_X1Y188:RAMB18_X1Y188:RAMB18_X4Y221
DSP48_X2Y188:DSP48_X4Y223}
```

Physical Plus honors these constraints and generates components optimized in the output netlist for place and route.

XDC (Block Region Assignment)

```
create_pblock rgn1
resize_pblock [get_pblocks rgn1]
   -add {SLICE_X54Y468:SLICE_X295Y558}
   -add {RAMB36_X1Y94:RAMB36_X4Y110}
   -add {RAMB18_X1Y188:RAMB18_X4Y221}
   -add {DSP48_X2Y188:DSP48_X4Y223}
add_cells_to_pblock [get_pblocks rgn1] [get_cells rgn1/flop]
   -clear_locs
```


Copyright © 2013 Synopsys, Inc.

Running Physical Plus Starting from RTL

Perform the following tasks:

1. Set up the input files for logic synthesis. Define the HDL, top-level constraints, any IP, and if required physical region constraints.

2. Run logic synthesis with Enhanced Optimization on, and place and route (PAR) the design.

   Run Synplify Premier logic synthesis, but make sure that the Physical Synthesis and Physical Plus options are disabled and Enhanced Optimization is enabled. Place and route the design.

   The Xilinx Vivado place-and-route software must run through the post-placement phase and generate the DCP file, but does not need to complete routing to use Physical Plus. Likewise, Xilinx ISE place-and-route software must run through the map phase without errors to generate the NCD file, but does not need to complete routing to use Physical Plus. Also, note that the Physical Plus process is disabled and therefore greyed out in the Process view.

   An important prerequisite for starting Physical Plus from RTL is that all I/O and clock generating components are pre-placed in the constraint file. This prerequisite placement information is included in the *design*_synplify.sdc files after logic synthesis completes. Physical Plus uses the EDIF and *design*_synplify.sdc file as inputs.

3. Import Xilinx constraints to generate the constraint files needed for Physical Plus.

   To do this, see *Setting Up I/O and Clock Component Constraints*, on page 67.

   When you run:

   – Vivado, the output files *designName*_xdc_placement.PDC and *designName*_xdc_timing.PDC files are generated and added to your project.

   – ISE, the output files *designName*_ncd.PDC and *designName*_ucf.PDC files are generated and added to your project.

4. From the Implementation Options Constraints tab, select the constraint files to be applied for the specified implementation. Make sure that any PDC constraint files are enabled for the implementation.

The Use Logical Constraints for Physical Plus option automatically forward-annotates constraints provided for logical synthesis to Physical Plus. The file that gets forward-annotated is named *design*_synplify.sdc. This option is enabled by default.

Copyright © 2013 Synopsys, Inc.

5. Run Physical Plus. Enable the Physical Plus option using one of the following methods:
   - From the Project view or the Options tab of the Implementation Options dialog box
   - From the Process view, right-click on Physical Plus and select Enable Physical Plus

6. Optionally create a PAR implementation for Physical Plus. To do this:
   - Select the Physical Plus implementation. Then, right-click and select Add Placement Place & Route.
   - Select the Run Place & Route following synthesis option, to run PAR automatically after synthesis.
   - Select the Generate Congestion Analysis + Run Timing & Placement Backannotation option, to backannotate timing and placement information and automatically generate a congestion report. Click OK.

Notice that the Place & Route Source is Physical Plus.

Copyright © 2013 Synopsys, Inc.

Running Physical Plus Starting from EDIF

Perform the following tasks:

1. Run logic synthesis with Enhanced Optimization on, and place and route (PAR) the design. This generates the EDIF that is the input for Physical Plus.

2. Import Xilinx constraints to generate the constraint files needed for Physical Plus. Another important prerequisite for starting Physical Plus from RTL is that all I/O and clock generating components are pre-placed in the constraint file.

   To do this, see *Setting Up I/O and Clock Component Constraints*, on page 67.

   When you run:

   – Vivado, the output files *designName*_xdc_placement.PDC and *designName*_xdc_timing.PDC files are generated and added to your project.

   – ISE, the output files *designName*_ncd.PDC and *designName*_ucf.PDC files are generated and added to your project.

3. From the Implementation Options Constraints tab, select the constraint files to be applied for the specified implementation. Make sure that any PDC constraint files are enabled for the implementation.

4. Run Physical Plus. Enable the Physical Plus option using one of the following methods:
   - From the Project view or the Options tab of the Implementation Options dialog box
   - From the Process view, right-click on Physical Plus and select Enable Physical Plus

Copyright © 2013 Synopsys, Inc.

5. Optionally create a PAR implementation for Physical Plus. To do this:
   - Select the Physical Plus implementation. Then, right-click and select Add Placement Place & Route.
   - Select the Run Place & Route following synthesis option, to run PAR automatically after synthesis.
   - Select the Generate Congestion Analysis + Run Timing & Placement Backannotation option, to backannotate timing and placement information and automatically generate a congestion report. Click OK.

     Notice that the Place & Route Source is Physical Plus.

Input/Output Files and Directory Structure

Note the following input/output file requirements and directory structures generated:

- For designs using ISE:
  - The NCD file should not be added to the Project file. Use the Import->Import Xilinx Constraints option to extract the I/O and clock component placement needed for Physical Plus.
  - UCF constraints can be added to the project using the Import->Import Xilinx Constraints option. UCF constraints added to the project will be passed to Xilinx PAR, but are not applied during synthesis.

Copyright © 2013 Synopsys, Inc.

- When you use ISE, the Import Xilinx Constraints command outputs the _ncd.pdc and _ucf.pdc files. This process translates placement and timing constraints from the Xilinx ISE NCD UCF files to PDC files. Constraints in the PDC file format are applied during physical synthesis.
  - It is highly recommended that you verify the constraints in the _ncd.pdc and ucf.pdc files. Edit them as needed.
  - The PDC file uses the same format as the SDC file; the extension was renamed to denote that this file be used in Physical Plus. You can also add the input SDC file to Physical Plus by specifying:

    ```
    add_file -placement_constraint fileName.sdc
    ```

    Alternatively from the GUI, you can right-click on an input SDC file and select Project Options from the Project view. On the File Options dialog box, select File Type as Physical Constraints.

- Physical Plus directory structures include the following:
  - Output files are stored in folders for the step that generated them. Logic synthesis result files are written to the implementation folder. When Physical Plus is enabled, a physical_plus subdirectory is automatically created in the implementation directory and all output files are written there. For example, output EDIF files will not be overwritten by subsequent steps and can keep their original names.

- Log files (for the *design*.srr and *design*.html) are generated in the implementation directory. If you run both logic synthesis and Physical Plus, two reports are appended in the SRR file with the Physical Plus log file written in the implementation directory generated by logic synthesis. However, log files generated from the Process steps are written to the synlog subdirectory in the implementation directory. These files can be distinguished by the step name being appended to the result files (for example, _fpga_mapper.srr, _place_premap.srr, or _place_job.srr).
- The top-level EDIF file contains a description string for how Synplify Premier was synthesized. Look for Synplify Premier Enhanced or Synplify Premier Physical Plus specified at the bottom of this file.
- The Physical Plus congestion estimation report (congestion_est_phys.rpt) is generated in the implementation directory. This congestion report contains a list of high-fanout nets that can cause potential routing congestion problems globally in the design. However, use the Routing Congestion implementation map within the Physical Analyst tool, which displays estimated routing congestion based on placement to identify local routing congestion problems in the design. For details, see *Using the Routing Congestion Map*, on page 713.

Physical Plus Dependencies

Here are some Physical Plus dependencies to consider:

- The Physical Plus and Physical Synthesis options are mutually exclusive. These features must be used independently. The GUI ensures that when one of these switches is enabled, the other option is disabled.

Copyright © 2013 Synopsys, Inc.

- Physical Plus runs the Synplify global placer. To run the Xilinx global placer, you must set the following attribute in the SDC file:

  ```
  define_global_attribute {syn_use_xilinx_placement} {1}
  ```

- If it is paramount that routing completes but timing constraints can be conceded somewhat; Physical Plus provides a special PDC attribute that optimizes for routing at the expense of timing. Add the following attribute to the PDC file:

  ```
  define_global_attribute syn_placer_effort_level {DECONGEST}
  ```

  Use this attribute for designs that are unroutable; set this attribute to DECONGEST to determine if routing improvements are possible for the design.

- Use the following attribute so that Synplify global placement only places the FIFO, RAM, and DSP blocks in the design and forward-annotates them for PAR:

  ```
  define_global_attribute syn_only_large_block_placement {1}
  ```

  This can provide quality of results (QoR) improvements for the design.

- Each implementation is limited to one Physical Plus run. This restriction is based on the usage model of a single logic synthesis run followed by a Physical Plus run. To make sure you preserve your current results, see *Saving Physical Plus Data Through Iterations*, on page 85.

Saving Physical Plus Data Through Iterations

The default input for Physical Plus is the logic synthesis `edf` file (for example, */rev1/top_level.edf) for the current implementation. By default, any iteration of Physical Plus overwrites the previous results. If you need to rerun Physical Plus and want to preserve the current results from being overwritten, do one of the following:

- Create a new implementation and click Run. This reruns logic synthesis and therefore has a runtime penalty.
- To save on logic synthesis runtime, copy all EDIF files from the current implementation to the new implementation. Disable logic synthesis in the Process View.
- Manually back up the results from the physical_plus and physical_pr directories and rerun the same implementation.

Copyright © 2013 Synopsys, Inc.

Hierarchical Project Management Flows

Some Altera, Lattice iCE65/iCE40, and Xilinx technologies

As designs grow in size and complexity, the industry uses team design and parallel development techniques to ensure that designs are finished on time. Typically, designs are split into smaller sub-projects or blocks, and different teams work on different blocks.

The team design approach could be either top-first or block-first. The hierarchical project management capabilities (HPM) in the FPGA synthesis tools facilitate both the top-first or block-first team design approaches. See the following for details:

- Block-First Development Flow for Hierarchical Projects, on page 88
- Top-First Development Flow for Hierarchical Projects, on page 89
- Top-Down Synthesis for Hierarchical Projects, on page 93
- Bottom-Up Synthesis for Hierarchical Projects, on page 91
- Mixed Block Synthesis for Hierarchical Projects, on page 95

Hierarchical Project Flows and Compile Points

The compile point and the hierarchical project management flows are both modular and support the team design approach, but the focus is slightly different. Compile point flows are block-based, but hierarchical project management flows focus on managing the entirety of the design. Hierarchical project management can use compile points to implement a hierarchical design.

- Hierarchical project management
  Hierarchical project management consists of various features and methodologies that help you develop and automatically manage a single FPGA project using multiple teams across different geographical areas. These flows are intended for distributed design development, and for parallel development of portions of the design. You create partitions at the RTL level, but you do not have to floorplan them. The hierarchical project management flows are modular, and can use compile points for its block-level components. Hierarchical project management includes features like a GUI that supports hierarchical project management and module import and export, top-first, block-first and mixed development flows, and both top-down and bottom-up synthesis flows (see the list in

Hierarchical Project Management Flows, on page 86 for information on these flows).

- Compile points
  Compile points implement block-based flows, based on RTL partitions that are defined prior to synthesis. Compile points are often used to implement incremental team design changes, or in design preservation flows like the Xilinx Design Preservation flow. They can also be used to reduce runtime. Depending on the tool you use, you can have manual or automatic compile points:
  - Manual compile points (*Synplify Pro and Synplify Premier*)
    You manually define these compile points and synthesize them in logic synthesis flows. Additionally, Synplify Premier users can improve runtime by synthesizing the compile points in parallel on multiprocessors on a single machine.
  - Automatic compile points (*Synplify Pro and Synplify Premier*)
    If you have multiple licenses, you can use this mode to speed up runtime. The tool automatically splits the design into compile points and runs parallel logic synthesis on multiple processors to reduce runtime. The final design is merged back together. See *The Automatic Compile Point Flow*, on page 1109 for details about automatic compile points.

Copyright © 2013 Synopsys, Inc.

Block-First Development Flow for Hierarchical Projects

In this design approach, blocks are developed independently and then stitched together at the top level. The advantage to this approach is independence, parallel development, and fewer design constraints at the team level. In the following example, different teams work on each block and the blocks are then integrated into the top level.

1. Create independent projects from blocks. For each block, do the following:
   - Add files.
   - Synthesize the block and check that results meet block-level goals. Iterate as needed.
2. Create the top-level project.
   - Add top-level source files, with black boxes defined for sub-project blocks.
   - Compile the top level to create a hierarchical view.
3. Link sub-project to the top level.
   - Open sub-projects in the same synthesis window as the top-level design.
   - Drag and drop sub-projects into the top-level project.
   - Save the top level and compile it so that you can see the hierarchical view.
4. To experiment with different settings, use multiple implementations at the subproject level or at the top level.

   For details, see *Working with Multiple Implementations in Hierarchical Projects*, on page 270.
5. Synthesize the top level.

- For top-down synthesis, the tool runs the design flat. For details about running top-down synthesis, see *Top-Down Synthesis for Hierarchical Projects*, on page 93.
- For bottom-up synthesis, the tool first synthesizes the blocks and generates output netlists for them. It then synthesizes the top level, based on the top-level RTL files and the netlist files from the sub-project blocks.For details about running bottom-up synthesis, see *Bottom-Up Synthesis for Hierarchical Projects*, on page 91.
- If your design contains a combination of top-down and bottom-up blocks, the tool first compiles the top-down blocks, then works on the bottom-up blocks, and finally synthesizes the top level. See *Mixed Block Synthesis for Hierarchical Projects*, on page 95 for details.

Top-First Development Flow for Hierarchical Projects

In this flow, you start with the top-level design and then divide it up into sub-projects (blocks). The block RTL is included in the top-level project. The advantage to using this approach is that you can take advantage of boundary optimizations to improve QOR.

1. Create a top-level project.
   - Add project files. You can have black boxes for the sub-project blocks.
   - Compile the design.
2. Open the hierarchical view and define the sub-projects.
   - Define the sub-projects. as described in See *Creating Hierarchical Subprojects by Exporting Blocks*, on page 254 and *Creating Hierarchical Subprojects by Exporting Instances*, on page 257 for information about creating the subprojects.

Copyright © 2013 Synopsys, Inc.

You must repeat the process for each sub-project block you want to create. If your top level has multiple instances of the same block with different parameter values, you must create sub-projects for each instance as you need separate output netlists for each instance at the top level. You do not need to create sub-projects for each instance if multiple instances have the same parameters or no parameters, unless you want them to have their own output netlist files.

- Save the top-level design.
- Send out the blocks you created to be developed independently.

3. Develop each block independently. The teams must do the following for each block they are working on:
   - Add source files.
   - Add constraints.
   - Synthesize the block and check that results meet block-level goals. Iterate as needed.
4. Optionally, create multiple implementations for your hierarchical project. You might want to do this if you want to experiment with different settings. See *Working with Multiple Implementations in Hierarchical Projects*, on page 270.
5. Synthesize the top level.
   - For top-down synthesis, the tool runs the design flat. For details about running top-down synthesis, see *Top-Down Synthesis for Hierarchical Projects*, on page 93.
   - For bottom-up synthesis, the tool first synthesizes the blocks and generates output netlists for them. It then synthesizes the top level, based on the top-level RTL files and the netlist files from the sub-project blocks.For details about running bottom-up synthesis, see *Bottom-Up Synthesis for Hierarchical Projects*, on page 91.
   - If your design contains a combination of top-down and bottom-up blocks, the tool first compiles the top-down blocks, then works on the bottom-up blocks and finally synthesizes the top level. See *Mixed Block Synthesis for Hierarchical Projects*, on page 95 for details.

Copyright © 2013 Synopsys, Inc.

Bottom-Up Synthesis for Hierarchical Projects

In this flow, you synthesize your design from the bottom up. This flow is independent of the approach used to develop the design; you can use bottom-up synthesis for a design developed with either the top-first or block-first design approach.

1. Open the top-level project.

   The top-level project can be developed from the top down or the bottom up (see *Top-First Development Flow for Hierarchical Projects*, on page 89 and *Block-First Development Flow for Hierarchical Projects*, on page 88 for details).

2. Check that the top-level entity or module is specified in the Implementation Options dialog box.

3. Configure the synthesis run parameters as described in *Configuring Synthesis Runs for Hierarchical Projects*, on page 278.
   - Select Hierarchical Project Options and specify which block implementation to run.
   - Set Run Type for the block to bottom_up.

Copyright © 2013 Synopsys, Inc.

- If necessary, synchronize block options with the top level with the Sync all Options to Top and Syn Required Options buttons. If the block options do not match the top-level settings, you could get errors.
- Click OK.

4. If the sub-project teams are not working in the default directories, do one of the following before running synthesis:
   - Use source control management and update the block projects before running top-level synthesis.
   - Copy the block output netlist and constraint files to the sub-project directories.
5. Synthesize the top-level design. Click Run to synthesize the top-level design.

By default, the tool synthesizes the active implementations of the top level and blocks. If you want to run all implementations, select Run->Run all Implementations instead of clicking Run.

For the bottom-up flow, the tool does the following:

- It automatically uses the latest output netlists for the sub-projects that were marked bottom-up when you configured settings for the run. It needed, it resynthesizes the blocks and generates the output netlists for them.
- It then compiles the top level, based on the top-level RTL files and the netlist files from the sub-project blocks.
- Finally, it maps the top level based on the compiled `srs` for the top level.

6. Analyze the results using the log file and the RTL view, as described in *Analyzing Synthesis Results for Hierarchical Projects*, on page 282.

Top-Down Synthesis for Hierarchical Projects

In this flow, you synthesize the design from the top down. Note that this flow is independent of the approach used to develop the design; you can use top-down synthesis for a design developed with either the top-first or block-first approach.

1. Open the top-level project.

   The top-level project can be developed from the top down or the bottom up (see *Top-First Development Flow for Hierarchical Projects*, on page 89 and *Block-First Development Flow for Hierarchical Projects*, on page 88 for details).

2. Check that the top-level entity or module is specified in the Implementation Options dialog box.

3. Configure the synthesis run parameters as described in *Configuring Synthesis Runs for Hierarchical Projects*, on page 278.
   - Select Hierarchical Project Options and specify which block implementation to run.
   - Set Run Type for the block to top_down.

Copyright © 2013 Synopsys, Inc.

– If necessary, synchronize block options with the top level with the Sync all Options to Top and Syn Required Options buttons. If the block options do not match the top-level settings, you could get errors.

– Click OK.

4. Click Run to synthesize the top-level design.

   By default, the tool synthesizes the active implementations of the top level and blocks. If you want to run all implementations, select Run->Run all Implementations instead of clicking Run.

   For the top-down flow, the tool runs the design flat. It compiles the block RTL and constraint files along with the top-level files, as if they were part of the top-level project.

5. Analyze the results using the log file and the RTL view, as described in *Analyzing Synthesis Results for Hierarchical Projects*, on page 282.

Mixed Block Synthesis for Hierarchical Projects

In this flow, you have some blocks that need to be synthesized from the top down and others that must be synthesized from the bottom up. Note that this flow refers to the synthesis process and is independent of the approach used to develop the design.

1. Open the top-level project.

   The top-level project and blocks can be developed from the top down or the bottom up (see *Top-First Development Flow for Hierarchical Projects*, on page 89 and *Block-First Development Flow for Hierarchical Projects*, on page 88 for details).

2. Check that the top-level entity or module is specified in the Implementation Options dialog box.

3. Configure the synthesis run parameters as described in *Configuring Synthesis Runs for Hierarchical Projects*, on page 278.

	camera_open_source_	Sub: camera_wb_if	Sub: camera_cb_table
Implementation:	rev_1	rev	rev_1
Run Type:		top_down	bottom_up
Options:			
Technology	Virtex6	Virtex6	Virtex6
Technology part	XC6VLX75T	XC6VLX75T	XC6VLX75T
Technology package	FF484	FF484	FF484
Fast Synthesis	☐	☐	☐
Synplify Global Placement	☐	☐	☐
Physical Synthesis	☐	☐	☐
Enhanced Optimization	☑	☑	☑
Fanout Guide	10000	10000	10000
Disable I/O Insertion	☐	☐	☐
FSM Explorer	☐	☐	☐
Pipelining	☐	☐	☐
Retiming	☐	☐	☐
Fix Gated Clocks	0	0	0
Fix Generated Clocks	3	3	3
Auto Constrain IO	☐	☐	☐
Disable Sequential Optimizations	☐	☐	☐

Copyright © 2013 Synopsys, Inc.

- Select Hierarchical Project Options and specify the block implementations to run.
- Set Run Type for each block appropriately. In a mixed design, some blocks will be top_down and others bottom_up.
- If necessary, synchronize block options with the top level with the Sync all Options to Top and Syn Required Options buttons. If the block options do not match the top-level settings, you could get errors. Click OK.

4. Click Run to synthesize the top-level design.

   By default, the tool synthesizes the active implementations of the top level and blocks. If you want to run all implementations, select Run->Run all Implementations instead of clicking Run.

   The synthesis tool does the following to synthesize a hierarchical design with mixed blocks:

   - The tool compiles the top-down blocks to generate `srs` files for them.
   - It automatically picks up the latest output netlists for the bottom-up blocks. If necessary, it synthesizes these blocks and generates output netlists for them.
   - It then compiles the top level, using the top-level RTL files, the `srs` files for the top-down blocks, and the netlist files for the bottom-up blocks.
   - Finally, it maps the top level based on the compiled top level results (`srs`) from the previous phase.

5. Analyze the results using the log file and the RTL view, as described in *Analyzing Synthesis Results for Hierarchical Projects*, on page 282.

Prototyping Flow

The prototyping flow uses the Synplify Premier products and Identify tool set to provide a complete design and verification environment. You can use the flow for single FPGA prototypes. For partitioning and timing optimizations in multi-FPGA designs, use the Certify product.

The prototyping flow provides support for the following:

- Instrumentation and debug of an operating FPGA directly from the source code.
- Use of HAPS prototyping boards.
- Use of numerous daughter boards.
- View the internal designs at full speed.
- Support gated clock conversions and DesignWare.
- Debug and display results in a waveform viewer.

Copyright © 2013 Synopsys, Inc.

CHAPTER 3

Preparing the Input

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

- Setting Up HDL Source Files, on page 100
- Using Mixed Language Source Files, on page 117
- Working with Constraint Files, on page 122
- Using Input from Related Tools, on page 127

Copyright © 2013 Synopsys, Inc.

Setting Up HDL Source Files

This section describes how to set up your source files; project file setup is described in *Setting Up Project Files*, on page 212. Source files can be in Verilog or VHDL. For information about structuring the files for synthesis, refer to the *Reference Manual*. This section discusses the following topics:

- Creating HDL Source Files, on page 100
- Using the Compiler Directives Editor, on page 102
- Using the Context Help Editor, on page 108
- Checking HDL Source Files, on page 110
- Editing HDL Source Files with the Built-in Text Editor, on page 111
- Using an External Text Editor, on page 116
- Setting Editing Window Preferences, on page 114

Creating HDL Source Files

This section describes how to use the built-in text editor to create source files, but does not go into details of what the files contain. For details of what you can and cannot include, as well as vendor-specific information, see the *Reference Manual*. If you already have source files, you can use the text editor to check the syntax or edit the file (see *Checking HDL Source Files*, on page 110 and *Editing HDL Source Files with the Built-in Text Editor*, on page 111).

You can use Verilog or VHDL for your source files. The files have `v` (Verilog) or `vhd` (VHDL) file extensions, respectively. With the Synplify Premier and Synplify Pro products, you can use Verilog and VHDL files in the same design. For information about using a mixture of Verilog and VHDL input files, see *Using Mixed Language Source Files*, on page 117.

1. To create a new source file either click the HDL file icon ( ) or do the following:
   - Select File->New or press Ctrl-n.
   - In the New dialog box, select the kind of source file you want to create, Verilog or VHDL. Note that you can use the Context Help Editor for Verilog designs that contain SystemVerilog constructs in the source

file. For more information, see *Using the Context Help Editor*, on page 108.

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

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

2. Type the source information in the window, or cut and paste it. See *Editing HDL Source Files with the Built-in Text Editor*, on page 111 for more information on working in the Editing window.

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

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

Once you have created a source file, you can check that you have the right syntax, as described in *Checking HDL Source Files*, on page 110.

Copyright © 2013 Synopsys, Inc.

Using the Compiler Directives Editor

The Compiler Directives Editor is an advanced text file editor used for compiler directives. The compiler directives file provides a convenient way to specify supported directives to be added to the source code. During compilation, the tool passes all active compiler directives files to the compiler.

Note: This feature is available in the Synplify Premier tools.

To use the Compiler Directives Editor:

1. Use File->New and select the file type of Compiler Directives.
2. Specify a new file name and click OK.

   A text editor opens for this new compiler directives (cdc) file.

3. You can specify compiler directives as follows:
   - Type the command; after you type three characters a popup menu displays the compiler directives command list. Select the command.
   - When you hover over a command, a tool tip is displayed for the selected command. The automatic command completion for the compiler directive values is not currently available.

Command List Popup Menu

Command Tool Tip

For more information about the command syntax, see *Compiler Directives Syntax*, on page 105.

4. You can also specify a command by using the compiler directives browser that displays a command list (the compiler directives syntax window does not currently support the command values). Click on the Hide Syntax Help button at the bottom of the editor window to close the syntax help browser.

Copyright © 2013 Synopsys, Inc.

5. When you save this file, the cdc file gets added to your project in the Compiler Directive directory if the Add to Project option was checked on the New dialog box. Thereafter, you can double-click on the cdc file to open this file in the text editor.

Alternative Method for Creating the Compiler Directives File

Alternatively, you can

1. Use the New Constraint File icon ( ) to select the type of constraint file to edit. The following dialog box is displayed.

2. Click on
   - Compiler Directives — to open the compiler directives editor
   - Constraint File (SCOPE) — to open the FPGA SCOPE constraint editor

   For information about using the FPGA SCOPE editor and compiler directives editor, see *Using the SCOPE Editor*, on page 130 and *Using the Compiler Directives Editor*, on page 102.

Compiler Directives Syntax

Compiler directives support the following directives:

- *syn_black_box*
- *syn_noprune*
- *syn_preserve*
- *syn_sharing*
- *syn_rename_module*
- *syn_unique_inst_module*

These compiler directives take precedence over the RTL directives, whenever a conflict exists.

Copyright © 2013 Synopsys, Inc.

Example

For example, use the syn_black_box directive to specify that a module or entity is a black box with only its interface defined for synthesis. The contents of the black box cannot be optimized during synthesis. This directive is associated with a module or entity or architecture of an entity in the cdc file and not an instance or component.

The syn_black_box directive is specified with an implicit Boolean value of 1 or true.

VHDL

define_directive {v:*libraryName***.***entityName***(***architectureName***)}**
{syn_black_box} {1|0}

Where:

- *libraryName* – optionally, specifies the name of the library. If undefined, this defaults to all the design libraries.
- *entityName* – defines the name for the entity.
- *architectureName* – defines the name of the architecture for the black box.
- **1|0** – specify 1 to define a module or component as a black box.
- VHDL objects are case insensitive.

Verilog/SystemVerilog

define_directive {v:*libraryName***.***moduleName***} {syn_black_box} {1|0}**

Where:

- *libraryName* – optionally, specifies the name of the library. If undefined, this defaults to all the design libraries.
- *moduleName* – defines the name for the module.
- **1/0** – specify 1 to define a module or component as a black box.
- Verilog objects are case sensitive.

For Verilog and VHDL examples, see *Compiler Directives File Examples*, on page 1168.

A message is generated

- if the same entity in the cdc file is found in more than one library (*warning*)
- if the same entity in the cdc file has more than one architecture (*warning*)
- for any illegal or unsupported commands found in the cdc file during compilation (*error*)

Future releases will not support the define_attribute command in the cdc file (*warning*) – t is recommended that you use define_directive instead.

When the compiler directives cdc file is added to the Project file, you can enable/disable a specific cdc file on the Constraints tab of the Implementation Options panel.

Copyright © 2013 Synopsys, Inc.

Using the Context Help Editor

When you create or open a Verilog design file, use the Context Help button displayed at the bottom of the window to help you code with SystemVerilog constructs in the source file. This feature is currently supported for some of the SystemVerilog constructs.

To use the Context Help Editor:

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

2. When you select a construct in the left-side of the window, the online help description for the construct is displayed. If the selected construct has this feature enabled, the online help topic is displayed on the top of the window and a generic code template for that construct is displayed at the bottom.

3. The Insert Template button is also enabled. When you click the Insert Template button, the code shown in the template window is inserted into your SystemVerilog file at the location of the cursor. This allows you to easily insert code and modify it for the design that you are going to synthesize.

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

Copyright © 2013 Synopsys, Inc.

Checking HDL Source Files

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

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

   The software detects syntax errors such as incorrect keywords and punctuation and reports any errors in a separate log file (syntax.log). If no errors are detected, a successful syntax check is reported at the bottom of this file.
3. To run a synthesis check, select Run->Synthesis Check or press Shift+F8.

   The software detects hardware-related errors such as incorrectly coded flip-flops and reports any errors in a separate log file (syntax.log). If there are no errors, a successful syntax check is reported at the bottom of this file.
4. Review the errors by opening the syntax.log file when prompted and use Find to locate the error message (search for @E). Double-click on the 5-character error code or click on the message text and push F1 to display online error message help.
5. Locate the portion of code responsible for the error by double-clicking on the message text in the syntax.log file. The Text Editor window opens the appropriate source file and highlights the code that caused the error.
6. Repeat steps 4 and 5 until all syntax and synthesis errors are corrected.

Copyright © 2013 Synopsys, Inc.

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

Editing HDL Source Files with the Built-in Text Editor

The built-in text editor makes it easy to create your HDL source code, view it, or edit it when you need to fix errors. If you want to use an external text editor, see *Using an External Text Editor*, on page 116.

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

   The Text Editor window opens and displays the source file. Lines are numbered. Keywords are in blue, and comments in green. String values are in red. If you want to change these colors, see *Setting Editing Window Preferences*, on page 114.

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

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

Copyright © 2013 Synopsys, Inc.

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

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

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

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

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

5. To fix errors or review warnings in the source code, do the following:
   - Open the HDL file with the error or warning by double-clicking the file in the project list.
   - Press F5 to go to the first error, warning, or note in the file. At the bottom of the Editing window, you see the message text.
   - To go to the next error, warning, or note, select Run->Next Error/Warning or press F5. If there are no more messages in the file, you see the message “No More Errors/Warnings/Notes” at the bottom of the Editing window. Select Run->Next Error/Warning or press F5 to go to the the error, warning, or note in the next file.
   - To navigate back to a previous error, warning, or note, select Run->Previous Error/Warning or press Shift-F5.
6. To bring up error message help for a full description of the error, warning, or note:
   - Open the text-format log file (click View Log) and either double click on the 5-character error code or click on the message text and press F1.

Copyright © 2013 Synopsys, Inc.

- Open the HTML log file (not available with the Synplify product) and click on the 5-character error code.
- In the Tcl window (not available with the Synplify product), click the Messages tab and click on the 5-character error code in the ID column.

7. To cross probe from the source code window to other views, open the view and select the piece of code. See *Crossprobing from the Text Editor Window*, on page 584 for details.

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

Setting Editing Window Preferences

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

1. Select Options->Editor Options and either Synopsys Editor or External Editor. For more information about the external editor, see *Using an External Text Editor*, on page 116.

2. Then depending on the type of file you open, you can to set the background, syntax coloring, and font preferences to use with the text editor.

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

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

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

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

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

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

6. Click OK on the Editor Options form.

Copyright © 2013 Synopsys, Inc.

Using an External Text Editor

You can use an external text editor like vi or emacs instead of the built-in text editor. Do the following to enable an external text editor. For information about using the built-in text editor, see *Editing HDL Source Files with the Built-in Text Editor*, on page 111.

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

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

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

3. Click OK.

Using Mixed Language Source Files

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

1. Remember that Verilog does not support unconstrained VHDL ports and set up the mixed language design files accordingly.

2. If you want to organize the Verilog and VHDL files in different folders, select Options->Project View Options and toggle on the View Project Files in Folders option.

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

3. When you open a project or create a new one, add the Verilog and VHDL files as follows:
   - Select the Project->Add Source File command or click the Add File button.
   - On the form, set Files of Type to HDL Files (*.vhd, *.vhdl, *.v).
   - Select the Verilog and VHDL files you want and add them to your project. Click OK. For details about adding files to a project, see *Making Changes to a Project*, on page 216.

Copyright © 2013 Synopsys, Inc.

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

4. When you set device options (Implementation Options button), specify the top-level module. For more information about setting device options, see *Setting Logic Synthesis Implementation Options*, on page 228.
   - If the top-level module is Verilog, click the Verilog tab and type the name of the top-level module.
   - If the top-level module is VHDL, click the VHDL tab and type the name of the top-level entity. If the top-level module is not located in the default work library, you must specify the library where the compiler can find the module. For information on how to do this, see *VHDL Panel*, on page 207.

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

5. Select the Implementation Results tab on the same form and select one output HDL format for the output files generated by the software. For more information about setting device options, see *Setting Logic Synthesis Implementation Options*, on page 228.
   - For a Verilog output netlist, select Write Verilog Netlist.
   - For a VHDL output netlist, select Write VHDL Netlist.
   - Set any other device options and click OK.

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

6. If you run into problems, see *Troubleshooting Mixed Language Designs*, on page 120 for additional information and tips.

Copyright © 2013 Synopsys, Inc.

Troubleshooting Mixed Language Designs

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

VHDL File Order

For VHDL-only designs or mixed designs where the top level is not specified, the FPGA synthesis tools automatically re-arrange the VHDL files so that the VHDL packages are compiled in the correct order.

However, if you have a mixed-language design where you have specified the top level, you must specify the VHDL file order for the tool. You only need to do this once, by selecting the Run->Arrange VHDL files command. If you do not do this, you get an error message

VHDL Global Signals

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

Passing VHDL Boolean Generics to Verilog Parameters

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

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

VHDL Entity Declaration	Verilog Instantiation

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

Passing VHDL Generics Without Inferring a Black Box

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

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

Copyright © 2013 Synopsys, Inc.

Working with Constraint Files

Constraint files are text files that are automatically generated by the SCOPE interface (see *Specifying SCOPE Constraints*, on page 135), or which you create manually with a text editor. They contain Tcl commands or attributes that constrain the synthesis run. Alternatively, you can set constraints in the source code, but this is not the preferred method.

This section contains information about

- When to Use Constraint Files over Source Code, on page 122
- Using a Text Editor for Constraint Files (Legacy), on page 122
- Tcl Syntax Guidelines for Constraint Files, on page 123
- Generating Constraint Files for Forward Annotation, on page 125

When to Use Constraint Files over Source Code

You can add constraints in constraint files (generated by SCOPE interface or entered in a text editor) or in the source code. In general, it is better to use constraint files, because you do not have to recompile for the constraints to take effect. It also makes your source code more portable. See *Using the SCOPE Editor*, on page 130 for more information.

However, if you have black box timing constraints like syn_tco, syn_tpd, and syn_tsu, you must enter them as directives in the source code. Unlike attributes, directives can only be added to the source code, not to constraint files. See *Specifying Attributes and Directives*, on page 245 for more information on adding directives to source code.

Using a Text Editor for Constraint Files (Legacy)

You can use the Legacy SCOPE editor for the SDC constraint files created before release version G-2012.09. However, it is recommended that you translate your SDC files to FDC files to enable the latest version of the SCOPE editor and to utilize the enhanced timing constraint handling in the tool.

If you choose to use the legacy SCOPE editor, this section shows you how to manually create a Tcl constraint file. The software automatically creates this file if you use the legacy SCOPE editor to enter the constraints. The Tcl

Copyright © 2013 Synopsys, Inc.

constraint file only contains general timing constraints. Black box constraints must be entered in the source code. For additional information, see *When to Use Constraint Files over Source Code*, on page 122.

1. Open a file for editing.
   - Make sure you have closed the SCOPE window, or you could overwrite previous constraints.
   - To create a new file, select File->New, and select the Constraints File (SCOPE) option. Type a name for the file and click OK.
   - To edit an existing file, select File->Open, set the Files of Type filter to Constraint Files (`sdc`) and open the file you want.
2. Follow the syntax guidelines in *Tcl Syntax Guidelines for Constraint Files*, on page 123.
3. Enter the timing constraints you need. For the syntax, see the *Reference Manual*. If you have black box timing constraints, you must enter them in the source code.
4. You can also add vendor-specific attributes in the constraint file using define_attribute. See *Specifying Attributes in the Constraints File*, on page 253 for more information.
5. Save the file.
6. Add the file to the project as described in *Making Changes to a Project*, on page 216, and run synthesis.

Tcl Syntax Guidelines for Constraint Files

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

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

Copyright © 2013 Synopsys, Inc.

– In Verilog modules, use the following syntax for instance, port, and net names:

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

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

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

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

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

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

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

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

Checking Constraint Files

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

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

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

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

   For details on this report, see *Constraint Checking Report*, on page 629 of the *Reference Manual*.

Generating Constraint Files for Forward Annotation

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

1. Set attributes to control forward annotation.
   – To forward-annotate timing constraints for Microsemi Axcelerator, Fusion, ProASIC (500K), ProASIC Plus (PA), and ProASIC3/3E /3L families, set the clock period, max delay, input delay, output delay, multiple-cycle paths, and false paths in the SCOPE interface.
   – To forward-annotate I/O constraints (define_input_delay and define_output_delay) to the `tcl` file for Altera designs or the synplicity.ucf file for Xilinx designs, set syn_forward_io_constraints with a value of 1 on the top level of the design or as a global attribute.
   – To forward-annotate clocks for Xilinx DCMs and DLLs, define the clock at the primary inputs and any Xilinx phase shift and frequency

Copyright © 2013 Synopsys, Inc.

multiplication properties you need. See *Defining Other Clock Requirements*, on page 185 for details. The synthesis software forward-annotates the DLL/DCM inputs.

– To forward-annotate clocks for Altera PLLs, define the input frequency value. See *Defining Other Clock Requirements*, on page 185 for details. The synthesis software forward-annotates the PLL inputs.

– For some Lattice designs, set the -from and -to false path and multi-cycle constraints on the Others tab of the SCOPE interface.

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

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

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

3. Click OK and run synthesis.

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

   – `acf` file for Altera

   – *filename*_sdc.sdc file for Microsemi

   – synplicity.ucf file for Xilinx

   – $DESIGN_synplify.lpf file for Lattice

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

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

Using Input from Related Tools

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

Using Input from System Designer

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

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

Using Input from the Synphony Model Compiler Tool

The Synphony Model Compiler tool generates IP for DSP designs. Include this in your synthesis project by adding the source files generated by these tools.

Copyright © 2013 Synopsys, Inc.

CHAPTER 4

Specifying Constraints

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

- Using the SCOPE Editor, on page 130
- Specifying SCOPE Constraints, on page 135
- Specifying Timing Exceptions, on page 146
- Finding Objects with Tcl find and expand, on page 152
- Using Collections, on page 161
- Converting SDC to FDC, on page 171
- Using the SCOPE Editor (Legacy), on page 173
- Translating XDC Constraints to FDC, on page 190
- Translating Altera QSF Constraints, on page 193
- Specifying Xilinx Constraints (Legacy), on page 195
- Converting and Using Xilinx UCF Constraints, on page 200

The following chapters discuss related information:

- Chapter 5, *Constraints*, for an overview of constraints
- Chapter 6, *SCOPE Constraints Editor*, for a description of the SCOPE editor

Copyright © 2013 Synopsys, Inc.

Using the SCOPE Editor

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

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

- For existing designs, run the sdc2fdc script to translate legacy SDC constraints and create a constraint file that contains Synopsys SDC standard timing constraints and design constraints. For details about this script, see *Converting SDC to FDC*, on page 171.
- For new designs, use the SCOPE editor. See *Creating Constraints in the SCOPE Editor*, on page 130 for more information.

Creating Constraints in the SCOPE Editor

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

1. To create a new constraint file, follow these steps:
   - Compile the design (F7).
   - Open the SCOPE window by:

     Clicking the SCOPE icon in the toolbar ( ).

     This brings up the New Constraint File dialog box; then, select FPGA Design Constraints (Synplify Premier tool only).

     OR

Pressing Ctrl-n or selecting File -> New. This brings up the New dialog box; then, select file type of FPGA Design Constraints (Synplify Premier tool only) and specify a new file name.

Using the SCOPE ICON

(Displays in Synplify Premier Tool Only)

Using File->New

Both of these methods open the SCOPE editor GUI.

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

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

Copyright © 2013 Synopsys, Inc.

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

4. You can enter or edit the following types of constraints:
   - Timing constraints – on the Clocks, Generated Clocks, Inputs/Outputs, Registers, or Delay Paths tab.
   - Design constraints – on the Collections, Attributes, I/O Standards, or Compile Points tab.

   For details about these constraints, see *Specifying SCOPE Constraints*, on page 135.

   For information about ways to enter constraints within the SCOPE editor, see *Guidelines for Entering and Editing Constraints*, on page 143.

5. The free form constraint editor is located in the TCL View tab, which is the last tab in SCOPE. The text editor has a help window on the right-

hand side. For more information about this text editor, see *Using the TCL View of SCOPE GUI*, on page 141.

6. Click on the Check Constraints button to run the constraint checker. The output provides information on how the constraints are interpreted by the tool.

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

Copyright © 2013 Synopsys, Inc.

The following example shows the contents of an FDC file.

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

set_rtl_ff_names {#}
###==== BEGIN Header

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

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

###==== END Header

###==== BEGIN Clocks - (Populated from tab in SCOPE, do not edit)
create_clock -name {clka} {p:clka} -period 10 -waveform {0 5.0}
create_clock -name {clkb} {p:clkb} -period 6.667 -waveform {0 3.3335}
set_clock_groups -derive -name default_clkgroup_0 -asynchronous -group [get_clocks {clka}]
set_clock_groups -derive -name default_clkgroup_1 -asynchronous -group [get_clocks {clkb}]
###==== END Clocks

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

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

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

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

Specifying SCOPE Constraints

Timing constraints define the performance goals for a design. The FPGA synthesis tool supports a subset of the Synopsys SDC Standard timing constraints (for example, create_clock, set_input_delay, and set_false_path). For additional support, see *Synopsys Standard Timing Constraints*, on page 136.

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

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

- Entering and Editing SCOPE Constraints
- Setting Clock and Path Constraints
- Defining Input and Output Constraints
- Specifying Standard I/O Pad Types

To set constraints for timing exceptions like false paths and multicycle paths, see *Specifying Timing Exceptions*, on page 146.

For information about collections, see *Using Collections*, on page 161.

Entering and Editing SCOPE Constraints

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

SCOPE Panel	See...	Tcl Commands
Clocks	Clocks	create_clock set_clock_groups set_clock_latency set_clock_uncertainty
Generated Clocks	Generated Clocks	create_generated_clock
Collections	Collections	define_scope_collection

Copyright © 2013 Synopsys, Inc.

SCOPE Panel	See...	Tcl Commands
Inputs/Outputs	Inputs/Outputs	set_input_delay set_output_delay
Registers	Registers	set_reg_input_delay set_reg_output_delay
Delay Paths	Delay Paths	set_false_path set_max_delay set_multicycle_path
Attributes	Attributes	define_attribute define_global_attribute
I/O Standards	I/O Standards	define_io_standard
Compile Points	Compile Points	define_compile_point define_current_design
TCL View	TCL View	--

Synopsys Standard Timing Constraints

The FPGA synthesis tools support Synopsys standard timing constraints for a subset of the clock definition (Clocks and Generated Clocks), I/O delay (Inputs/Outputs), and timing exception constraints (Delay Paths). For complete information about using the FPGA timing constraints with your project, see:

- For specific information on individual constraint options and arguments, see the *Synthesis Commands* PDF document at https://solvnet.synopsys.com/dow_retrieve/G-2012.06/manpages/ni/syn2.pdf.
- For information on which options and arguments are supported, see the *SDC Standard for FPGA Synthesis* document on SolvNet.
- For general information on the Design Constraints Format, see the *Using the Synopsys Design Constraints Format Application Note* on SolvNet.

Setting Clock and Path Constraints

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

To define...	Pane	Do this to set the constraint...
Clocks	Clock	Select the clock object (Clock). Specify a clock name (Clock Alias), if required. Type a period (Period). Change the rise and fall edge times for the clock. waveforms of the clock in nanoseconds, if needed. Change the default clock group, if needed. Check the Enabled box. See *Defining Clocks*, on page 179 for information about clock attributes.
Generated Clocks	Generated Clocks	Select the generated clock object. Specify the master clock source (a clock source pin in the design). Specify whether to use invert for the generated clock signal. Specify whether to use: edges, divide_by, or multiply_by. Check the Enabled box.
Input/output delays	Inputs/ Outputs	See *Defining Input and Output Constraints (Legacy)*, on page 186 for information about setting I/O constraints.
Maximum path delay	Delay Paths	Select the Delay Type path of Max Delay. Select the start/from point for either a port or register (From/Through). See *Defining From/To/Through Points for Timing Exceptions*, on page 146 for more information. Select the end/to point for either an output port or register. Specify a through point for a net or hierarchical port/pin (To/Through). Set the delay value (Max Delay). Check the Enabled box.
Multicycle paths	Delay Paths	See *Defining Multicycle Paths*, on page 150.

Copyright © 2013 Synopsys, Inc.

To define...	Pane	Do this to set the constraint...
False paths	Delay Paths	See *Defining False Paths*, on page 151 for details.
Global attributes	Attributes	Set Object Type to <global>. Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.
Attributes	Attributes	Do either of the following: • Select the type of object (Object Type). Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box. • Set the attribute (Attribute) and its value (Value). Select the object (Object). Check the Enabled box.

Defining Input and Output Constraints

In addition to setting I/O delays in the SCOPE window as described in *Setting Clock and Path Constraints*, on page 177, you can also set the Use clock period for unconstrained IO option.

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

  You can specify multiple constraints for the same port. The software applies all the constraints; the tightest constraint determines the worst slack. If there are multiple constraints from different levels, the most specific overrides the more global. For example, if there are two bit constraints and two port constraints, the two bit constraints override the two port constraints for that bit. The other bits get the two port constraints.
- Specify the constraint value in the SCOPE window:
  - Select the type of delay: input or output (Type).
  - Type a delay value (Value).
  - Check the Enabled box, and save the constraint file in the project.

  Make sure to specify explicit constraints for each I/O path you want to constrain.
- To determine how the I/O constraints are used during synthesis, do the following:
  - Select Project->Implementation Options, and click Constraints.
  - To use only the explicitly defined constraints disable Use clock period for unconstrained IO.
  - To synthesize with all the constraints, using the clock period for all I/O paths that do not have an explicit constraint enable Use clock period for unconstrained IO.
  - Synthesize the design. When you forward-annotate the constraints, the constraints used for synthesis are forward-annotated for place-and-route.

Copyright © 2013 Synopsys, Inc.

- Input or output ports with explicitly defined constraints, but without a reference clock (-ref option) are included in the System clock domain and are considered to belong to every defined or inferred clock group.
- If you do not meet timing goals after place-and-route and you need to adjust the input constraints; do the following:
  - Open the SCOPE window with the input constraint.
  - Use the set_clock_route_delay command to translates the -route option for the constraint, so that you can specify the actual route delay in nanoseconds, as obtained from the place-and-route results. Adding this constraint is equivalent to putting a register delay on the input register.
  - Resynthesize your design.

Specifying Standard I/O Pad Types

You can specify a standard I/O pad type to use in the design. The equivalent Tcl command is define_io_standard.

1. Open the SCOPE window and go to the I/O Standard tab.
2. In the Port column, select the port. This determines the port type in the Type column.
3. Enter an appropriate I/O pad type in the I/O Standard column. The Description column shows a description of the I/O standard you selected.

   For details of supported I/O standards, see *Industry I/O Standards*, on page 496.
4. Where applicable, set other parameters like drive strength, slew rate, and termination.

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

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

Using the TCL View of SCOPE GUI

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

- Uses dynamic keyword expansion and tool tips for commands that
  - Automatically completes the command from a popup list
  - Displays complete command syntax as a tool tip
  - Displays parameter options for the command from a popup list
  - Includes a keyword command syntax help
- Checks command syntax and uses color indicators that
  - Validates commands and command syntax
  - Distinguishes between FPGA design constraints and SCOPE legacy constraints
- Allows for standard editor commands, such as copy, paste, comment/un-comment a group of lines, and highlighting of keywords

To use the TCL View of the SCOPE GUI:

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

Copyright © 2013 Synopsys, Inc.

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

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

Guidelines for Entering and Editing Constraints

1. Enter or edit constraints as follows:
   - For attribute cells in the spreadsheet, click in the cell and select from the pull-down list of available choices.
   - For object cells in the spreadsheet, click in the cell and select from the pull-down list. When you select from the list, the objects automatically have the proper prefixes in the SCOPE window.

Copyright © 2013 Synopsys, Inc.

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

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

– For cells with values, type in the value or select from the pull-down list.

– Click the check box in the Enabled column to enable the constraint or attribute.

– Make sure you have entered all the essential information for that constraint. Scroll horizontally to check. For example, to set a clock constraint in the Clocks tab, you must fill out Enabled, Clock, Period, and Clock Group. The other columns are optional. For details about setting different kinds of constraints, go to the appropriate section listed in *Specifying SCOPE Constraints*, on page 135.

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

To...	Do...
Cut, copy, paste, undo, or redo	Select the command from the popup (hold down the right mouse button to get the popup) or from the Edit menu.

To...	Do...
Copy the same value down a column	Select Fill Down (Ctrl-d) from the Edit or popup menus.
Insert or delete rows	Select Insert Row or Delete Rows from the Edit or popup menus.
Find text	Select Find from the Edit or popup menus. Type the text you want to find, and click OK.

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

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

Copyright © 2013 Synopsys, Inc.

Specifying Timing Exceptions

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

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

The following shows you how to specify timing exceptions in the SCOPE GUI. For the equivalent Tcl syntax, see Chapter 17, *Batch Commands and Scripts* in the *Reference Manual*.

- Defining From/To/Through Points for Timing Exceptions, on page 146
- Defining Multicycle Paths, on page 150
- Defining False Paths, on page 151

For information about resolving timing exception conflicts, see *Conflict Resolution for Timing Exceptions*, on page 513 in the *Reference Manual*.

Defining From/To/Through Points for Timing Exceptions

For multi-cycle path, false path, and maximum path delay constraints, you must define paths with a combination of From/To/Through points. Whenever the tool encounters a conflict in the way timing-exception constraints are written, see *Conflict Resolution for Timing Exceptions*, on page 513 to determine how resolution occurs based on the priorities defined.

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

- In the From field, identify the starting point for the path. The starting point can be a clock, input or bidirectional port, or register. Only black box output pins are valid. To specify multiple starting points:
  - Such as the bits of a bus, enclose them in square brackets: A[15:0] or A[*].
  - Select the first start point from the HDL Analyst view, then drag and drop this instance into the From cell in SCOPE. For each subsequent

instance, press the Shift key as you drag and drop the instance into the From cell in SCOPE. For example, valid Tcl command format include:

```
set_multicycle_path -from {i:aq i:bq} 2
```

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

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

  For example, if you specify n:net1, the constraint applies to any path passing through net1.
- To specify an OR when constraining a list of through points, you can type the net names in the Through field or you can use the POS UI. To do this:
  - Click in the Through field and click the arrow. This opens the Product of Sums interface.
  - Either type the first net name in a cell in a Prod row or drag the net from an HDL Analyst view into the cell. Repeat this step along the same row, adding other nets in the Sum columns. The nets in each row form an OR list.

Copyright © 2013 Synopsys, Inc.

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

– Click Save.

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

- To specify an AND when constraining a list of through points, type the names in the Through field or do the following:

  – Open the Product of Sums interface as described previously.

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

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

Defining Multicycle Paths

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

1. From the Delay Type pull-down menu, select Multicycle.
2. Select a port or register in the From or To columns, or a net in the Through column. You must set at least one From, To, or Through point. You can use a combination of these points. See *Defining From/To/Through Points for Timing Exceptions*, on page 146 for more information.
3. Select another port or register if needed (From/To/Through).
4. Type the number of clock cycles or nets (Cycles).
5. Specify the clock period to use for the constraint by going to the Start/End column and selecting either Start or End.

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

   multicycle_distance = clock_distance + (cycles -1) * reference_clock_period

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

Defining False Paths

You define false paths by setting constraints explicitly on the Delay Paths tab or implicitly on the Clock tab. See *Defining From/To/Through Points for Timing Exceptions*, on page 146 for object naming and specifying through points.

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

  The software treats this as an explicit false constraint and assigns it the highest priority. Any other constraints on this path are ignored.
- To define a false path between two clocks, select the SCOPE Clocks tab, and assign the clocks to different clock groups:

  The software implicitly assumes a false path between clocks in different clock groups. This false path constraint can be overridden by a maximum path delay constraint, or with an explicit constraint.
- To set an implicit false path on a path to/from an I/O port, do the following:
  - Select Project->Implementation Options->Constraints.
  - Disable Use clock period for unconstrained IO.

Copyright © 2013 Synopsys, Inc.

Finding Objects with Tcl find and expand

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

- Specifying Search Patterns for Tcl find, on page 152
- Refining Tcl Find Results with -filter, on page 154
- Using the Tcl Find Command to Define Collections, on page 155
- Using the Tcl expand Command to Define Collections, on page 157
- Checking Tcl find and expand Results, on page 158
- Using Tcl find and expand in Batch Mode, on page 159

Once you have located objects with the find or expand commands, you can group them into collections, as described in *Using Collections*, on page 161, and apply constraints to all the objects in the collection at the same time.

Specifying Search Patterns for Tcl find

The usage tips in the following table apply for Tcl find search patterns, regardless of whether you specify the find command in the SCOPE window or as a Tcl command. For full details of the command syntax, refer to *Tcl Find Syntax*, on page 1775 of the *Reference Manual*.

Case rules	Use the case rules for the language from which the object was generated: • VHDL: case-insensitive • Verilog: case-sensitive. Make sure that the object name you type in the SCOPE window matches the Verilog name. For mixed language designs, use the case rules for the parent module. The top level for this example is VHDL, so the following command finds any object in the current view that starts with either a or A: `find {a*} -nocase`
Pattern matching	You have two pattern-matching choices: • Specify the -regexp argument, and then use regular expressions for pattern matching. • Do not specify -regexp, and use only the * and ? wildcards for pattern matching. For hierarchical instance names that use dots as separators, the dots must be escaped with a backward slash (\). For example: abc\.d.
Scope of the search	The scope of the search varies, depending on where you enter the command. If you enter it in the SCOPE environment, the scope of the search is the entire database, but if it is entered in the Tcl window, the default scope of the search is the current HDL Analyst view. See *Comparison of Methods for Defining Collections*, on page 161 for a list of the differences. To set the scope to include the hierarchial levels below the current view in HDL Analyst, use the -hier argument. This example finds all objects below the current view that begin with a: `find {a*} -hier`
Restricting search by type of object	Use the *-object_type* argument. The following command finds all nets that contain syn: `find -net {*syn*}`
Restricting search by object property	Use the -filter option, as described in *Refining Tcl Find Results with -filter*, on page 154.

Copyright © 2013 Synopsys, Inc.

Refining Tcl Find Results with -filter

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

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

   - Compile or synthesize the design. After compilation, the tool annotates the design with properties that you can specify with the -filter option, like clock pins.

2. Specify the command using the find pattern as usual, and then specify the -filter option as the last argument:

   find *searchPattern* -filter *expression*
   find *searchPattern* -filter !*expression*

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

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

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

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

For further information about the command, see the following:

For...	See
Tips on using find search patterns	*Specifying Search Patterns for Tcl find*, on page 152
find syntax details	*Tcl find Command*, on page 1774 in the Reference Manual
find -filter syntax details	*Tcl Find -filter Command*, on page 1785in the Reference Manual

Examples of Useful Find -filter Commands

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

Using the Tcl Find Command to Define Collections

It is recommended that you use the SCOPE window rather than the Tcl window described here to specify the find command, for the reasons described in *Comparison of Methods for Defining Collections*, on page 161.

Copyright © 2013 Synopsys, Inc.

The Tcl find command returns a collection of objects. If you want to create a collection of connectivity-based objects, use the Tcl expand command instead of find (*Specifying Search Patterns for Tcl find*, on page 152). This section lists some tips for using the Tcl find command.

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

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

   The result is a random number like s:49078472, which is the collection of objects found. The following table lists some usage tips for specifying the find command. For full details of the syntax, refer to *Tcl Find Syntax*, on page 1775 of the *Reference Manual*.

2. Check your find constraints. See *Checking Tcl find and expand Results*, on page 158.

3. Once you have defined the collection, you can view the objects in the collection, using one of the following methods, which are described in more detail in *Viewing and Manipulating Collections with Tcl Commands*, on page 167:
   - Print the collection using the -print option to the find command.
   - Print the collection without carriage returns or properties, using c_list.
   - Print the collection in columns, with optional properties, using c_print.

4. To manipulate the objects in the collection, use the commands described in *Viewing and Manipulating Collections with Tcl Commands*, on page 167.

5. Combine the Tcl find command with other commands:

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

Using the Tcl expand Command to Define Collections

The Tcl expand command returns a list of objects that are logically connected between the specified expansion points. This section contains tips on using the Tcl expand command to generate a collection of objects that are related by their connectivity. For the syntax details, refer to *Tcl expand Command*, on page 1791 in the *Reference Manual*.

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

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

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

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

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

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

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

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

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

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

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

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

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

4. To print a list of the objects found, either use the -print argument to the expand command, or use the c_print or c_list commands (see *Creating Collections using Tcl Commands*, on page 164).

Copyright © 2013 Synopsys, Inc.

Checking Tcl find and expand Results

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

1. Run the Constraints Checker, either from the UI or at the command line:
   - From the UI, select Run->Constraint Check.
   - At the command line specify the -run constraint_check option to the synthesis tool command. For example: synplify_pro -batch *design*.prj -run constraint_check.
   - If there are issues, the tool reports them in the *design*_cck.rpt report file. Check the Summary and Inapplicable Constraints sections in this file.
2. To list objects selected by the find or expand commands, use one of these methods:
   - List the results by specifying the -print option to the command.
   - List the results with the c_list command.
   - Print out the results one item per line, using the c_print command.
3. To visually validate the objects selected by the find or expand commands, do the following:
   - Run the command and save the results as a collection.
   - On the SCOPE Collections tab, select the collection.
   - Right-click and choose Select in Analyst. The objects in the collection are highlighted in the RTL view. The example below shows high fanout nets that drive more than 20 destinations.

Copyright © 2013 Synopsys, Inc.

Using Tcl find and expand in Batch Mode

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

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

   This batch script uses the find command to find DSP48Es and negative slack, and then writes out the results to separate text files:

```
open_design implementation_a/top.srm
set find_DSP48Es [find -hier -inst{*} -filter @view == {DSP48E*}]
set find_negslack [find -hier -seq -inst {*} -filter @slack
   < {-0.0}]
```


Copyright © 2013 Synopsys, Inc.

```
c_print $find_DSP48Es -file DSP48Es.txt
c_print -prop slack -prop view $find_negslack -file negslack.txt
```

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

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

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

The tool generates two text files as specified, with the results of the two searches. The DSP48s.txt file lists the DSP48Es, and the negslack.txt file lists the instances with negative slack.

Combining Tcl find with Other Operations

When combined with or embedded in other commands, the find command allows you to isolate or group objects, and then manipulate them easily. This section illustrates how to effectively use this command in combination with other commands.

Selecting Objects Using Tcl find

Finding Primitives

Using Collections

Synplify Pro and Synplify Premier only

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

- Creating and Using SCOPE Collections, on page 162
- Creating Collections using Tcl Commands, on page 164
- Viewing and Manipulating Collections with Tcl Commands, on page 167

Comparison of Methods for Defining Collections

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

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

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

Copyright © 2013 Synopsys, Inc.

Creating and Using SCOPE Collections

The following procedure shows you how to define collections in the SCOPE window. The SCOPE method is preferred over typing the commands in the Tcl window (*Creating Collections using Tcl Commands*, on page 164) for the reasons described in *Comparison of Methods for Defining Collections*, on page 161.

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

	Enable	Name	Command	Comment
1	☑	find_all	-hier -inst {*usbSlaveControl.u_endpMux.*}	
2	☑	find_reg	-hier -seq [*usbSlaveControl.u_endppMux.*}	
3	☑	frin_comb	-hier -inst {*usbSlaveControl.u_endpMux.*} -filter @is_combination	
4				
5				

Co Collections

   - In the Commands column, enter the command. See the *Reference Manual* for complete syntax details. Additional information about specifying search patterns is described in *Specifying Search Patterns for Tcl find*, on page 152 and *Specifying Search Patterns for Tcl find*, on page 152.

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

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

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

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

You can crossprobe the objects selected by the find and expand commands, by right-clicking and choosing Select in Analyst column. The schematic views highlight the objects located by these commands. For other viewing operations, see *Viewing and Manipulating Collections with Tcl Commands*, on page 167.

2. To create a collection that is made up of other collections, do this:
   - Define the collections as described in the previous step. These collections must be defined before you can concatenate them or add them together in a new collection.
   - To concatenate collections or add to collections, type a name for the new collection in the Name column. Type the appropriate operator commands like c_union or c_diff in the Command column. See *Creating Collections using Tcl Commands*, on page 164 for a list of available commands and the *Reference Manual* for their syntax.

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

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

Copyright © 2013 Synopsys, Inc.

	Enabled	Delay Type	From	To	Through	Start/End	Cycles	Max Delay(ns)	Commen
1	☑	False	special_regs.status[7:0]						
2			i:decode.opcode_goto						
3			i:decode.opcode_call						
4			i:decode.opcode_retlw						
5			i:decode.skip						
6			i:decode.decodes[13:0]						
			i:prgmcntr...r_0_[10:0]						
			i:prgmcntr...r_1_[10:0]						
			i:prgmcntr...r_2_[10:0]						
			i:prgmcntr...r_3_[10:0]						
			i:prgmcntr...r_4_[10:0]						

Delay Paths

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

Example: Attribute Attached to a Collection

The following example shows the xc_area_group attribute applied to $find_reg, which results in all the registers in this collection being placed in the same region. Check the `srr` file, the netlist, and if you are using Synplify Premier Design Planner view to see that the attribute is honored.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	☑	instance	$find reg	xc_area_group	rr#cc#rr#cc	string	Specifies the region where instance should be placed	
2								
3								
4								
5								
6								

Attributes

Creating Collections using Tcl Commands

This section describes how to use the Tcl collection commands at the command line or in a script instead of entering them in the SCOPE window (*Creating and Using SCOPE Collections*, on page 162). There are differences in operation depending on where the collection commands are entered, and it is recommended that you use the SCOPE window, for the reasons described in *Comparison of Methods for Defining Collections*, on page 161.

For details of the syntax for the commands described here, refer to *Collections*, on page 480 in the *Reference Manual*.

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

   A collection can consist of individual objects, Tcl lists (which can consist of a single element), or other collections. You can embed the Tcl find and expand commands in the set command to locate objects for the collection (see *Using the Tcl Find Command to Define Collections*, on page 155 and *Specifying Search Patterns for Tcl find*, on page 152). The following example creates a collection called my_collection which consists of all the modules (views) found by the embedded find command:

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

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

To...	Use this command...
Add objects to a collection	c_union. See *Examples: c_union Command*, on page 166
Concatenate collections	c_union. See *Examples: c_union Command*, on page 166.
Isolate differences between collections	c_diff. See *Examples: c_diff Command*, on page 166.
Find common objects between collections	c_intersect. See *Examples: c_intersect Command*, on page 166.
Find objects that belong to just one collection	c_symdiff. See *Examples: c_symdiff Command*, on page 167.

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

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

Copyright © 2013 Synopsys, Inc.

Once you have created a collection, you can do various operations on the objects in the collection (see *Viewing and Manipulating Collections with Tcl Commands, on page 167*), but you cannot apply constraints to the collection.

Examples: c_union Command

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

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

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

This example concatenates collection1and collection2 and names the new collection combined_collection:

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

Examples: c_diff Command

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

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

You can also use the command to compare two collections:

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

Examples: c_intersect Command

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

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

Copyright © 2013 Synopsys, Inc.

You can also use the command to compare two collections:

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

Examples: c_symdiff Command

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

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

You can also use the command to compare two collections:

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

Examples: Names with Special Characters

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

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

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

Alternatively, use a backslash to escape the special character:

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

Viewing and Manipulating Collections with Tcl Commands

The following section describes various operations you can do on the collections you defined. For full details of the syntax, see *Collections*, on page 480 in the *Reference Manual*.

1. To view the objects in a collection, use one of the methods described in subsequent steps:
   – Select the collection in an HDL Analyst view (step 2).

Copyright © 2013 Synopsys, Inc.

- Print the collection without carriage returns or properties (step 3).
- Print the collection in columns (step 4).
- Print the collection in columns with properties (step 5).

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

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

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

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

   The c_list command prints the collection without carriage returns or properties. Use this command when you want to perform subsequent Tcl commands on the list. See *Example: c_list Command*, on page 170.

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

```
{i:EP0RxFifo.u_fifo.dataOut[0]}
{i:EP0RxFifo.u_fifo.dataOut[1]}
{i:EP0RxFifo.u_fifo.dataOut[2]}
{i:EP0RxFifo.u_fifo.dataOut[3]}
{i:EP0RxFifo.u_fifo.dataOut[4]}
{i:EP0RxFifo.u_fifo.dataOut[5]}
```

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

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

```
Object Name                        slack  view  clock
{i:EP0RxFifo.u_fifo.dataOut[0]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[1]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[2]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[3]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[4]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[5]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[6]} 0.3223 "FDE" clk
{i:EP0RxFifo.u_fifo.dataOut[7]} 0.3223 "FDE" clk
{i:EP0TxFifo.u_fifo.dataOut[0]} 0.1114 "FDE" clk
{i:EP0TxFifo.u_fifo.dataOut[1]} 0.1114 "FDE" clk
```

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

6. You can do a number of operations on a collection, as listed in the following table. For details of the syntax, see *Collections*, on page 480 in the *Reference Manual*.

To...	Do this...
Copy a collection	Create a new variable for the copy and copy the original collection to it with the set command. When you make changes to the original, it does not affect the copy, and vice versa. **set** *my_collection_copy* **$***my_collection*
List the objects in a collection	Use the c_print command to view the objects in a collection, and optionally their properties, in column format: "v:top" "v:block_a" "v:block_b" Alternatively, you can use the -print option to an operation command to list the objects.

Copyright © 2013 Synopsys, Inc.

To...	Do this...
Generate a Tcl list of the objects in a collection	Use the c_list command to view a collection or to convert a collection into a Tcl list. You can manipulate a Tcl list with standard Tcl commands. In addition, the Tcl collection commands work on Tcl lists. This is an example of c_list results: {"v:top" "v:block_a" "v:block_b"} Alternatively, you can use the -print option to an operation command to list the objects.

Example: c_list Command

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

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

The c_list command returns a list:

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

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

```
proc terminal_to_owner_instance {terminal_name terminal_type} {
   regsub -all $terminal_type$ $terminal_name {} suffix
   regsub -all {^t:} $suffix {i:} prefix
   return $prefix
   }

foreach get_component $get_components_list {
   append owner [terminal_to_owner_instance $get_component {.CE}]
" "
   }

puts "terminal owner is $owner"
```

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

```
terminal owner is i:EP0RxFifo.u_fifo.dataOut[0]
i:EP0RxFifo.u_fifo.dataOut[1] i:EP0RxFifo.u_fifo.dataOut[2]
```

Copyright © 2013 Synopsys, Inc.

Converting SDC to FDC

The sdc2fdc Tcl shell command translates legacy FPGA timing constraints to Synopsys FPGA timing constraints. From the Tcl command line in the synthesis tool, the sdc2fdc command scans the input SDC files and attempts to convert constraints for the implementation.

To run the sdc2fdc Tcl shell command:

1. Load your Project file.
2. From the Tcl command line, type:

   ```
   sdc2fdc
   ```

3. Check the constraint results directory for details about this translation.
4. The new constraints file is automatically updated for your project. Save the new settings.

   The constraint results directory is created at

   projectDir/FDC_constraints/*implName*

   This directory includes the following results files:

   – *topLevel*_translated.fdc – Contains the Synopsys FPGA design constraints (FPGA design constraints and the Synopsys standard timing constraints)

   – *topLevel*|*compilePoint*_translate.log – Contains details about the translation. Translation error messages explain issues and how to fix them. Any translation errors not addressed when you run synthesis appear in the SRR log file, but does not stop synthesis from running.

5. Open the FDC file resulting from translation in the FPGA SCOPE editor to check these constraints and make any changes to them.
6. Run the constraints checker.
7. Save this version of the FDC to run synthesis.

For information about the FDC file, see *FPGA Design Constraint (FDC) File*, on page 989.

Copyright © 2013 Synopsys, Inc.

Note: Since the basic Synplify product does not have a Tcl window, you must run sdc2fdc from a command shell in batch mode. The syntax is:

```
synplify -batch test.prj -tclcmd "sdc2fdc -batch"
```

For details about the translated files and troubleshooting guidelines, see *sdc2fdc Tcl Shell Command*, on page 988.

Using the SCOPE Editor (Legacy)

You can use the Legacy SCOPE editor for the SDC constraint files created before release version G-2012.09. However, it is recommended that you translate your SDC files to FDC files to enable the latest version of the SCOPE editor and to utilize the enhanced timing constraint handling in the tool. The latest version of the SCOPE editor automatically formats timing constraints using Synopsys Standard syntax (such as create_clock, and set_multicyle_path).

To do this, add your SDC constraint files to your project and run the following at the command line:

```
% sdc2fdc
```

This feature translates all SDC files in your project.

If you choose to do so, the following procedure shows you how to use the legacy SCOPE editor to create constraints for the constraint file (SDC).

1. Open an existing file for editing.
   - Make sure you have closed the SCOPE window, or you could overwrite previous constraints.
   - Double-click on an existing constraint file (`sdc`) in the project.
   - Select File->Open, set the Files of Type filter to Constraint Files (`sdc`) and open the file you want.
2. Enter the timing or design constraints you need.

Use SCOPE...	To Define...
Clocks	Clock frequencies define_clock. See *Defining Clocks*, on page 179 for additional information.
	Clock frequency other than the one implied by the signal on the clock pin syn_reference_clock (attribute). See *Defining Clocks*, on page 179 for additional information
	Clock domains with asymmetric duty cycles define_clock. See *Defining Clocks*, on page 179 for additional information

Copyright © 2013 Synopsys, Inc.

Use SCOPE...	To Define...
Clock to Clock	Edge-to-edge clock delays define_clock_delay. See *Defining Clocks*, on page 179 for additional information
Collections	Set constraints for a group of objects you have defined as a collection with the Tcl command.
Inputs/Outputs	Speed up paths feeding into a register define_reg_input_delay.
	Speed up paths coming from a register define_reg_output_delay.
Registers	Input delays from outside the FPGA define_input_delay. See *Defining Input and Output Constraints (Legacy)*, on page 186 for additional information
	Output delays from your FPGA define_output_delay. See *Defining Input and Output Constraints (Legacy)*, on page 186 for additional information
Delay Paths	Paths with multiple clock cycles define_multicycle_path. See *Defining Multicycle Paths*, on page 150 for additional information
	False paths (certain technologies) define_false_path. See *Defining False Paths (Legacy)*, on page 188 for additional information.
	Path delays define_path_delay. See *Defining From/To/Through Points for Timing Exceptions*, on page 146 for additional information
Attributes	Assign attributes for objects specifying their values
I/O Standards	Define an I/O standard for ports
Compile Points	Specify compile points for your design
Other	Enter newly-supported constraints for advanced users.

Entering and Editing SCOPE Constraints (Legacy)

Enter constraints directly in the SCOPE window. You can use the Initialize Constraint panel to enter default constraints, and then use the direct method to modify, add, or delete constraints.

The tool also lets you add constraints automatically. For information about auto constraints, see *Using Auto Constraints*, on page 739.

1. Click the appropriate tab at the bottom of the window to enter the kind of constraint you want to create:

To define...	Click...
Clock frequency for a clock signal output of clock divider logic A specific clock frequency that overrides the global frequency	Clocks
Edge-to-edge clock delay that overrides the automatically calculated delay.	Clock to Clock
Constraints for a group of objects you have defined as a collection with the Tcl command. For details, see *Creating and Using SCOPE Collections*, on page 162.	Collections
Input/output delays that model your FPGA input/output interface with the outside environment	Inputs/ Outputs
Delay constraints for paths feeding into/out of registers	Registers
Paths that require multiple clock cycles	Delay Paths
Paths to ignore for timing analysis (false paths)	Delay Paths
Maximum delay for paths	Delay Paths
Attributes, like syn_reference_clock, that were not entered in the source files	Attributes
I/O standards for any port in the I/O Standard panel of the SCOPE window.	I/O Standard
Compile points in a top-level constraint file. See *Synthesizing Compile Points*, on page 1109 for more information about compile points.	Compile Points
Place and route tool constraints Other constraints not used for synthesis, but which are passed to other tools. For example, multiple clock cycles from a register or input pin to a register or output pin	Other

Copyright © 2013 Synopsys, Inc.

The SCOPE window displays columns appropriate to the kind of constraint you picked. You can now enter constraints using the wizard, or work directly in the SCOPE window.

2. Save the file by clicking the Save icon and naming the file.

   The software creates a TCL constraint file (`sdc`). See *Working with Constraint Files*, on page 122 for information about the commands in this file.

3. To apply the constraints to your design, you must add the file to the project now or later.
   - Add it immediately by clicking Yes in the prompt box that opens after you save the constraint file.
   - Add it later, following the procedure for adding a file described in *Making Changes to a Project*, on page 216.

Specifying SCOPE Timing Constraints (Legacy)

You can define timing constraints in the SCOPE GUI, which automatically generates a Tcl constraints file, or manually with a text editor, as described in *Using a Text Editor for Constraint Files (Legacy)*, on page 122.

The SCOPE GUI is much easier to use, and you can define various timing constraints in it. For the equivalent Tcl syntax, see Chapter 17, *Batch Commands and Scripts* in the *Reference Manual*. See the following for different timing constraints:

- Entering Default Constraints, on page 177
- Setting Clock and Path Constraints, on page 177
- Defining Clocks, on page 179
- Defining Input and Output Constraints (Legacy), on page 186
- Specifying Standard I/O Pad Types, on page 140

To set constraints for timing exceptions like false paths and multicycle paths, see *Specifying Timing Exceptions*, on page 146.

For information about physical constraints, see *Setting Constraints for Physical Synthesis*, on page 320

Entering Default Constraints

To edit or set individual constraints, or to create constraints in the Other tab, work directly in the SCOPE window (*Setting Clock and Path Constraints*, on page 177). For auto constraints in Synplify Pro, see *Using Auto Constraints*, on page 739. To apply the constraints, add the file to the project according to the procedure described in *Making Changes to a Project*, on page 216. The constraints file has an `sdc` extension. See *Working with Constraint Files*, on page 122 for more information about constraint files.

Setting Clock and Path Constraints

The following table summarizes how to set different clock and path constraints from the SCOPE window. For information about setting compile point constraints or attributes, see *Synthesizing Compile Points*, on page 1109 for more information about compile points and *Specifying Attributes Using the SCOPE Editor*, on page 250. For information about setting default constraints, see *Entering Default Constraints*, on page 177.

To define...	Pane	Do this to set the constraint...
Clocks	Clock	Select the clock object (Clock). Specify a clock name (Clock Alias), if required. Type a frequency value (Frequency) or a period (Period). Change the default Duty Cycle or set Rise/Fall At, if needed. Change the default clock group, if needed Check the Enabled box. See *Defining Clocks*, on page 179 for information about clock attributes.
Virtual clocks	Clock	Set the clock constraints as described for clocks, above. Check the Virtual Clock box.
Route delay	Clock Inputs/ Outputs Registers	Specify the route delay in nanoseconds. Refer to *Defining Clocks*, on page 179, *Defining Input and Output Constraints (Legacy)*, on page 186 and the Register Delays section of this table details.
Edge-to-edge clock delay	Clock to Clock	Select the starting edge for the delay constraint (From Clock Edge). Select the ending edge for the constraint (To Clock Edge). Enter a delay value. Mark the Enabled check box.

Copyright © 2013 Synopsys, Inc.

To define...	Pane	Do this to set the constraint...
Input/output delays	Inputs/ Outputs	See *Defining Input and Output Constraints (Legacy)*, on page 186 for information about setting I/O constraints.
Register delays	Registers	Select the register (Register). Select the type of delay, input or output (Type). Type a delay value (Value). Check the Enabled box. If you do not meet timing goals after place-and-route, adjust the clock constraint as follows: • In the Route column for the constraint, specify the actual route delay (in nanoseconds), as obtained from the place-and-route results. Adding this constraint is equivalent to putting a register delay on that input register. • Resynthesize your design.
Maximum path delay	Delay Path	Select the Delay Type path of Max Delay. Select the port or register (From/Through). See *Defining From/To/Through Points for Timing Exceptions*, on page 146 for more information. Select another port or register if needed (To/Through). Set the delay value (Max Delay). Check the Enabled box.
Multi-cycle paths	Delay Paths	See *Defining Multicycle Paths*, on page 150.
False paths	Delay Paths Clock to Clock	See *Defining False Paths (Legacy)*, on page 188 for details.

To define...	Pane	Do this to set the constraint...
Global attributes	Attributes	Set Object Type to <global>. Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.
Attributes	Attributes	Do either of the following: • Select the type of object (Object Type). Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box. • Set the attribute (Attribute) and its value (Value). Select the object (Object). Check the Enabled box.
Other	Other	Type the TCL command for the constraint (Command). Enter the arguments for the command (Arguments). Check the Enabled box.

Defining Clocks

Clock frequency is the most important timing constraint, and must be set accurately. If you are planning to auto constrain your design (*Using Auto Constraints*, on page 739), do not define any clocks. The following procedures show you how to define clocks and set clock groups and other constraints that affect timing:

- Defining Clock Frequency, on page 179
- Constraining Clock Enable Paths, on page 183
- Defining Other Clock Requirements, on page 185

Defining Clock Frequency

This section shows you how to define clock frequency either through the GUI or in a constraint file. See *Defining Other Clock Requirements*, on page 185 for other clock constraints. If you want to use auto constraints, do not define your clocks.

1. Define a realistic global frequency for the entire design, either in the Project view or the Constraints tab of the Implementation Options dialog box.

   This target frequency applies to all clocks that do not have specified clock frequencies. If you do not specify any value, a default value of 1

Copyright © 2013 Synopsys, Inc.

MHz (or 1000 ns clock period) applies to all timing paths whenever the clock associated with both start and end points of the path is not specified. Each clock that uses the global frequency is assigned to its own clock group. See *Defining Other Clock Requirements*, on page 185 for more information about clock group settings.

The global frequency also applies to any purely combinatorial paths. The following figure shows how the software determines constraints for specified and unspecified start or end clocks on a path:

If clkA is...	And clkB is...	The effect for logic C is...
Undefined	Defined	The path is unconstrained unless you specify that clkB be constrained to the inferred clock domain for clkA
Defined	Undefined	The path is unconstrained unless you specify that clkA be constrained to the inferred clock domain for clkB.
Defined	Defined	For related clocks in the same clock group, the relationship between clocks is calculated; all other paths between the clocks are treated as false paths.
Undefined	Undefined	The path is unconstrained.

2. Define frequency for individual clocks on the Clocks tab of the SCOPE window (define_clock constraint).
   - Specify the frequency as either a frequency in the Frequency column (-freq Tcl option) or a time period in the Period column (-period Tcl option). When you enter a value in one column, the other is calculated automatically.
   - For asymmetrical clocks, specify values in the Rise At (-rise) and Fall At (-fall) columns. The software automatically calculates and fills out the Duty Cycle value.

The software infers all clocks, whether declared or undeclared, by tracing the clock pins of the flip-flops. However, it is recommended that you specify frequencies for all the clocks in your design. The defined frequency overrides the global frequency. Any undefined clocks default to the global frequency.

3. Define internal clock frequencies (clocks generated internally) on the SCOPE Clocks tab (define_clock constraint). Apply the constraint according to the source of the internal clock.

Source	Add SCOPE constraint/define_clock to...
Register	Register.
Instance, like a PLL or clock DLL	Instance. If the instance has more than one clock output, apply the clock constraints to each of the output nets, making sure to use the n: prefix (to signify a net) in the SCOPE table.
Combinatorial logic	Net. Make sure to use the n: prefix in the SCOPE interface.

4. For signals other than clocks, define frequencies with the syn_reference_clock attribute. You can add this attribute on the SCOPE Attributes tab, as follows:
   - Define a dummy clock on the Clocks tab (define_clock constraint).
   - Add the syn_reference_clock attribute (Attributes tab) to the affected registers to apply the clock. In the constraint file, you can use the Find command to find all registers enabled by a particular signal and then apply the attribute:

```
define_clock -virtual dummy -period 40.0
define_attribute {find -seq * -hier -filter @(enable == en40)}
   syn_reference_clock dummy
```

In earlier releases, limited clocking resources might have forced you to use an enable signal as a clocking signal, and use the syn_reference_clock attribute to define an enable frequency. However, because of changes in the reporting of clock start and end points, it is recommended that you use a multicycle path constraint instead for designs that use an enable signal and a global clock, and where paths need to take longer than one clock cycle. See *Constraining Clock Enable Paths*, on page 183 for a detailed explanation.

Copyright © 2013 Synopsys, Inc.

Note: This method is often used for designs that have an enable signal and a global clock, and where paths need to take longer than one clock cycle. The registers in the design are actually connected to the global clock, however, the tool treats the registers as having a virtual clock at the frequency of the enable signal.

Using this method to constrain paths for technologies with clock buffer delays requires careful analysis with the Timing Analysis Reports (STA). The virtual clock does not include clock buffer delays. However, non-virtual clocks that pass through clock buffers do include clock buffer delays. The register that generates the enable signal is on the non-virtual clock domain, whereas the registers connected to the enable signal are on the virtual clock domain. Timing analysis shows that the enable signal is on the path between the non-virtual and virtual clock domains. For the actual design, the enable signal is on a path in the non-virtual clock domain. Any paths between virtual and non-virtual clocks are reported with a clock buffer delay on the non-virtual clock. This may result in the critical path reporting negative slack.

In the following example, the path comes from a register on a non-virtual clock and goes to a register on a virtual clock.

```
Path information for path number 1:
   Requested Period:3.125
   - Setup time: 0.229
   = Required time: 2.896

   - Propagation time: 1.448
   - Clock delay at starting point: 1.857
   = Slack (critical: -0.409

   Number of logic level(s): 0
   Starting point:  SourceFlop / Q
   Ending point:  DestinationFlop / CE
   The start point is clocked by Non-VirtualClock [rising] on pin C
   The end point is clocked by VirtualClock [rising] on pin C

The path is reported with a negative slack of -0.49.
```

Timing analysis specifies a Clock delay at starting point that is the delay in the clock buffers of the non-virtual clock, but not a Clock delay at ending point. In the actual design, this delay exists at the end point. Since the clock end point is a virtual clock, the clock buffer delay creates a negative slack that does not exist in the actual design.

It is recommended that you use a multicycle path constraint instead to constrain all registers driven by the enable signal in the design.

5. For Altera PLLs and Xilinx DCMs and DLLs, define the clock at the primary inputs.
   - For Altera PLLs, you must define the input frequency, because the synthesis software does not use the input value you specified in the Mega wizard software. The synthesis tool assigns all the PLL outputs to the same clock group. It forward-annotates the PLL inputs.
   - If needed, use the Xilinx properties directly to define the DCMs and DLLs. The synthesis software assigns defined DCMs and DLLs to the same clock group, because it considers these clocks to be related. It forward-annotates the DLL/DCM inputs. The following shows some examples of the properties you can specify

DLLs	Phase shift and frequency multiplication properties like duty_cycle_correction and clkdv_divide
DCMs	DCM properties like clkfx_multiply and clkfx_divide

6. After synthesis, check the Performance Summary section of the log file for a list of all the defined and inferred clocks in the design.
7. If you do not meet timing goals after place-and-route, adjust the clock constraint as follows:
   - Open the SCOPE window with the clock constraint.
   - In the Route column for the constraint, specify the actual route delay (in nanoseconds), as obtained from the place-and-route results. Adding this constraint is equivalent to putting a register delay on all the input registers for that clock.
   - Resynthesize your design.

Constraining Clock Enable Paths

You might use an enable signal as a clocking signal if you have limited clocking resources. If the enable is slower than the clock, you can ensure more accuracy by defining the enable frequency separately, instead of slowing down the clock frequency. If you slow down the clock frequency, it affects all other registers driven by the clock, and can result in longer run times as the tool tries to optimize a non-critical path.

Copyright © 2013 Synopsys, Inc.

There are two ways to define clock enables:

- By setting a multicycle path constraint to constrain all flip-flops driven by the clock enable signal (see *Defining Multicycle Paths*, on page 150). This is the recommended method. You must use this method for the more recent Xilinx Virtex technologies.
- Using the syn_reference_clock attribute, as described in step 4 of *Defining Clock Frequency*, on page 179. Although this method was used in earlier releases, it is not recommended any more because of changes in the way the clock start and end points are reported. In particular, it is not recommended for the more recent Xilinx Virtex technologies, as critical paths could be reported with negative slack. An explanation of the clock start and end points reporting follows.

Clock Domains for Clock Enables Defined with syn_reference_clock

When you use the syn_reference_clock attribute to constrain an enable signal, you are telling the tool to treat the flip-flops as if they had a virtual clock at the frequency of the enable signal, when the flip-flops are actually connected to the global clock. This could result in critical paths being reported with negative slack.

The flip-flop that generates the enable signals is in the non-virtual clock domain.The flip-flops that are connected to the enable signal are in the virtual clock domain. The timing analyst considers the enable signal to be on a path that goes between a non-virtual clock domain and a virtual clock domain. In the actual circuit, the enable signal is on a path within a non-virtual clock domain. The timing analyst reports any paths between virtual and non-virtual clocks with a clock buffer delay on the non-virtual clock. This is why critical paths might be reported with negative slack.

If you use this method to constrain paths in a technology that includes clock buffer delays, you must carefully analyze the timing analysis reports. The virtual clock does not include clock buffer delays, but any non-virtual clock that passes through clock buffers will include clock buffer delays.

The following is an example report of a path from a clock enable, starting from a flip-flop on a non-virtual clock to a flip-flop on a virtual clock. The path is reported with a negative slack of -0.49.

```
Path information for path number 1:
   Requested Period: 3.125
   - Setup time: 0.229
   = Required time:2.896
```

```
    - Propagation time:  1.448
    - Clock delay at starting point:     1.857
    = Slack (critical) :  -0.409

    Number of logic level(s):     0
    Starting point:SourceFlop/ Q
    Ending point:DestinationFlop / CE

The start point is clocked by Non-VirtualClock [rising]on pin C

The end point is clocked by VirtualClock [rising] on pin C
```

This timing analysis report includes a Clock delay at starting point, but does not include Clock delay at ending point. The clock delay at the starting point is the delay in the clock buffers of the non-virtual clock. In the actual circuit, this delay would also be at the ending point and not affect the calculation of slack. However as the ending clock is a virtual clock, the clock buffer delay ends up creating a negative slack that does not exist in the actual circuit.

This report is a result of defining the clock enables with the syn_reference_clock attribute. This is why it is recommended that you use multicycle paths to constrain all the flip-flops driven by the enable signal.

Defining Other Clock Requirements

Besides clock frequency (described in *Defining Clock Frequency*, on page 179), you can also set other clock requirements, as follows:

- If you have limited clock resources, define clocks that do not need a clock buffer by attaching the syn_noclockbuf attribute to an individual port, or the entire module/architecture.
- Define the relationship between clocks by setting clock domains. By default, each clock is in a separate clock group named default_clkgroup<*n*> with a sequential number suffix.
  - On the SCOPE Clocks tab, group related clocks by putting them into the same clock group. Use the Clock Group field to assign all related clocks to the same clock group.
  - Make sure that unrelated clocks are in different clock groups. If you do not, the software calculates timing paths between unrelated clocks in the same clock group, instead of treating them as false paths.
  - Input and output ports that belong to the System clock domain are considered a part of every clock group and will be timed. See *Defining*

Copyright © 2013 Synopsys, Inc.

Input and Output Constraints (Legacy), on page 186 for more information.

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

- Define all gated clocks with the define_clock constraint.

  Avoid using gated clocks to eliminate clock skew. If possible, move the logic to the data pin instead of using gated clocks. If you do use gated clocks, you must define them explicitly, because the software does not propagate the frequency of clock ports to gated clocks.

  To define a gated clock, attach the define_clock constraint to the clock source, as described above for internal clocks. To attach the constraint to a keepbuf (a keepbuf is a placeholder instance for clocks generated from combinatorial logic), do the following:

  – Attach the syn_keep attribute to the gated clock to ensure that it retains the same name through changes to the RTL code.

  – Attach the define_clock constraint to the net or pin connected to the keepbuf instance generated for the gated clock.

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

- After synthesis, check the Performance Summary section of the log file for a list of all the defined and inferred clocks in the design.

Defining Input and Output Constraints (Legacy)

In addition to setting I/O delays in the SCOPE window as described in *Setting Clock and Path Constraints*, on page 177, you can also set the Use clock period for unconstrained IO option.

- Open the SCOPE window, click Inputs/Outputs, and select the port (Port). You can set the constraint for

  – All inputs and outputs (globally in the top-level netlist)

  – For a whole bus

  – For single bits

  You can specify multiple constraints for the same port. The software applies all the constraints; the tightest constraint determines the worst

slack. If there are multiple constraints from different levels, the most specific overrides the more global. For example, if there are two bit constraints and two port constraints, the two bit constraints override the two port constraints for that bit. The other bits get the two port constraints.

- Specify the constraint value in the SCOPE window:
  - Select the type of delay: input or output (Type).
  - Type a delay value (Value).
  - Check the Enabled box, and save the constraint file in the project.

  Make sure to specify explicit constraints for each I/O path you want to constrain.
- To determine how the I/O constraints are used during synthesis, do the following:
  - Select Project->Implementation Options, and click Constraints.
  - To use only the explicitly defined constraints disable Use clock period for unconstrained IO.
  - To synthesize with all the constraints, using the clock period for all I/O paths that do not have an explicit constraint enable Use clock period for unconstrained IO.
  - Synthesize the design. When you forward-annotate the constraints, the constraints used for synthesis are forward-annotated for place-and-route.
- Input or output ports with explicitly defined constraints, but without a reference clock (-ref option) are included in the System clock domain and are considered to belong to every defined or inferred clock group.
- If you do not meet timing goals after place-and-route and you need to adjust the input constraints; do the following:
  - Open the SCOPE window with the input constraint.
  - In the Route column for the input constraint, specify the actual route delay in nanoseconds, as obtained from the place-and-route results. Adding this constraint is equivalent to putting a register delay on the input register.
  - Resynthesize your design.

Copyright © 2013 Synopsys, Inc.

Defining False Paths (Legacy)

You define false paths by setting constraints explicitly on the Delay Paths tab or implicitly on the Clock and Clock to Clock tabs. See *Defining From/To/Through Points for Timing Exceptions*, on page 146 for object naming and specifying through points.

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

  The software treats this as an explicit false constraint and assigns it the highest priority. Any other constraints on this path are ignored.
- To define a false path between two clocks, select the SCOPE Clocks tab, and assign the clocks to different clock groups:

  The software implicitly assumes a false path between clocks in different clock groups. This false path constraint can be overridden by a maximum path delay constraint, or with an explicit constraint.
- To define a false path between two clock edges, select the SCOPE Clock to Clock tab, and do the following:
  - Specify one clock as the starting clock edge (From Clock Edge).
  - Specify the other clock as the ending clock edge (To Clock Edge).
  - Click in the Delay column, and select false.
  - Mark the Enabled check box.

  Use this technique to specify a false path between any two clocks, regardless of clock groups. This constraint can be overridden by a maximum delay constraint on the same path
- To override an implicit false path between any two clocks described previously, set an explicit constraint between the clocks by selecting the SCOPE Clock to Clock tab, and doing the following:
  - Specify the starting (From Clock Edge) and ending clock edges (To Clock Edge).
  - Specify a value in the Delay column.

Copyright © 2013 Synopsys, Inc.

  - Mark the Enabled check box.

  The software treats this as an explicit constraint. You can use this method to constrain a path between any two clocks, regardless of whether they belong to the same clock group.

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

Copyright © 2013 Synopsys, Inc.

Translating XDC Constraints to FDC

Xilinx

Vivado uses the XDC format for constraints. If you want to use XDC constraints for MIG IP in the synthesis run, you must convert them to the synthesis FDC format. To do this, you use the xdc2fdc command, as described below. You can use the command to translate a single xdc file or all the xdc files in a project.

The following topics describe the details:

- Converting Constraints with the xdc2fdc Command, next
- Debugging xdc2fdc Conversions, on page 191

Converting Constraints with the xdc2fdc Command

The following procedure shows you how to convert Xilinx XDC constraints to FDC constraints that you can use to guide synthesis:

1. To translate a single xdc file, type the following from the Tcl command line in the synthesis tool, specifying the xdc file to be converted and the fdc file to which the translated constraints should be written:

   xdc2fdc -xdc_file *file* **-fdc_file** *file*

   The xdc2fdc command runs on the Xilinx file specified with the -xdc_file option and translates it into FDC constraints which are written to the file specified with the -fdc_file option. See *xdc2fdc Command*, on page 1020 of the *Reference Manual* for a detailed explanation of the syntax and examples.

   The tool prints a conversion report to the Tcl window and to the log file specified in the report.

2. To translate all the xdc files in a project, do the following:
   - Open a synthesis project and add the xdc files you want to convert to the project. This command is intended for use with Xilinx MIG IP generated from the Vivado IP catalog. While it is expected that the command may be successfully used to translate other XDC files, currently it has only been tested with XDC files for MIG IP.
   - You must have the project open.

– Type the following command in the Tcl window of the synthesis tool:

xdc2fdc -project

This command translates all the xdc files in the current project, and then adds the translated fdc files to the current project. It marks the original xdc file for place-and-route only, and uses the generated fdc files for synthesis.

3. Check your converted constraints.
   – Check the log file for constraints that were not translated, or for incorrect conversions. You can search for keywords such as constraint or .fdc.
   – Manually modify any untranslated or incorrect constraints. See *Debugging xdc2fdc Conversions*, on page 191 for more information.
4. Synthesize your design.

   The tool forward-annotates fdc constraints to place-and-route as an xdc file. If you want to use the original xdc file for place-and-route, disable the xdc file generated after synthesis.

Debugging xdc2fdc Conversions

The xdc2fdc command reports any xdc constraints it fails to translate. Some of these situations are described here:

- Inability to translate a constraint

  The xdc2fdc command may not be able to translate every constraint from the input XDC file. Manually modify any untranslated constraints. The command flags untranslated constraints with messages like this one:

  Untranslated constraint at line 124 of XDC file.

- Possible errors in using original xdc file for place and route

  If you use the converted fdc file for synthesis, but the original xdc file for place-and-route, be aware that the synthesis tool might rename a constrained element during synthesis. The translation process accounts for this in the fdc file, but the original xdc file remains untouched. Make sure you manually edit the element in the original xdc file.

  The command generates a warning like the following when it detects such an issue:

Copyright © 2013 Synopsys, Inc.

Warning: The following constraint has an element with a _reg suffix in its name. Synthesis may rename this element, and this can cause the constraint not to work if you use the original Xilinx XDC file for P&R. It is suggested that you edit the XDC file and remove _reg from the element name to avoid this problem.

See *xdc2fdc Command*, on page 1020 in the *Reference Manual* for examples.

- Incorrect translations

  Manually modify the incorrectly translated constraints in the fdc file.

  The following message is generated when the xdc2fdc command tries to remove an existing FDC file that matches the translated FDC, before a new file is added to the project. This only happens using the -project option. You can safely ignore this message.

  No files found matching:
  C:\Projects\project_mig\project_mig.srcs\sources_1\ip\mig_7series_v1_8_a_0\synplify_project\FDC_constraints\rev_1\mig_7series_v1_8_a_0_xdc_translated.fdc

  The following message is generated if the XDC file is removed from the project first, before adding it back for P&R only. You can safely ignore this message.

  Removed file:
  C:\Projects\project_mig\project_mig.srcs\sources_1\ip\mig_7series_v1_8_a_0\mig_7series_v1_8_a_0\user_design\constraints\mig_7series_v1_8_a_0.xdc

Translating Altera QSF Constraints

If you have an Altera Quartus Settings File (QSF) with I/O constraints, you can use the legacy qsf2sdc translator and the following procedure to translate these constraints to the sdc format and use the translated constraints to drive synthesis. For example, you can translate the I/O constraints and use it for physical synthesis.

If you want to convert an entire Quartus project to a synthesis project, use the `qsf2syn` utility and the procedure described in *Importing Projects from Quartus*, on page 1033.

1. Run the qsf2sdc utility.
   - Make sure the input QSF file has a `qsf` extension.
   - From the command line, run the translator on the QSF file. The translator is in the bin directory: *install_dir*/bin/qsf2sdc.exe. Use the following syntax:

     ```
     installDir/bin/qsf2sdc -iqsf constraintsFile.qsf
            -osdc constraintsFile.sdc
            [-oqsf residualConstraintsFile>.qsf] [-all]
            [-silent]
     ```

     The translator generates a constraint file in the sdc format, which contains the I/O constraints from the `qsf` file that are relevant to synthesis. It ignores the other back-end constraints in the file. See *Altera qsf2sdc Utility*, on page 994 in the *Reference Manual* for details of the syntax and a list of supported pin location and I/O constraints.

2. After translating the constraints, edit the new `sdc` file.
   - Visually inspect the translated file.

     The original `qsf` commands are written as comments in the new `sdc` file so that you can validate the translated constraints. Constraints which were successfully translated are specified as Supported. However, constraints which were unsuccessfully translated are specified as Unsupported. Use the -silent option to suppress all the #Supported and #Unsupported messages in the `sdc` file.
   - Manually edit the `sdc` file to complete the translation of constraints, as necessary.

Copyright © 2013 Synopsys, Inc.

- Optionally, use the -all option to convert any instances with location assignments. By default, only pin location assignments and IO standards are automatically converted.

3. To run physical synthesis, create a single `sdc` file that contains all of the constraints.
   - Include timing constraints created previously into the `sdc` file containing the translated physical constraints. Make sure that all of the following types of constraints are combined into the `sdc` file

     Timing Constraints:
     Clock
     Clock-to-clock
     IO delays
     IO standard, drive, slew and pull-up/pull-down
     Multi-cycle and false paths
     Max-delay paths
     DCM parameters
   - Include any physical constraints you have, like syn_loc constraints on I/O pins and pad types.
   - Include any synthesis attributes from logic synthesis, such as syn_ramstyle, into the `sdc` file.
4. Edit the original `qsf` file.
   - Remove all translated constraints from the original `qsf` file.
   - If there are any untranslated QSF commands left in the file, add the `qsf` file to your project. The file must have the same base name as the `vqm` netlist so that the Altera P&R tool can source the file.
5. Run a constraint check by selecting Run->Constraint Check.

   This command generates a report that checks the syntax and applicability of the timing constraints in any `sdc` files for your project. The report is written to the *projectName*_cck.rpt file.
6. Add the generated `sdc` file to the project, and use it to drive synthesis.

Specifying Xilinx Constraints (Legacy)

For Xilinx designs, you can import Xilinx constraints from a `ucf` file in addition to specifying constraints within the synthesis tool. In the output files, the synthesis tool separates the timing constraints from the physical constraints. Timing constraints are written to the synplicity.ucf file and physical constraints to the *design*.ncf file, as shown in this figure:

1. To specify user constraints, double-click on an existing `sdc` constraint file.

   See *Entering and Editing SCOPE Constraints (Legacy)*, on page 175 for details on how to specify constraints.

2. To use constraints from a Xilinx UCF file, use the procedures described in *Converting and Using Xilinx UCF Constraints*, on page 200.

3. Synthesize the design.

   The synthesis tool writes out the timing constraints and physical constraints into separate files:

synplicity.ucf	Contains all timing constraints, whether user-specified or translated from a `ucf` file
<design>.ncf	Contains all physical constraints

Copyright © 2013 Synopsys, Inc.

4. Use synplicity.ucf and *design*.ncf as input to the Xilinx place-and-route tool. Update scripts or older par_opt files if needed to ensure that these files are used to drive place-and-route.

Setting Clock Priority in Xilinx Designs (Legacy)

You use the syn_clock_priority attribute to set clock priority and resolve clock conflicts in Xilinx designs. You can use this attribute effectively to override DCM clocks, and to resolve paths with timing conflicts.

You set the attribute by specifying a positive value for the clock, with 1 being the highest priority.

```
define_attribute {n:u_fx_clkrstgen.clk_100_dcm}
{syn_clock_priority} {1}
```

For details about the attribute and how it is forward-annotated, refer to *syn_clock_priority*, on page 1241 of the *Reference Manual*.

The following sections discuss how to use the attribute to override DCM clocks and resolve paths with multiple timing specifications:

- Overriding DCM Clocks, on page 196
- Specifying Clock Priority for BUFG and BUFGMUX Elements, on page 198
- Defining a False Path for Xilinx BUFGMUX_CTRL, on page 198

Overriding DCM Clocks

When you override DCM clocks with declared clocks, ISE does not honor this. Instead of manually editing the UCF file, you can use the syn_clock_priority attribute to assign a priority to a particular clock. The tool forward-annotates the clock priority as a TIMESPEC or PERIOD statement in the `ucf` file. For details, see *Forward-annotation of Clock Priority to UCF*, on page 1242 of the *Reference Manual* for details.

1. If you want to override a DCM or other derived clock with a user-defined clock, set syn_clock_priority when you are prompted to do so.

   The tool automatically prompts you to do this.

   If the override clock is in a different group from the DCM source clock, the tool adds TIGs to the UCF between (from/to and to/from) the DCM

source clock net and the DCM output net where the overriding clock is defined.

2. If you also set a priority for the override clock at the DCM output, make sure that the priority of the DCM input clock is lower than the priority on the override clock.

   You must do this because DCM derived clocks inherit the priority of the DCM base clock. If you do not ensure that the DCM input clock has a lower priority, the tool will forward-annotate it instead of the override clock.

3. To correctly specify the syn_clock_priority attribute to a derived clock output of a DCM, apply it immediately at the output where the derived clock is created.

   To set syn_clock_priority for the DCM CLKFX output in the following figure, you must specify the following syntax in the `sdc` file:

   ```
   define_attribute {n:dcm_module_b.clk0fx} {syn_clock_priority} {1}
   ```

4. For DCMs with dual output clock pins, specify which clock is to be forward-annotated by setting the clock priority, unless the input clock select pin of the DCM is tied high or low and indicates an explicit choice.

   For DCMs with dual input clock pins, only one of the clocks is propagated through the DCM to create the derived clocks in ISE. This is true even if the two clocks on these pins are unrelated. So, unless the input

Copyright © 2013 Synopsys, Inc.

clock select pin explicitly indicates the choice. If the input clock select pin is tied high or low, you do not need to set clock priority, because there is no clock conflict.

Specifying Clock Priority for BUFG and BUFGMUX Elements

The FPGA synthesis tools allow multiple clocks to propagate along a single net, through unate logic or through a mux. The Xilinx ISE does not do this, so you must indicate clock priority in case of conflict with the syn_clock_priority attribute.

1. If you want to specify clock priority on a path with BUFGMUX components, set syn_clock_priority.

   In some cases, the tool generates a warning and prompts you to do this when it detects multiple timing on a path.

2. If you set a priority for both the nets driving the data inputs of a BUFGMUX, make sure that one has a higher priority than the other.

   The tool only propagates the clock with the highest priority values through the mux. If you set the same value on both mux inputs, you see a warning message in the log file.

3. To specify clock priority for a BUFG on a derived clock, specify the syn_clock_priority directly on the BUFG, as shown in this example:

   ```
   define_clock {i:CLK_BUF0FB} -name {i:CLK_BUF0FB} -freq 100
      -clockgroup clk0_derived_clock

   define_clock {i:CLK_BUF0} -name {i:CLK_BUF0} -freq 30
      -clockgroup clk0fx_derived_clock

   define_attribute {i:CLK_BUF0} {syn_clock_priority} {1}
   define_attribute {i:CLK_BUF0FB} {syn_clock_priority} {1}
   ```

   You cannot set this attribute on any other instances except BUFGs.

Defining a False Path for Xilinx BUFGMUX_CTRL

BUFGMUX_CTRL is a wrapper around a Xilinx BUFGCTRL primitive. It consists of a clock buffer with two clock inputs, one clock output, and a select line. The tool does not consider the BUFGMUX_CTRL a sequential element and does not perform timing analysis on this path.

Copyright © 2013 Synopsys, Inc.

If you want to define a false path constraint for BUFGMUX_CTRL and forward-annotate it, do the following:

1. Specify TIG (timing ignore) constraints on both the S0 and S1 pins of the BUFGCTRL beneath the BUFGMUX_CTRL, as in this example:

   ```
   PIN "BUFGMUX_CTRL_inst/BUFGCTRL.S0" TIG;
   PIN "BUFGMUX_CTRL_inst/BUFGCTRL.S1" TIG;
   ```

   You must apply the constraint to the BUFGCTRL pins, not the wrapper. Although you can set the constraint on the clock enable pins, it is recommended that you set it on the select pins, because this allows you to switch between the clock inputs without any glitches in setup/hold times.

2. Save the constraints in a separate `ucf` file.

3. Add the file to the project.

4. Synthesize as usual.

   The tool forward-annotates the constraints you specified to the place-and-route tool.

Copyright © 2013 Synopsys, Inc.

Converting and Using Xilinx UCF Constraints

As you iterate through the flow, you might want to use Xilinx UCF constraints to guide synthesis. To do this, you must translate the UCF constraints into SDC constraints that the synthesis tools can use. This is primarily used to move a Synplify Pro project to the Synplify Premier synthesis tool to run physical synthesis by converting the project and the UCF constraints.

The following procedures show you how to use the GUI commands to run the underlying ucf2sdc utility and convert the UCF constraints from a logic synthesis design into a physical synthesis design, and forward-annotate them for place-and-route. The first procedure describes how to deal with UCF constraints if you do not want to create a project.

- Converting UCF Constraints Without Creating a Project, on page 200
- Using Xilinx UCF Constraints in a Logic Synthesis Design, on page 201
- Using Xilinx UCF Constraints in a Physical Synthesis Design, on page 203
- Support for UCF Conversion, on page 206

These procedures do not describe how to translate UCF constraints from a Xilinx project into a synthesis project. For that information, see *Converting Xilinx Projects with ise2syn*, on page 1051.

Converting UCF Constraints Without Creating a Project

If you want to use UCF constraints but do not want to create a new project to do so (*Using Xilinx UCF Constraints in a Logic Synthesis Design*, on page 201 and *Using Xilinx UCF Constraints in a Physical Synthesis Design*, on page 203), you must translate the UCF constraints manually. If you just add the UCF file to the project, the UCF constraints will be passed to the Xilinx tool, but the place-and-route tool might not honor them because object names might not match after synthesis. To ensure that the UCF constraints are used, do the following:

1. Manually convert the UCF constraints to SDC constraints.
2. Add the `sdc` file to the project and run synthesis.
3. Run place-and-route.

The synthesis tool correctly forward-annotates the sdc constraints that you converted in step 1. The Xilinx tool runs place-and-route tool using the forward-annotated constraints.

Using Xilinx UCF Constraints in a Logic Synthesis Design

You can run logic synthesis in the Synplify Pro tool or in the Synplify Premier tool in logic synthesis mode. The following procedure shows you how to use Xilinx UCF constraints for a logic synthesis run with either of the tools.

1. Start with the Xilinx constraint files to be translated.
   - You can use the following kinds of files:

UCF	Top-level constraint file, with corresponding EDIF file (`edf`)
NCF	Block-level constraint file, with corresponding EDIF file (`edn`, `edf`, `ngc`, or `ngo`)
XCF	Block-level constraint file, with corresponding EDIF file (`ngc` or `ngo`)

     These files must refer to design objects in the mapped synthesis tool database so as to be consistent with subsequent synthesis runs. If you use a UCF file that refers to XST design objects, naming might be inconsistent. You can have multiple constraint files, one for the top-level, and others for blocks. See *Supported Input Files for UCF Conversion*, on page 206 for details about the input files, and *Xilinx ucf2sdc Utility*, on page 1004 in the *Reference Manual* for information about the supported constraints.
   - Add all Xilinx constraint files to be converted to the logic synthesis project.
   - Add the corresponding netlist files to the project, along with the constraint file.
2. Do an initial synthesis run.
   - Set up a P&R implementation.
   - Synthesize the design and run P&R.
   - Check the log files for any constraint-related warnings and fix them before proceeding.

Copyright © 2013 Synopsys, Inc.

3. Select Project->Convert Vendor Constraints to open the UCF to SDC Conversion dialog box.

4. Specify the translation options:
   - Specify a name for the new project in Project Name.
   - Set a location for the new project in Project Location.
   - In the Constraint Files section, enable the files you want to use. This section lists the files you added to the project in step 2. If you do not have corresponding EDIF files for the constraint files you enable, you see warning messages in the box at the bottom of the dialog box.
   - Enable Run Constraints Checker after Conversion and Invoke Report File.
   - Click the Convert button in the upper right.

     The tool uses information from the project `srd` file and translates the constraints in the input files, using a separate process for the top

level and for each block. It then creates a new project. Note that it does not delete the original project or files, but creates a new one. See *Generated Files after UCF Conversion*, on page 207 for names and descriptions of the files generated after conversion.

Finally, it runs the constraints checker and reports any Xilinx constraints that cannot be translated. See *Support for UCF Conversion*, on page 206 for information about supported and unsupported constraints.

– Check the ucf2sdc.log file for any errors or warnings.

5. To use the generated `sdc` file to drive synthesis for the new project, do the following:
   – Open the `sdc` file and check it. Edit it if necessary. You can also rename this file.
   – Make sure the file is added to the project.
   – Run logic synthesis by clicking Run.
6. After logic synthesis, you can do either or both of the following:
   – Use the newly-generated project and the `sdc` files with translated constraints for physical synthesis. See *Using Xilinx UCF Constraints in a Physical Synthesis Design*, on page 203 for more information.
   – Use the synplicity.ucf and unsupported.ucf files for Xilinx P&R. You can use the ucf2sdc.log file and the unsupported.ucf file to manually translate any remaining constraints.

Using Xilinx UCF Constraints in a Physical Synthesis Design

When you use UCF constraints in a physical synthesis design, it is assumed that you did a baseline logic synthesis run, and that you have certain files for the design. The following procedure shows you how to translate and use UCF constraints in a physical synthesis design with the Synplify Premier software.

1. Do an initial logic synthesis and P&R run using either Synplify Pro, or the Synplify Premier tool in logic synthesis mode. See step 1 of *Using Xilinx UCF Constraints in a Logic Synthesis Design*, on page 201.

   For Synplify Premier designs, the conversion of UCF constraints requires the `srd`, `prj`, and `ucf` files.

2. Do not enable physical synthesis.

Copyright © 2013 Synopsys, Inc.

If you enable physical synthesis, you get an error message. You can only convert the project when you are in logic synthesis mode.

3. Select Project->Convert Vendor Constraints to open the UCF to SDC Conversion dialog box.

4. Do the following:
   - Specify a name for the new project in Project Name.
   - Set a location for the new project in Project Location.
   - In the Constraint Files section, check that the files you want to translate are enabled. The list should include the files you added in step 2. If you do not have corresponding EDIF files for the constraint files you enable, you see warning messages in the box at the bottom of the dialog box.
   - Enable Run Constraints Checker after Conversion and Invoke Report File.

– Click the Convert button in the upper right.

The tool uses information from the project `srd` file, the `prj` file, and the `ucf` files. It translates the constraints in the Xilinx input constraint files and creates a new project. It does not overwrite or delete the files from the input project. Finally, it runs the constraints checker and reports any Xilinx constraints that cannot be translated. See *Supported UCF Constraints*, on page 208 for a list of supported and unsupported constraints.

It generates the following output constraint files:

*ucfFile*_conv.sdc	Translated UCF constraints. Could have multiple files.
unsuppported.ucf	Unsupported input Xilinx constraints, in the `ucf` format for forward-annotation. This includes any physical constraints that are not used by the synthesis tools.

5. To use the generated `sdc` file to drive further synthesis, do the following:
   – Open the ucf2sdc.log and check each `ucf` translation section in this file for errors or warnings.
   – Open the `sdc` file and check it. Edit it if necessary.
   – Make sure the file is added to the project.
   – Run physical synthesis.
6. Use the synplicity.ucf and `ncf` files for Xilinx P&R.

   You can check the ucf2sdc.log file for messages and manually translate any untranslated constraints in the synplify.ucf file before running P&R.

Copyright © 2013 Synopsys, Inc.

Support for UCF Conversion

For procedures on converting UCF constraints, see the methods listed *Converting and Using Xilinx UCF Constraints*, on page 200. The following describe what the software supports when translating UCF constraints to SDC.

- Supported Input Files for UCF Conversion, on page 206
- Generated Files after UCF Conversion, on page 207
- Supported UCF Constraints, on page 208

Supported Input Files for UCF Conversion

The synthesis software can translate Xilinx constraints from UCF, NCF, and XCF files with the Project->Convert Vendor Constraints command. The UCF file is for the top-level design, and the XCF and NCF files are for blocks. The following table lists support criteria for each of these formats:

UCF	• You can only have UCF files for the top-level project. • Paths referring to elements must start at the top level. • The UCF file must be one written for the Synopsys FPGA synthesis netlist. If it is an XST netlist, object names may not match. • The Convert Vendor Constraints command does not convert constraints if the `ucf` file was generated by the Synopsys FPGA tools. It ignores everything after the following comment line: `# Constraints generated by Synplify Pro maprc, Build` *number* The tool notifies you that it is ignoring these comments, and puts the unconverted constraints in the *design*_unsupported.ucf file. If you want to convert constraints from a Synopsys `ucf` file, delete the comment line.
NCF	• You can only use block-level NCF files. • A project can have multiple NCF files. • Each NCF file must have a corresponding `edn`, `edf`, `ngc`, or `ngo` file with the same name.
XCF	• You can only use block-level XCF files. • A project can have multiple XCF files. • Each XCF file must have a corresponding `ngc` or `ngo` file with the same name.

Generated Files after UCF Conversion

The tool creates these files after UCF conversion:

ucf2sdc.log	Log file that contains messages after ucf conversion completes.
*prjFile*_conv.prj	The default name for the new project that was generated.
*ucfFile*_conv.sdc	Contains converted Xilinx constraints for logic synthesis. The tool generates a corresponding `sdc` file for each input `ucf` file. The name for this file is derived from the input UCF file name.
synplify.ucf	After logic synthesis, this one file contains all the supported input Xilinx constraints in the `ucf` format. Unsupported constraints are in a separate file. After physical synthesis, this one file contains both supported and unsupported constraints in the `ucf` format.
unsupported.ucf	File that contains all the unsupported Xilinx constraints in the `ucf` format after logic synthesis. This can include any physical constraints not used for synthesis.

Copyright © 2013 Synopsys, Inc.

The next figure shows how the project-level input files are handled in a post-translation synthesis run:

Supported UCF Constraints

The UCF converter supports the following types of constraints:

	FF	RAM	ROM	DSP	Net	Inst	View	Collection	Port	Pin
PERIOD					Yes			Yes	Yes	Yes
FROM/TO	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
TIG	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
OFFSET	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
TNM TNM_NET TIMEGRP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
LOC	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
IO PROPS					Yes	Yes		Yes	Yes	
General PROPs	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Unsupported UCF Constraints

Currently, the UCF converter does not handle the following:

- Back-annotated netlists from the physical synthesis flow.
- Case-sensitive matching on instance and net names. For example: aBc.
- Nets driven by LUTs, except for nets that source OPADs.
- Collections that include inferred RAMs or DSPs. The tool cannot guarantee that inferred components match.
- The MAXDELAY constraint.

The UCF converter does not currently convert the following keywords:

Unsupported Keywords	Description
BRAMS_PORT[A/B]	Predefined keyword
INPUT_JITTER, PRIORITY, DATAPATHONLY	TIMESPEC constraint
RISING, FALLING	TIMEGRP constraint
CLOSED, OPEN	AREA_GROUP constraint
HIGH, LOW, VALID	OFFSET constraint

Copyright © 2013 Synopsys, Inc.

CHAPTER 5

Setting up a Logic Synthesis Project

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

- Setting Up Project Files, on page 212
- Managing Project File Hierarchy, on page 220
- Setting Up Implementations, on page 226
- Setting Logic Synthesis Implementation Options, on page 228
- Specifying Attributes and Directives, on page 245
- Working with Hierarchical Projects, on page 254
- Searching Files, on page 284
- Archiving Files and Projects, on page 287

To set up a physical synthesis project, you follow the steps for setting up a logic synthesis project, and then additionally follow the procedures described in Chapter 6, *Setting up a Physical Synthesis Project*.

Copyright © 2013 Synopsys, Inc.

Setting Up Project Files

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

- Creating a Project File, on page 212
- Opening an Existing Project File, on page 215
- Making Changes to a Project, on page 216
- Setting Project View Display Preferences, on page 217
- Updating Verilog Include Paths in Older Project Files, on page 219

For a specific example on setting up a project file, refer to the tutorial for the tool you are using.

For information about working with hierarchical projects, refer to *Working with Hierarchical Projects*, on page 254.

Creating a Project File

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

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

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

2. Add the source files to the project.
   - Make sure the Look in field at the top of the form points to the right directory. The files are listed in the box. If you do not see the files, check that the Files of Type field is set to display the correct file type. If you have mixed input files, follow the procedure described in *Using Mixed Language Source Files*, on page 117.

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

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

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

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

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

     For information about setting Verilog and VHDL file options, see *Setting Verilog and VHDL Options*, on page 239. You can also set these file options later, before running synthesis.

     For additional vendor-specific information about using vendor macro libraries and black boxes, see *Optimizing Microsemi Designs*, on page 789, *Optimizing Altera Designs*, on page 760, *Optimizing Lattice Designs*, on page 772, and *Optimizing Xilinx Designs*, on page 794.

- For generic technology components, you can either add the technology-independent Verilog library supplied with the software (*install_dir*/lib/generic_ technology/gtech.v) to your design, or add your own generic component library. Do not use both together as there may be conflicts.

4. Check file order in the Project view. File order is especially important for VHDL files.
   - For VHDL files, you can automatically order the files by selecting Run->Arrange VHDL Files. Alternatively, manually move the files in the Project view. Package files must be first on the list because they are compiled before they are used. If you have design blocks spread over many files, make sure you have the following file order: the file containing the entity must be first, followed by the architecture file, and finally the file with the configuration.
   - In the Project view, check that the last file in the Project view is the top-level source file. Alternatively, you can specify the top-level file when you set the device options.
5. Select File->Save, type a name for the project, and click Save. The Project window reflects your changes.
6. To close a project file, select the Close Project button or File->Close Project.

Opening an Existing Project File

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

1. If the project you want to open is one you worked on recently, you can select it directly: File->Recent Projects-> *projectName*.

Copyright © 2013 Synopsys, Inc.

2. Use one of the following methods to open any project file:

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

The project opens in the Project window.

Making Changes to a Project

Typically, you add, delete, or replace files.

1. To add source or constraint files to a project, select the Add Files button or Project->Add Source File to open the Select Files to Add to Project dialog box. See *Creating a Project File*, on page 212 for details.

2. To delete a file from a project, click the file in the Project window, and press the Delete key.

3. To replace a file in a project,
   - Select the file you want to change in the Project window.
   - Click the Change File button, or select Project->Change File.
   - In the Source File dialog box that opens, set Look In to the directory where the new file is located. The new file must be of the same type as the file you want to replace.
   - If you do not see your file listed, select the type of file you need from the Files of Type field.
   - Double-click the file. The new file replaces the old one in the project list.

Copyright © 2013 Synopsys, Inc.

4. To specify how project files are saved in the project, right click on a file in the Project view and select File Options. Set the Save File option to either Relative to Project or Absolute Path.

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

Setting Project View Display Preferences

You can customize the organization and display of project files.

1. Select Options->Project View Options.

   The Project View Options form opens. Available options vary, depending on the tool. The Synplify Premier and Synplify Pro options are the same.

Synplify Options

Synplify Pro and Synplify Premier Options

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

Copyright © 2013 Synopsys, Inc.

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

3. Control file display with the following:
   - Automatically display all the files, by checking Show Project Library. If this is unchecked, the Project view does not display files until you click on the plus symbol and expand the files in a folder.
   - Check one of the boxes in the Project File Name Display section of the form to determine how filenames are displayed. You can display just the filename, the relative path, or the absolute path.
4. To view project files in customized custom folders, check View Project Files in Custom Folders. For more information, see *Creating Custom Folders*, on page 220. Type folders are only displayed if there are multiple types in a custom folder.

5. To open more than one implementation in the same Project view, check Allow Multiple Projects to be Opened. You can only use multiple implementations with the Synplify Pro and Synplify Premier tools.

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

Updating Verilog Include Paths in Older Project Files

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

- Manually edit the `prj` file in a text editor and add the following on the line before each set_option -include_path:

```
set_option -project_relative_includes 1
```

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

Copyright © 2013 Synopsys, Inc.

Managing Project File Hierarchy

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

- Creating Custom Folders
- Manipulating Custom Project Folders
- Manipulating Custom Files

This information applies to file management; for information about managing and working with hierarchical projects, see *Working with Hierarchical Projects*, on page 254.

Creating Custom Folders

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

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

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

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

– To rename a folder, right-click on the folder and select Rename from the popup menu. The Rename Folder dialog box appears; specify a new name.

2. Use the Add Files to Project dialog box to add the entire contents of a folder hierarchy, and optionally place files into custom folders corresponding to the OS folder hierarchies listed in the dialog box display.

– To do this, select the Add File button in the Project view.

– Select any requested folders such as dsp from the dialog box, then click the Add button. This places all the files from the dsp hierarchy into the custom folder you just created.

Copyright © 2013 Synopsys, Inc.

– To automatically place the files into custom folders corresponding to the OS folder hierarchy, check the option called Add Files to Custom Folders on the dialog box.

– By default, the custom folder name is the same name as the folder containing files or folder to be added to the project. However, you can modify how folders are named, by clicking on the Folders Option button. The following dialog box is displayed.

To use:

– Only the folder containing files for the folder name, click on Use OS Folder Name.

– The path name to the selected folder to determine the level of hierarchy reflected for the custom folder path.

3. You can drag and drop files and folders from an OS Explorer application into the Project view. This feature is available on Windows and Linux desktops running KDE.

   – When you drag and drop a file, it is immediately added to the project. If no project is open, the software creates a project.

   – When you drag and drop a file over a folder, it will be placed in that folder. Initially, the Add Files to Project dialog box is displayed asking you to confirm the files to be added to the project. You can click OK to

accept the files. If you want to make changes, you can click the Remove All button and specify a new filter or option.

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

Manipulating Custom Project Folders

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

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

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

     For example, if the existing custom folder directory is:

     `/Examples/Verilog/RTL`

Copyright © 2013 Synopsys, Inc.

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

Manipulating Custom Files

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

1. To suppress the display of files in the Type folders, right-click in the Project view and select Project View Options or select Options->Project View Options. Disable the option View Project Files in Type Folders on the dialog box.

2. To display files in alphabetical order instead of project order, check the Sort Files button in the Project view control panel. Click the down arrow key in the bottom-left corner of the panel to toggle the control panel on and off.

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

Copyright © 2013 Synopsys, Inc.

Setting Up Implementations

Synplify Pro and Synplify Premier

Implementations are extensions of the project metaphor used in the Synplify Pro and Synplify Premier synthesis software. An implementation is one version of the project, implemented with a specific set of constraints and other settings. A project can contain multiple implementations, each with its own settings.

The Synplify software does not support multiple implementations.

Working with Multiple Implementations

Synplify Pro and Synplify Premier

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

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

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

   The new implementation uses the same source code files, but different device options and constraints. It copies some files from the previous implementation: the `tlg` log file, the `srs` RTL netlist file, and the *design*_fsm.sdc file generated by FSM Explorer. The software keeps a repeatable history of the synthesis runs.

2. Run synthesis again with the new settings.

– To run the current implementation only, click Run.

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

You can use multiple implementations to try a different part or experiment with a different frequency. See *Setting Logic Synthesis Implementation Options*, on page 228 for information about setting options.

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

3. Compare the results.

   – Use the Watch window to compare selected criteria. Make sure to set the implementations you want to compare with the Configure Watch command. See *Using the Watch Window*, on page 518 for details.

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

   – To compare details, compare the log file results.

4. To rename an implementation, click the right mouse button on the implementation name in the project view, select Change Implementation Name from the popup menu, and type a new name.

   Note that the current UI overwrites the implementation; releases prior to 9.0 preserve the implementation to be renamed.

5. To copy an implementation, click the right mouse button on the implementation name in the project view, select Copy Implementation from the popup menu, and type a new name for the copy.

6. To delete an implementation, click the right mouse button on the implementation name in the project view, and select Remove Implementation from the popup menu.

Copyright © 2013 Synopsys, Inc.

Setting Logic Synthesis Implementation Options

You can set global options for your synthesis implementations, some of them technology-specific. This section describes how to set global options like device, optimization, and file options with the Implementation Options command. For information about setting constraints for the implementation, see *Specifying SCOPE Constraints*, on page 135. For information about overriding global settings with individual attributes or directives, see *Specifying Attributes and Directives*, on page 245.

This section discusses the following topics:

- Setting Device Options, on page 228
- Setting Optimization Options, on page 231
- Specifying Global Frequency and Constraint Files, on page 234
- Specifying Result Options, on page 237
- Specifying Timing Report Output, on page 238
- Setting Verilog and VHDL Options, on page 239

Setting Device Options

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

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

2. Select the technology, part, package, and speed. Available options vary, depending on the technology you choose; the Synplify Premier software supports a subset of the technologies supported by the Synplify and Synplify Pro tools.

Copyright © 2013 Synopsys, Inc.

Also, when using the Synplify Premier tool with a HAPS board, selecting a Synopsys HAPS entry from the Technology drop-down menu automatically fills in the appropriate part, package, and speed grade for the corresponding device on the HAPS board.

Similarly, specifying a Synopsys HAPS technology with a set_option -technology command selects the appropriate part, package, and speed grade (for example, set_option -technology HAPS-60 selects a Virtex-6 XC6VLX760 device in an FF1760 package in speed grade 1).

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

Copyright © 2013 Synopsys, Inc.

For more information about setting fanout limits, pipelining, and retiming, see *Setting Fanout Limits*, on page 444, *Pipelining*, on page 426, and *Retiming*, on page 430, respectively. For details about other vendor-specific options, refer to the appropriate vendor chapter and technology family in the *Reference Manual*. Note that the Synplify tool does not support all these optimization options.

Device Mapping Options Vary by Technology

4. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options*, on page 228 for a list of choices). Click OK.

5. Click the Run button to synthesize the design. The software compiles and maps the design using the options you set.

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

The following table shows a majority of the device options.

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

Setting Optimization Options

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

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

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

Copyright © 2013 Synopsys, Inc.

Optimization Options

Project View

Implementation Options->Options

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

Fast Synthesis	Using Fast Synthesis, on page 1063
Auto Compile Point	The Automatic Compile Point Flow, on page 1109
Continue on Error	Continue on Error, on page 233
Physical Plus	Running Physical Synthesis, on page 506
FSM Compiler	*Optimizing State Machines*, on page 454
FSM Explorer	*Running the FSM Explorer*, on page 459 *Note*: Only a subset of the Xilinx, Altera, and Microsemi technologies support the FSM Explorer option. Use the Project->Implementation Options->Options panel to determine if this option is supported for the device you specify in your tool.
Resource Sharing	*Sharing Resources*, on page 448
Pipelining	*Pipelining*, on page 426

Retiming	*Retiming*, on page 430
Power Optimization	*Power Optimization for Xilinx Designs*, on page 2105 in the *Reference Manual*
HDL Analyst Quick Load	*HDL Analyst Quick Load*, on page 234

The equivalent Tcl set_option command options are as follows:

Option	set_option Tcl Command Option
Fast Synthesis	`-fast_synthesis {1|0}`
Auto Compile Point	`-automatic_compile_point {1|0}`
Continue on Error	`-continue_on_error {1|0}`
Physical Plus	`-run_physical_plus {1|0}`
FSM Compiler	`-symbolic_fsm_compiler {1|0}`
FSM Explorer	`-use_fsm_explorer {1|0}`
Resource Sharing	`-resource_sharing {1|0}`
Pipelining	`-pipe {0|1}`
Retiming	`-retiming {1|0}`
Power Optimization	`-power {1|0}`
HDL Analyst Quick Load	`-hdl_qload {1|0}`

3. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options*, on page 228 for a list of choices). Click OK.
4. Click the Run button to run synthesis.

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

Continue on Error

The “continue-on-error” feature is available for users of Synplify Premier to allow the compilation process to continue for certain, non-syntax-related compiler errors. For more information, see *Continue on Error*, on page 531.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Quick Load

By default, the software reads the entire design, performs logic optimizations and timing propagation, and writes output to a single netlist (`srs`). As designs get larger, the time to run and debug the design becomes more challenging.

The "HDL Analyst Quick Load" feature is available for Synplify Premier users. This feature allows the compiler to pre-partition the design into multiple modules that are written to separate netlist files (`srs`). To enable this feature, select the HDL Analyst Quick Load checkbox on the Options tab of the Implementation Options dialog box. This feature improves memory usage significantly for large designs.

This feature can also be enabled from the Tcl Script window using the following set_option Tcl command:

```
set_option -hdl_qload 1
```

When the HDL Analyst Quick Load option is enabled, the HDL Analyst tool displays these top-level RTL modules. The tool can take advantage of this feature by dynamically loading only the affected design hierarchy. For example, the hierarchy browser can expand only the lower-level hierarchy as needed for quick load.

Specifying Global Frequency and Constraint Files

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

1. To set a global frequency, do one of the following:
   - Type a global frequency in the Project view.
   - Open the Implementation Options form by clicking the Implementation Options button or selecting Project->Implementation Options, and click the Constraints tab.

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

   You can override the global frequency with local constraints, as described in *Specifying SCOPE Constraints, on page 135*. In the Synplify Pro tool, you can automatically generate clock constraints for your design instead of setting a global frequency. See *Using Auto Constraints, on page 739* for details.

Global Frequency and Constraints

Project View

Implementation Options->Constraints

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

   To create constraint files, see *Specifying SCOPE Constraints*, on page 135.

Copyright © 2013 Synopsys, Inc.

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

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

4. To specify or remove a Synplify Premier design plan (`sfp`), use the techniques described in steps 2 and 3, or do the following:
   - Select Project->Implementation Options->Design Planning. Check the box next to the file you want.
   - To delete a file, disable the check box next to the file name on the Design Planning tab.

   When the implementation is synthesized, the Synplify Premier tool uses the region assignments in this file for the second phase of optimization to perform physical synthesis.

5. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options*, on page 228 for a list of choices). Click OK.

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

Specifying Result Options

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

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

2. Specify the output files you want to generate.
   - To generate mapped netlist files, click Write Mapped Verilog Netlist or Write Mapped VHDL Netlist.
   - To generate a vendor-specific constraint file for forward annotation, click Write Vendor Constraint File. See *Generating Constraint Files for Forward Annotation*, on page 125 for more information.
3. Set the directory to which you want to write the results.
4. Set the format for the output file. The equivalent Tcl command for scripting is project -result_format *format*.

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

Copyright © 2013 Synopsys, Inc.

For certain Altera technologies (see *Generating Vendor-Specific Output, on page 834*), the `vqm` result format allows you to also select the version of Quartus II you are using from the pop-up menu.

5. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options, on page 228* for a list of choices). Click OK.

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

Specifying Timing Report Output

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

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

3. Specify the number of start and end points you want to see reported in the critical path sections.

4. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options*, on page 228 for a list of choices). Click OK.

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

Setting Verilog and VHDL Options

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

Setting Verilog File Options

You set Verilog file options by selecting either Project->Implementation Options->Verilog, or Options->Configure Verilog Compiler. For information about creating always block hierarchy for Synplify Premier, see *Setting Synplify Premier Prototyping Tools Optimizations*, on page 302.

1. Specify the Verilog format to use.

Copyright © 2013 Synopsys, Inc.

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

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

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

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

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

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

5. In the Include Path Order, specify the search paths for the include commands for the Verilog files that are in your project. Use the buttons in the upper right corner of the box to add, delete, or reorder the paths.

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

7. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options*, on page 228 for a list of choices). Click OK.

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

Copyright © 2013 Synopsys, Inc.

Setting VHDL File Options

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

For VHDL source, you can specify the options described below. For information about creating process hierarchy for Synplify Premier, see *Setting Synplify Premier Prototyping Tools Optimizations*, on page 302.

1. Specify the top-level module if you did not already do this in the Project view. If the top-level module is not located in the default work library, you must specify the library where the compiler can find the module. For information on how to do this, see *VHDL Panel*, on page 207.

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

2. For user-defined state machine encoding, do the following:
   - Specify the kind of encoding you want to use.
   - Disable the FSM compiler.

   When you synthesize the design, the software uses the compiler directives you set here to encode the state machines and does not run the

FSM compiler, which would override the compiler directives. Alternatively, you can define state machines with the syn_encoding attribute, as described in *Defining State Machines in VHDL*, on page 341.

3. To extract generics from the source code, do this:
   - Click Extract Generic Constants.
   - To override the default, enter a new value for a generic.

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

4. To push tristates across process/block boundaries, check that Push Tristates is enabled. For details, see *Push Tristates Option*, on page 220 in the *Reference Manual*.

5. Determine the interpretation of the synthesis_on and synthesis_off directives:
   - To make the compiler interpret synthesis_on and synthesis_off directives like translate_on/translate_off, enable the Synthesis On/Off Implemented as Translate On/Off option.
   - To ignore the synthesis_on and synthesis_off directives, make sure that this option is not checked. See translate_off/translate_on, on page 1593 in the *Reference Manual* for more information.

6. Set other implementation options as needed (see *Setting Logic Synthesis Implementation Options*, on page 228 for a list of choices). Click OK.

Copyright © 2013 Synopsys, Inc.

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

Specifying Attributes and Directives

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

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

	Attributes	Directives
VHDL	Yes	Yes
Verilog	Yes	Yes
cdc Compiler Directives File (*Synplify Premier tools only*)	No	Supported directives*
SCOPE Editor	Yes	No
Constraints File	Yes	No

* See Specifying Directives in a CDC File, on page 248

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

If compiler directives (`cdc`), SCOPE/constraints file, and the HDL source code are specified for a design, the constraints has the highest priority when there are conflicts. Then, the `cdc` compiler directives take precedence over the HDL source code.

For further details, refer to the following:

- Specifying Attributes and Directives in VHDL, on page 246
- Specifying Attributes and Directives in Verilog, on page 247
- Specifying Directives in a CDC File, on page 248
- Specifying Attributes Using the SCOPE Editor, on page 250
- Specifying Attributes in the Constraints File, on page 253

Copyright © 2013 Synopsys, Inc.

Specifying Attributes and Directives in VHDL

You can use other methods to add attributes to objects, as listed in *Specifying Attributes and Directives*, on page 245. However, you can specify directives only in the source code. There are two ways of defining attributes and directives in VHDL:

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

For details of VHDL attribute syntax, see *VHDL Attribute and Directive Syntax*, on page 963in the *Reference Manual.*

Using the Predefined VHDL Attributes Package

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

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

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

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

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

   For example:

   ```
   entity simpledff is
      port (q: out bit_vector(7 downto 0);
            d : in bit_vector(7 downto 0);
            clk : in bit);
   attribute syn_noclockbuf of clk : signal is true;
   ```

   For details of the syntax conventions, see *VHDL Attribute and Directive Syntax*, on page 963 in the *Reference Manual.*

3. Add the source file to the project.

Declaring VHDL Attributes and Directives

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

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

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

   For example:

   ```
   entity simpledff is
      port (q: out bit_vector(7 downto 0);
            d : in bit_vector(7 downto 0);
            clk : in bit);
   attribute syn_noclockbuf : boolean;
   attribute syn_noclockbuf of clk :signal is true;
   ```

2. Add the source file to the project.

Specifying Attributes and Directives in Verilog

You can use other methods to add attributes to objects, as described in *Specifying Attributes and Directives*, on page 245. However, you can specify directives only in the source code.

Verilog does not have predefined synthesis attributes and directives, so you must add them as comments. The attribute or directive name is preceded by the keyword synthesis. Verilog files are case sensitive, so attributes and directives must be specified exactly as presented in their syntax descriptions. For syntax details, see *Verilog Attribute and Directive Syntax*, on page 758 in the *Reference Manual*.

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

Verilog Block Comment Syntax	Verilog Line Comment Syntax
/* **synthesis** *attributeName* = *value* */	// **synthesis** *attributeName* = *value*
/* **synthesis** *directoryName* = *value* */	// **synthesis** *directoryName* = *value*

Copyright © 2013 Synopsys, Inc.

For details of the syntax rules, see *Verilog Attribute and Directive Syntax, on page 758* in the *Reference Manual*. The following are examples:

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

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

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

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

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

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

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

Specifying Directives in a CDC File

This feature is available in the Synplify Premier tools.

A cdc file provides a convenient way to specify supported directives, without making changes to your HDL files. You can specify the following directives on views, entities, architectures and modules:

- *syn_black_box*
- *syn_noprune*
- *syn_preserve*
- *syn_sharing*
- *syn_rename_module*
- *syn_unique_inst_module*

Use this procedure to create a cdc file and specify directives:

1. Create a constraints file with a cdc extension that contains the Tcl directives you want. To use the compiler directives editor, see *Using the Compiler Directives Editor*, on page 102.

2. Use the following syntax for the directives you want. Use the syntax that matches the HDL source code you are using.

VHDL	**define_directive {v:**[*libraryName*.]*entityName*[(*architectureName*)]**} {***directive***} {***value***}**
Verilog	**define_directive {v:**[*libraryName*.]*moduleName***} {***directive***} {***value***}**

The following example sets the syn_black_box attribute on all architectures of the sub entity in the MyLib library:

```
define_directive {v:MyLib.sub}{syn_black_box}{1}
```

You must specify the attribute or directive on a view (v:). The *libraryName* and *architectureName* arguments are optional. If you do not specify a library, the tool defaults to all design libraries. If you include an architecture, make sure to enclose it in parentheses. Note that Verilog objects are case-sensitive, but VHDL objects are not. See *Compiler Directives File Examples*, on page 1168 in the *Reference Manual* for examples.

3. Add the file to your project. The tool creates a new directory in the Project view.

Note, you can also use the set_option -compiler_constraint command to add cdc files to your project (see *set_option*, on page 1679 of the *Reference Manual*).

4. Compile the design.

Copyright © 2013 Synopsys, Inc.

When the design is compiled, the tool passes all active cdc files to the compiler. The compiler references the object names in these files with the original RTL objects and assigns the corresponding directives.

Specifying Attributes Using the SCOPE Editor

The SCOPE window provides an easy-to-use interface to add any attribute. You cannot use it for adding directives, because they must be added to the source files. (See *Specifying Attributes and Directives in VHDL*, on page 246 or *Specifying Attributes and Directives in Verilog*, on page 247). The following procedure shows how to add an attribute directly in the SCOPE window.

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

2. Click the Attributes tab at the bottom of the SCOPE window.

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

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	<any>	<global>	syn_noclockbuf	1	boolean	Use normal input buffer
2	☑						
3	☑						
4	☑						
5	☑						
6							
7							
8							

Attributes

3. To specify the object, do one of the following in the Object column. If you already specified the attribute, the Object column lists only valid object choices for that attribute.

   – Select the type of object in the Object Filter column, and then select an object from the list of choices in the Object column. This is the best way to ensure that you are specifying an object that is appropriate, with the correct syntax.

– Drag the object to which you want to attach the attribute from the RTL or Technology views to the Object column in the SCOPE window. For some attributes, dragging and dropping may not select the right object. For example, if you want to set syn_hier on a module or entity like an and gate, you must set it on the view for that module. The object would have this syntax: v:*moduleName* in Verilog, or v:*library.moduleName* in VHDL, where you can have multiple libraries.

– Type the name of the object in the Object column. If you do not know the name, use the Find command or the Object Filter column. Make sure to type the appropriate prefix for the object where it is needed. For example, to set an attribute on a view, you must add the v: prefix to the module or entity name. For VHDL, you might have to specify the library as well as the module name.

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

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

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

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

6. Save the file.

Copyright © 2013 Synopsys, Inc.

The software creates a Tcl constraint file composed of define_attribute statements for the attributes you specified. See *How Attributes and Directives are Specified*, on page 1166 of the *Reference Manual* for the syntax description.

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

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

Specifying Attributes in the Constraints File

When you use the SCOPE window (*Specifying Attributes Using the SCOPE Editor*, on page 250), the attributes are automatically written to a constraint file using the Tcl define_attribute syntax. This is the preferred method for defining constraints as the syntax is determined for you.

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

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

   define_attribute {*objectName***}** *attributeName value*

   For commands and syntax, see *Attribute and Directive Summary (Alphabetical)*, on page 1170 in the *Reference Manual*.
3. Save the constraints in a file using the FDC file extension.

The following code excerpt provides an example of attributes defined in the constraint file. (Some of these attributes are specific to Xilinx devices):

```
# Assign a location for scalar port "sel".
   define_attribute {sel} xc_loc "P139"

# Assign a pad location to all bits of a bus.
   define_attribute {b[7:0]} xc_loc "P14, P12, P11, P5, P21,
      P18, P16, P15"

# Assign a fast output type to the pad.
   define_attribute {a[5]} xc_fast 1

# Use a regular buffer instead of a clock buffer for clock "clk_slow".
   define_attribute {clk_slow} syn_noclockbuf 1

# Relax timing by not buffering "clk_slow", because it is the slow clock
# Set the maximum fanout to 10000.
   define_attribute {clk_slow} syn_maxfan 10000
```

For information about editing constraints, see *Using a Text Editor for Constraint Files (Legacy)*, on page 122.

Copyright © 2013 Synopsys, Inc.

Working with Hierarchical Projects

You can use the top-down or bottom-up flows to develop hierarchical designs. You can also use top-down or bottom-up synthesis or a combination of the two to synthesize these designs. For more information about these flows, see *Hierarchical Project Management Flows*, on page 86.

This section describes how to set up and work with hierarchical projects in the flows mentioned above. The following topics provide more detail:

- Creating Hierarchical Subprojects by Exporting Blocks, on page 254
- Creating Hierarchical Subprojects by Exporting Instances, on page 257
- Creating Nested Subprojects, on page 263
- Generating Dependent File Lists, on page 269
- Working with Multiple Implementations in Hierarchical Projects, on page 270
- Working with Multiple Instances and Parameterized Modules, on page 271
- Allocating Resources for Subprojects, on page 275
- Setting Initial Timing Budgets for Subprojects, on page 277
- Configuring Synthesis Runs for Hierarchical Projects, on page 278
- Analyzing Synthesis Results for Hierarchical Projects, on page 282

Creating Hierarchical Subprojects by Exporting Blocks

When you use a top-first approach to team design (*Top-First Development Flow for Hierarchical Projects*, on page 89), you must partition the design and specify subprojects. This procedure shows you how to designate a block as a subproject; see *Creating Hierarchical Subprojects by Exporting Instances*, on page 257 for information about creating subprojects based on instances.

Unlike an instance-based subproject, any changes made to a block-based subproject affect all instances of the block.

1. Open the compiled top-level design.

You must compile the design because you cannot export subprojects from an uncompiled design. It does not matter if the top level has black boxes.

2. Export a subproject.
   – Go to the Design Hierarchy view, and right-click the block you want to make into a subproject; then select Create Subproject (Design Block) from the popup menu.

The Export "*blockname*" as subproject dialog box opens.

Copyright © 2013 Synopsys, Inc.

- Specify a name for the project and a location.
- Specify source files for the subproject. Check the selected files in the list. The tool automatically enables needed source files from the top-level design, but the list might not be complete. If a source file is not listed, click the Add File button and add the missing file to the list. The file is automatically selected when it is added with Add File. When the tool creates the subproject, it adds the selected files to it.

  You must have at least one file in order to create a subproject. Some files may overlap with the parent project or a sibling project.
- To link updates to the projects, enable the Link sub projects to the parent project option.
- Click OK.

The tool creates a new subproject for the specified block and automatically links it to the top-level project. The block icon in the Design Hierarchy view changes to a green rectangle with a P in it, to indicate that it is now defined as a subproject. In the Project Files view, you see a Sub-projects directory that contains the new block project. The implementation

options for the block project match the options of the active implementation of the parent project.

See *Working with Multiple Implementations in Hierarchical Projects*, on page 270 and *Working with Multiple Instances and Parameterized Modules*, on page 271 for information about creating additional implementations for the block.

3. Save the design. Until you do so, the hierarchy you created is not saved.

4. Synthesize the design, following the instructions in *Top-Down Synthesis for Hierarchical Projects*, on page 93 and *Bottom-Up Synthesis for Hierarchical Projects*, on page 91.

Creating Hierarchical Subprojects by Exporting Instances

When using the top-down development flow (*Top-First Development Flow for Hierarchical Projects*, on page 89) you can also generate hierarchy by separating an instance into an independent subproject, instead of a block (*Creating Hierarchical Subprojects by Exporting Blocks*, on page 254). The following procedure describes how to export instances for a hierarchical project and then synthesize the design.

An instance-based subproject is not connected to any other instances, so any changes made to an instance-based subproject only affect that instance.

Copyright © 2013 Synopsys, Inc.

1. Load the design for your hierarchical project. Once loaded, run Compile Only from the Run menu.

You must compile the design because you cannot export subprojects from an uncompiled design.

2. Create the subproject.
   - Click on the Design Hierarchy tab, which shows both the design blocks and instances for your design.

– Expand the hierarchy if needed, right-click the instance you want and select Create Subproject (Instance).

– Select the input files required for this instance. By default, the synthesis tool picks the required RTL files, but can miss auxiliary files such as `include files that define macros or packages. You must manually add these files by checking the corresponding check box.

Copyright © 2013 Synopsys, Inc.

The instance is highlighted and marked with a green icon [P] ( ) for the subproject. The subproject name is the instance name with the hierarchical path for that instance.

3. Save the design.

   Until you save the design, the hierarchy you created is not written to the disk or saved as part of the project.

4. Repeat the previous steps for each hierarchical instance to export.

The Project Files view shows that each exported instance has become an independent subproject, identified by the green icon.

5. Edit synthesis options for the subprojects, as needed.
   - In the Project Files view, right-click the project file, select Hierarchical Project Options from the pull-down menu.

Copyright © 2013 Synopsys, Inc.

– In the window that opens, change options for any of the subprojects, as needed. For example, you can change Run Type from bottom_up to top_down.

	Top: top	Sub: ablock_inst_b2_a2	Sub: bblock_inst_c1_b2	Sub: bblock_inst_c1_b1
Implementation:	rev_1	rev_1	rev_1	rev_1
Run Type:		top_down	bottom_up	bottom_up
Options:				
Technology	Virtex7	Virtex7	Virtex7	Virtex7
Technology part	XC7VX485T	XC7VX485T	XC7VX485T	XC7VX485T
Technology package	FFG1157	FFG1157	FFG1157	FFG1157
Fast Synthesis	☐	☐	☐	☐
Enhanced Optimization	☑	☑	☑	☑
Fanout Guide	10000	10000	10000	10000
Disable I/O Insertion	☐	☑	☑	☑
FSM Explorer	☐	☐	☐	☐
Pipelining	☑	☑	☑	☑
Retiming	☐	☐	☐	☐
Fix Gated Clocks	3	3	3	3
Fix Generated Clocks	3	3	3	3
Auto Constrain IO	☐	☐	☐	☐
Disable Sequential Optimizations	☐	☐	☐	☐

– Click OK.

6. Synthesize the design by clicking the Run button.

Each exported instance is treated as a subproject and linked to the top-level module. Alternatively, you can select which instances to link for the exported block. Exported blocks for this example are ablock_inst_b2_a2 and bblock_inst_c1.

Any instances that are not exported are linked to the module definition in the top-level project.

You can view the dependent files for a subproject or generate a list of them, as described in *Generating Dependent File Lists*, on page 269.

7. Analyze your synthesis results in the HDL Analyst views, as described in *Analyzing Synthesis Results for Hierarchical Projects*, on page 282.

Creating Nested Subprojects

With hierarchical design, you can have subprojects that contain other subprojects. You can use the block and instance exporting techniques to recursively export parts of the design as subprojects.

Copyright © 2013 Synopsys, Inc.

The following procedure shows how to export nested subprojects, and is based on a design containing three instances of ablock. The ablock module instantiates two bblock instances (b1, b2), bblock instantiates two instances of cblock (c1, c2), and cblock instantiates two instances of dblock (d1, d2).

1. Load the top-level project and compile it with Run->Compile Only.

   You must compile the design before you can export an instance as a subproject. The Design Hierarchy tab and the RTL view show the a1, a2, and a3 instances at ablock at the top level.

2. Create a subproject.
   - In the Design Hierarchy view, right-click an instance (a3) and select one of the Create Sub-project commands. For details about using them to export subprojects, see *Creating Hierarchical Subprojects by Exporting*

Blocks, on page 254 and *Creating Hierarchical Subprojects by Exporting Instances*, on page 257.

– A dialog box shows you the files needed for the selected instance. Edit the list as needed to reflect the file list for the instance. For details, see *Generating Dependent File Lists*, on page 269.

– Click OK. The view now shows a green P icon next to the instance to indicate it is an exported subproject.

The Project Files view shows the newly-created subproject. It has its own implementation, separate from the top-level implementation.

3. Save the design.

Copyright © 2013 Synopsys, Inc.

Until you save the design, the subproject you just created is not written to disk.

4. Create a subproject within a subproject.
   - Compile the subproject under which you want to create another subproject (a3), by selecting the subproject implementation and selecting Run->Compile Only.

     You must compile again at this level, because you cannot export a subproject from an uncompiled design, and at this point the a3 subproject is not compiled.

   - In the Design Hierarchy view, right-click an instance and select one of the Create Sub-project commands to export it. You can select any instance in the hierarchy. The following figure shows the current subproject expanded, the next level skipped (bblock), and a cblock instance selected from two levels below:

– Check that the list of dependent files for the instance is accurate in the window that opens, and click OK when you are done. The Design Hierarchy and Project Files views reflect the creation of a new cblock subproject under the parent ablock subproject.

5. Save the design.

6. Compile and map the entire design by selecting the top-level implementation and clicking Run.

   The top-level RTL shows exported subprojects in green, to indicate that they were exported. The following figure shows the a3 instance that was exported from the top level.

Copyright © 2013 Synopsys, Inc.

The RTL view indicates exported subprojects relative to the current level. If you push into the a3 instance from the top level, and then descend into the bblock instance, the exported cblock instance is not in green, because the reference point is the top level. However, if you select the ablock subproject and then generate an RTL view, the cblock instance is in green when you push down to it. This is because the cblock instance was exported relative to the ablock subproject.

Generating Dependent File Lists

The compiler automatically generates a comprehensive list of the dependent files for a subproject when you create it, and displays the files from the parent project in a window like this one:

There are two ways to explicitly generate a list of dependent files for a project:

1. Select an instance in the Design Hierarchy view and then select Generate Dependent File List from the popup menu.

2. Right-click an instance or block in the RTL view, and select Generate Dependent File List from the popup menu.

Copyright © 2013 Synopsys, Inc.

This command writes the list of dependent files to a text file in the dm subdirectory under the project directory, and opens the text file. Dependent files include the following files:

- Language standard and compiled library information files
- All required dependent (child) and referencing (parent) files
- Library and include directory paths
- The path order for any include files

Working with Multiple Implementations in Hierarchical Projects

The following procedure show you how to work with multiple implementations in hierarchical projects. For example, you might want to create multiple implementations with different run configurations if you are experimenting with different implementation options.

1. Open the top-level hierarchical project and compile it (F7).

   This updates the hierarchical project views.

2. To add an implementation to the top-level project or a block-level project, do the following in the Project Files view:
   - Right-click the top level or the block and select Add Implementation.
   - Set the new implementation options you want.

3. To add subproject implementations to the top-level project, do one of the following:
   - In the Project Files view, right-click a top-level implementation and select Add Sub-project Implementations.
   - Select a top-level implementation in the Project Files view. Select the Project-> Add SubProject Implementation command.

   The tool automatically adds a new implementation to each defined subproject, and updates the top-level implementation to use the new block implementations. The new block implementations are based on the selected top-level implementation.

4. To specify which implementation to run, do the following:
   - Select Project->Edit Run Configuration.

– In the dialog box, set Implementation to the block implementation you want to synthesize. To prevent synthesis from being run set this option to <off>. If there are red cells in the table, synchronize options across implementations as needed with the Sync buttons. Click OK.

– To run all implementations, select the Run->Run all Implementations instead of clicking the Run button to synthesize.

By default the tool runs the active implementation.

Working with Multiple Instances and Parameterized Modules

A module can have various parameters, such as SIZE or WIDTH. When multiple instances of a module have different parameter values, they are called parameterized modules. The following procedure shows you how to work with multiple instantiations of parameterized modules.

1. Create your design and instantiate the parameterized modules.

   In the design example used to illustrate this procedure, cblock is instantiated four times. The instantiations have different values defined for DWIDTH and USE_AND:

   ```
   `include "meminfo.h"
   parameter USE_AND = 1;
   //parameter USE_AND = 0;
   //parameter DWIDTH = 2;
   parameter DWIDTH = 6;
   ```

   The following table shows the parameter values for the four cblock instances in the design. It also shows that there are three unique sets of parameters between the instances; cblock_1 and cblock_3 share the same parameter set.

cblock Instances	USE_AND	DWIDTH	Parameter Set
cblock_1	0	6	1
cblock_2	1	6	2
cblock_3	0	6	1
dblock_4	1	2	3

2. Export one instance of the parameterized module.

Copyright © 2013 Synopsys, Inc.

– Compile the design.

– Open the Design Hierarchy tab, and select one instance of the parameterized module. You can select any instance of the module in the design to export.

– Export the subproject by right-clicking the module and selecting the appropriate Create Subproject command.

To...	Export Command
Work on a specific instantiation without affecting other instantiations	Create Subproject from Instance
Handle multiple instantiations together, and fine-tune a few instances	Create Subproject from Block

The tool creates the subprojects based on the export option you selected. Instance-based export results in a single subproject that only affects the selected module. A block-based export creates a single subproject that affects all instances of the module. For parameterized modules exported as blocks, the tool creates a single subproject with multiple implementations, one for each unique parameter set.

– If required, add any additional files needed for the subproject to the automatically generated list that displays, and click OK.

After you set the files for the subproject, the module is marked with a green icon in the Design Hierarchy view. Module-based subprojects are

denoted by rev_#(%), as shown in the following figure, where cblock was exported as a module.

Copyright © 2013 Synopsys, Inc.

– Check the Project Files view. It shows that a subproject was created for cblock, and that it is a parameterized implementation:

3. Click the Save All icon ( ) in the top bar to save all the exported subproject and project file information.

4. Set options and constraints for synthesis.

   For instance-based subprojects, the options and constraints only apply to the selected instance. For module-based subprojects, the options and constraints affect all instances of the module.

5. Run the project.

   All exported subprojects are synthesized from the bottom up by default. If you want to allow the subproject to be optimized, you can change the default, as described in *Configuring Synthesis Runs for Hierarchical Projects*, on page 278.

6. If you make changes to parameters in a design, do the following:

   – Right-click the top-level project implementation and select Subproject Parameter Sync.

- Rerun synthesis.

Allocating Resources for Subprojects

You can automatically allocate RAM and DSP resources among the subprojects in your design by using the following procedure:

1. Create a hierarchical subproject from scratch. It is important that you start with a new design.
   - Create the top-level project.
   - Optionally, specify RAM and DSP resource limits at the top level with the syn_allowed_resources attribute. If none are specified, the tool uses the defaults for the technology.
   - Compile the design.
   - Create instance-based subprojects. You cannot set timing budgets for block-based subprojects. See *Creating Hierarchical Subprojects by Exporting Instances*, on page 257 for details.
   - Save the design.
2. To specify resources for a subproject, follow these steps:
   - Right-click a subproject or anywhere in the Design Hierarchy view and select Allocate Timing and Resource Budgets (Beta). The Tcl equivalent for this command is generate_instance_constraints.
   - In the window that opens, enable the checkbox in the Resource column for the subproject.

Copyright © 2013 Synopsys, Inc.

– Click Generate.

The synthesis tool estimates the RAM and DSP resources to the subproject based on the top-level specification or default and writes them out in an fdc file for the subproject.

3. Optionally, open the fdc file and edit the resource limits set with syn_allowed_resources for the subproject.

4. Synthesize the design.

The tool honors any resource allocation parameters set with the syn_allowed_resources attribute. You can set this attribute at the top level or the subproject level. The tool does not consider black boxes during resource allocation, so you might want to account for resource usage by black boxes when you set resource limits. The tool automatically accounts for resource usage in IPs, based on their netlists.

– If you do not explicitly specify the resource limits with the syn_allowed_resources attribute, the tool uses the defaults defined for the technology.

– If syn_allowed_resources is defined at both the subproject and top levels, resource allocation is determined by the methodology used for synthesis. If you synthesize from the bottom up, the tool bases resource allocation on the subproject limits, but it uses the top-level resource limits if the design is synthesized from the top down. See *Configuring Synthesis Runs for Hierarchical Projects, on page 278* for information about synthesis choices for hierarchical projects.

5. Check the resource usage sections of the subproject and top level log files to ensure that your design is not over-utilized.

Setting Initial Timing Budgets for Subprojects

When you first create a hierarchical design, you can automatically allocate timing budgets for the subprojects using the following procedure:

1. Create a hierarchical subproject from scratch. It is important that you start with a new design.
   - Create the top-level project.
   - Create the top-level constraints.
   - Compile the design.
   - Create instance-based subprojects. You cannot set timing budgets for block-based subprojects. See *Creating Hierarchical Subprojects by Exporting Instances*, on page 257 for details.
   - Save the design.
2. To specify timing budgets for the subproject, follow these steps:
   - Right-click a subproject or anywhere in the Design Hierarchy view and select Allocate Timing and Resource Budgets (Beta). The Tcl equivalent for this command is generate_instance_constraints.
   - In the window that opens, make sure the checkboxes in the Timing Budgets column are enabled for the subprojects.

Copyright © 2013 Synopsys, Inc.

– Click Generate.

The synthesis tool generates an initial constraints file for the subproject, based on the top level constraints and optimized slack across the whole design. It assigns timing paths in the subproject a percentage of the clock period.

The supported constraints are create_clock, create_generated_clock, group_path, set_clock_latency, set_clock_uncertainty, set_false_path, set_input_delay, set_max_delay, set_min_delay, set_multicycle_path, and set_output_delay. However, this command does not currently support the -through option for timing exceptions like false paths and multicycle delays; for these constraints use -from and -to as much as possible.

The tool automatically accounts for IP timing, based on their netlists, but it has a limitation with ngc cores in subprojects.

3. Open the subproject constraint file created in the previous step, and check the constraints.

   Edit them if needed. You can also add other constraints to the subproject as needed.

4. Synthesize the design from the bottom up.

   The tool synthesizes the subprojects based on the timing constraints defined for them.

5. Check timing results in the subproject and top-level log files.

Configuring Synthesis Runs for Hierarchical Projects

Before you synthesize the hierarchical project, set options for the synthesis run. This procedure shows you how to use the Hierarchical Project Options command and set synthesis options for the top level design or for the blocks. You can set different run configurations for different implementations.

1. Select the Hierarchical Project Options command.
   – Select the top level or the block in the Project Files view.
   – Either right-click and select Hierarchical Project Options from the popup menu or select Project->Hierarchical Project Options.

2. Set options in the dialog box that opens:
   - In the block column, set Implementations to the implementation you want to run. If you do not want to synthesize a block, set it to <off>. When you set a block to <off>, that subproject is treated as a black box.

Copyright © 2013 Synopsys, Inc.

Hierarchical Project options for camera_open_source_xlnx | rev_1

	camera_open_source_	Sub: camera_wb_if	Sub: camera_io_calc	Sub: camera_cb_table
Implementation:	rev_1	rev	rev_1	rev_1
Run Type:		bottom_up	top_down	bottom_up
Options:				
Technology	Virtex6	Virtex6	Virtex6	Virtex6
Technology part	XC6VLX75T	XC6VLX75T	XC6VLX75T	XC6VLX75T
Technology package	FF484	FF484	FF484	FF484
Fast Synthesis	☐	☐	☐	☐
Synplify Global Placement	☐	☐	☐	☐
Physical Synthesis	☐	☐	☐	☐
Enhanced Optimization	☑	☑	☑	☑
Fanout Guide	10000	10000	10000	10000
Disable I/O Insertion	☐	☐	☐	☐
FSM Explorer	☐	☐	☐	☐
Pipelining	☐	☐	☐	☐
Retiming	☐	☐	☐	☐
Fix Gated Clocks	0	0	0	0
Fix Generated Clocks	3	3	3	3
Auto Constrain IO	☐	☐	☐	☐
Disable Sequential Optimizations	☐	☐	☐	☐

Top-level and block-level options for [rev_1].
Conflicting options are highlighted; resolve them before running an implementation.

Synchronization with top level required Synchronization with top level optional

Synchronize All Options with Top Level OK Cancel

- Set Run Type for the block to reflect how you want it synthesized: top_down or bottom_up. For details of the differences, see *Bottom-Up Synthesis for Hierarchical Projects*, on page 91 and *Top-Down Synthesis for Hierarchical Projects*, on page 93.
- Set any other implementation options you want for the synthesis run.

3. If necessary, synchronize the top-level and block settings. If you see a red cell in the table, it means that option must be synchronized before synthesis.

   The following figure shows that pink highlighting in the device option field. This indicates that you must synchronize the subproject with the top level setting. The Disable I/O Insertion option is yellow and synchronization with the top level is optional.

	camera_open_source_	Sub: camera_wb_if	Sub: camera_io_calc	Sub: camera_cb_tab
Implementation:	rev_1	rev	rev_1	rev_1
Run Type:		bottom_up	top_down	bottom_up
Options:				
Technology	Virtex6	Virtex5	Virtex6	Virtex6
Technology part	XC6VLX75T	XC5VLX20T	XC6VLX75T	XC6VLX75T
Technology package	FF484	FF323	FF484	FF484
Fast Synthesis	☐	☐	☐	☐
Synplify Global Placement	☐	☐	☐	☐
Physical Synthesis	☐	☐	☐	☐
Enhanced Optimization	☑	☑	☑	☑
Fanout Guide	10000	10000	10000	10000
Disable I/O Insertion	☐	☐	☑	☐
FSM Explorer	☐	☐	☐	☐
Pipelining	☐	☐	☐	☐
Retiming	☐	☐	☐	☐
Fix Gated Clocks	0	0	0	0
Fix Generated Clocks	3	3	3	3
Auto Constrain IO	☐	☐	☐	☐

– Click the Synchronize All Options with Top Level button at the bottom of the box to match the required block device settings to the settings at the top level. If the required device options match, this button is grayed out and unavailable.

4. Click OK.

5. Synthesize the design.

See *Bottom-Up Synthesis for Hierarchical Projects*, on page 91, *Top-Down Synthesis for Hierarchical Projects*, on page 93 and *Mixed Block Synthesis for Hierarchical Projects*, on page 95 for details.

Copyright © 2013 Synopsys, Inc.

Analyzing Synthesis Results for Hierarchical Projects

After synthesizing a hierarchical project, you can analyze the results as described below.

1. To view the synthesis results, check the RTL view.

   The following figure of the top-level RTL view shows the linked netlists from subprojects.

   This is an example of an RTL view of the subprojects:

2. Check the log file for hierarchical reports for each subproject.
3. For resource usage, check the corresponding sections in the top-level and subproject log files.

Copyright © 2013 Synopsys, Inc.

Searching Files

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

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

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

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

Identifying the Files to Search

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

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

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

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

Filtering the Files to Search

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

- A pattern without a wildcard or a “.” (period) is interpreted as a filename extension. For example, `sdc` restricts the search to only constraint files.
- Multiple patterns can be specified using a semicolon delimiter. For example, v;vhd restricts the files searched to only Verilog and VHDL files.
- Wildcard characters can be used in the pattern to match file names. For example, a*.vhd restricts the files searched to VHDL files that begin with an “a” character.
- Leaving the File filter field empty searches all files that meet the Find In criteria.

Copyright © 2013 Synopsys, Inc.

- The Match Case, Whole Word, and Regular Expressions search options can be used to further restrict searches.

Initiating the Search

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

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

Search Results

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

fullpath_to_file(*lineNumber*): *matching_line_text*

For example, the entry

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

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

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

Archiving Files and Projects

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

Whenever you have a `sar` file that contains relative or absolute include paths for the files in the project, use the _SEARCHFILENAMEONLY_ directive to have the compiler remove the relative/absolute paths from the 'include and search only for the file names. Otherwise, you may have problems using the archive utility. For details, see *_SEARCHFILENAMEONLY_ Directive*, on page 218.

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

- Archive a Project
- Un-Archive a Project
- Copy a Project

Archive a Project

Use the archive utility to store the files for a design project into a single archive file in a proprietary format (`sar`). You can archive an entire project or selected files from a project. If you want to create a copy of a project without archiving the files, see *Copy a Project*, on page 294.

Here are the steps to create an archive:

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

   The Tcl command equivalent is project -archive. For a complete description of the project Tcl command options for archiving, see *project*, on page 1662 of the *Reference Manual*.

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

Copyright © 2013 Synopsys, Inc.

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

– Click Next.

The tool summary displays all the files in the archive and shows the full uncompressed file size. The actual size is smaller after the archiving operation as there is no duplication of files.

Copyright © 2013 Synopsys, Inc.

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

4. Verify that the current archive contains the files that you want, then click Archive which creates the project archive `sar` file and displays the following prompt:

5. Click Done if you are finished.

Un-Archive a Project

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

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

   The Tcl command equivalent is project -unarchive. For a complete description of the project Tcl command options for archiving, see *project*, on page 1662 of the *Reference Manual*.

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

Copyright © 2013 Synopsys, Inc.

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

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

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

6. Click Un-Archive.

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

Copyright © 2013 Synopsys, Inc.

7. If the destination directory already contains project files with the same name as the files you are extracting, you are prompted so that the existing files can be overwritten by the extracted files.

Copy a Project

Use this utility to create an unarchived copy of a design project. You can copy an entire project or just selected files from the project. However, if you want to create an archive of the project, where the entire project is stored as a single file, see *Archive a Project*, on page 287.

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

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

   The Tcl command equivalent is project -copy. For a complete description of the project Tcl command options for archiving, see *project*, on page 1662 of the *Reference Manual*.

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

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

Copyright © 2013 Synopsys, Inc.

Synplicity Copy Utility - [C:/Designs/ramCtrl/ramprj.prj]
Step 2: Please select files to be copied.
Project File List
Create project using:
Source Files
SRS
Add Extra Files...
Archive Summary: Total file(s): 41 Total Size: 2.1 M
C:/Designs/ramCtrl/ramprj.prj
Verilog
_____Designs_ramCtrl/alu.v
_____Designs_ramCtrl/data_mux.v
_____Designs_ramCtrl/eight_bit_uc.v
_____Designs_ramCtrl/ins_decode.v
_____Designs_ramCtrl/ins_rom.v
_____Designs_ramCtrl/io.v
_____Designs_ramCtrl/reg_file.v
_____Designs_ramCtrl/spcl_regs.v
rev_1 (File: 23 Size: 2.0 M)
Files in [C:/Designs/ramCtrl/]
Extra Input Files (File: 10 Size: 80.8 k)
C:/tools/syn880qt_056R/lib/pro.v
C:/Designs/ramCtrl/alu.v
C:/Designs/ramCtrl/data_mux.v
C:/Designs/ramCtrl/eight_bit_uc.v
rev_1 Type Modified
backup
syntmp
AutoConstr... Constral ... 10:31:16 25-Apr-2007
run_options... File 10:31:16 25-Apr-2007
spcl_regs.ar... File 10:31:16 25-Apr-2007
spcl_regs.edn File 10:31:16 25-Apr-2007
spcl_regs.fse File 10:31:16 25-Apr-2007
spcl_regs.htm HTML Lo... 10:32:19 25-Apr-2007
spcl_regs.map File 10:31:16 25-Apr-2007
spcl_regs.sap File 10:31:16 25-Apr-2007
spcl_regs.sdf File 10:31:16 25-Apr-2007
spcl_regs.srd File 10:31:16 25-Apr-2007
spcl_regs.srm Gate Net... 10:31:16 25-Apr-2007
spcl_regs.srr File 10:31:16 25-Apr-2007
spcl_regs.tlg File 10:31:16 25-Apr-2007
spcl_regs_s... Constral ... 10:31:16 25-Apr-2007
Active Impl.
Local File
Remote File Ref.
Remote File (will be copied locally before archiving)
Missing File
< Back
Next >
Cancel

– Click Next.

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

   This creates the project copy.

Copyright © 2013 Synopsys, Inc.

CHAPTER 6

Setting up a Physical Synthesis Project

The process to set up a traditional physical synthesis project is the same as for a logic synthesis project, but it requires a few additional steps. The following describe the additional steps needed to set up a physical synthesis project:

- Setting up for Physical Synthesis, on page 300
- Setting Options for Physical Synthesis, on page 302
- Setting Constraints for Physical Synthesis, on page 320
- Forward-Annotating Physical Synthesis Constraints, on page 323
- Backannotating Physical Synthesis Constraints, on page 326

Copyright © 2013 Synopsys, Inc.

Setting up for Physical Synthesis

This figure summarizes the steps for setting up a physical synthesis project. The shaded boxes are either optional steps, or steps that are specific to a flow or certain technologies. Although the figure makes a distinction between setting options or constraints for logic synthesis and physical synthesis, you can actually define both types of options at the same time.

See the following for details about physical synthesis setup:

For information about...	See...
Implementation options	Setting Options for Physical Synthesis, on page 302
Constraints	Setting Constraints for Physical Synthesis, on page 320

For information about...	See...
Creating floorplan files	Using Design Planner Floorplan Constraints, on page 320
Creating P&R implementations	Creating a Place and Route Implementation, on page 303
Creating P&R options files	Specifying Altera Place-and-Route Options, on page 308 Specifying Xilinx Place-and-Route Options in a Tcl File, on page 311

Copyright © 2013 Synopsys, Inc.

Setting Options for Physical Synthesis

After you have set up logic synthesis options for the implementation with the Implementation Options command (see *Setting Options for Physical Synthesis*, on page 302), you can set other options specific to physical synthesis.

The following describe them:

- Setting Synplify Premier Prototyping Tools Optimizations, on page 302
- Creating a Place and Route Implementation, on page 303
- Specifying Altera Place-and-Route Options, on page 308
- Specifying Xilinx Place-and-Route Options in a Tcl File, on page 311
- Specifying Xilinx Place-and-Route Options in an opt File, on page 314
- Specifying Xilinx Global Placement Options, on page 318

Setting Synplify Premier Prototyping Tools Optimizations

You can set these options if you have bit-sliced your design or for other restructuring operations. The prototyping tools optimizations are only available in the Synplify Premier tool.

1. Select Project->Implementation Options, and click on the GCC & Prototyping Tools tab.

2. To reduce the number of ports, eliminate feedthrough routes through modules by enabling Feedthrough Optimization.

3. To reduce area, enable Constant Propagation.

   Where possible, this option eliminates the logic used when constant inputs to logic cause their outputs to be constant. It is sometimes possible to eliminate this type of logic altogether during optimization.

4. To provide more granularity for applying a design plan to large modules at the always block or process level, enable Create Always/Process Level Hierarchy. Use this option with the Synplify Premier DP tool only.

   Currently a design plan can be applied to either modules or to individual gates, registers, and so on. For a module that is too large to fit in a row or defined region, you might need an extra level of granularity which is not as detailed as a gate-level description. This option creates an additional, intermediate level of hierarchy to which you can apply a design plan.

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

5. To remove unnecessary and redundant logic in the netlist, enable Optimize Netlist.

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

Creating a Place and Route Implementation

For Altera, Microsemi, and Xilinx technologies, the Synplify Pro and Synplify Premier tools allows you to run a place-and-route implementation after synthesis completes. The following steps show you how to manually create a place-and-route implementation.

1. Make sure you have the correct version of the P&R tool installed, and that all variables for the tool have been set.
   - Check the release notes for version information. Select Help->Online Documents->release_notes.pdf and go to *Third Party Tool Versions*.
   - For Microsemi technologies, set the Microsemi ALSDIR and PATH environment variables to point to a valid installation of the place and route tool.
   - For Altera technologies, set the QUARTUS_ROOTDIR and PATH environment variables to point to a valid installation of the place and route tool.

Copyright © 2013 Synopsys, Inc.

- For Xilinx technologies, set the XILINX and PATH environment variables to point to a valid installation of the place and route tool.

  The Synplify Premier UI automatically sets the PAR_BELDLYRPT environment variable to 1. This environment variable is set so that the Xilinx place-and-route placement file (`xdl`) is generated with a particular format. The software uses this `xdl` file to backannotate the placement information

2. To create a place-and-route implementation, do one of the following:
   - Click on the Add P& R Implementation button from the Project view.
   - Select an implementation in the Project view, then right-click and select Add New Place & Route Job.

   The Add New Place and Route Task dialog box opens. The available options differ slightly, depending on the technology.

3. Set the options you need. Available place-and-route options vary depending on the synthesis tool and technology.
   - Specify the Place & Route Job Implementation. The default is pr_*n*. Avoid using spaces in the implementation name.
   - Enable Run Place and Route following synthesis.
   - Enable Run Timing & Placement Backannotation + Generate Congestion Analysis Report.
   - Select a place-and-route options file. If you do not specify one, the tool uses the settings in the default file.

Copyright © 2013 Synopsys, Inc.

Microsemi	*install_dir*/lib/microsemi/microsemi_par.opt
Altera	*install_dir*/lib/altera/altera_par.tcl
Xilinx	*install_dir*/lib/xilinx/xilinx_par.opt[1] *install_dir*/lib/xilinx/xilinx_par5.opt[2]

1. Used with Xilinx Virtex-4 and eariler devices
2. Used with Xilinx Spartan-6 and Virtex-5 and later devices

For Altera and Xilinx designs, refer to the information in *Specifying Altera Place-and-Route Options*, on page 308 or *Specifying Xilinx Place-and-Route Options in a Tcl File*, on page 311 for details. For Xilinx designs, you can also override global placement options with an environment variable, as described in *Specifying Xilinx Global Placement Options*, on page 318.

– For Xilinx designs, you can run the SmartGuide flow. Enable the Smart Guide option.

– If you are going to run Synplify Premier physical synthesis, you can choose to backannotate data for Xilinx technologies. See *Backannotating Place-and-Route Data*, on page 326 for details.

– For Xilinx designs, you can automatically generate a coreloc file with backannotated data after place-and-route. See *Generating a Xilinx Coreloc Placement File*, on page 327 for more information.

– For physical synthesis with Altera Stratix IV GT, Stratix IV, Stratix III, Stratix II GX, or Stratix II devices you can select the Use placement constraints from physical synthesis. See *Forward Annotating Altera Constraints*, on page 323.

4. Enable the Run Place & Route following synthesis option. Click OK if you are not setting the options described in the next step.

   This creates the place-and-route implementation under the current synthesis implementation. Currently, you cannot change the location of the P&R directory.

   Conversely, if you do not want to create a place-and-route implementation, disable the Run Place & Route following synthesis option.

5. For Xilinx and Altera technologies, you can also do the following:
   - Specify a place-and-route options file. If you do not specify one, the tool automatically uses the default options.

     You can change or override the default options. See *Specifying Altera Place-and-Route Options*, on page 308 and *Specifying Xilinx Place-and-Route Options in a Tcl File*, on page 311 for details.
   - Backannotate constraints to the P&R tool. See *Backannotating Place-and-Route Data*, on page 326 for details.
   - Forward-annotate constraints from the P&R tool. See *Forward Annotating Altera Constraints*, on page 323.
   - Click OK.
6. Select the implementation in the Project view to see the place-and-route implementation.

   To create subsequent place-and-route implementations, select the place-and-route implementation, right-click, and select Add Place & Route Job. You can repeat the preceding steps to add as many P&R implementations as you need.

7. Synthesize the design.
   - Enable the P&R implementation you want to use, if you have not already done so (Implementation Options->Place and Route tab).

Copyright © 2013 Synopsys, Inc.

– Click the Run button, or right-click in the Project view and select Run Place & Route Job from the popup menu.

If the synthesis implementation associated with the place-and-route implementation has not been synthesized, Run Place & Route Job invokes synthesis as well. After synthesis, the place-and-route tool is automatically run. If you have a Xilinx design and have specified an options file, the software uses these options during place-and-route.

8. To run in batch mode, do this:
   – Create a place-and-route implementation, as described previously.
   – Use the -run all command. If the synthesis implementation is selected the software only runs synthesis; you must run place-and-route separately. Otherwise, make the current implementation the place-and-route implementation before issuing the batch command.

Specifying Altera Place-and-Route Options

This section shows you how to customize your Altera place-and-route run by specifying a place-and-route options file or `tcl` script. You can use either the default file or create a custom file.

1. To use the default place-and-route options, click the Add P&R Implementation button in the Project view and select Standard Options File in the dialog box. Click OK.

The software uses the options in the altera_par.tcl file which is located in the installation directory.

2. To use an existing options file (.tcl script):
   - Click the Add P&R Implementation button in the Project view.
   - Click Existing Options File. Select the file name in the next dialog box, and click Open.
   - Return to the Add New Place & Route Task dialog box and make sure the correct options file is selected. Click OK.
3. To create a new place-and-route options file:
   - Click the Add P&R Implementation button in the Project view. In the dialog box, click Create New Options File. Specify the file name in the next dialog box, and click OK.

A text window opens with the default options file. This file is automatically added to the project.

   - Edit the default options to customize this options file. For more information about the contents of this file, see *Options in the Altera Place-and-Route Options File*, on page 310.
   - Save the file.
   - Return to the Add New Place & Route Task dialog box, and make sure the options file you created is selected.

Copyright © 2013 Synopsys, Inc.

– Select Run Place & Route following synthesis. Click OK.

The software uses the options file to place and route the design after synthesis.

4. View the results.
   – Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
   – View the log file quartus.log for information about the run.

Options in the Altera Place-and-Route Options File

To customize the Altera place-and-route options file, you can edit the default options file (altera_par.tcl). This file contains the options for the following place-and-route processes:

- Fitter Options
- Timing Analyzer Options
- Analysis & Synthesis Options

Fitter Options

Edit the following default fitter options for the Quartus process shown below.

```
#Fitter Options
set_global_assignment -name OPTIMIZE_HOLD_TIMING "IO PATHS AND MINIMUM TPD PATHS"
set_global_assignment -name OPTIMIZE_FAST_CORNER_TIMING OFF
set_global_assignment -name OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING ON
set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_RETIMING OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING OFF
set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT NORMAL
set_global_assignment -name FITTER_EFFORT "AUTO FIT"
set_global_assignment -name SEED 1
```

Timing Analyzer Options

Edit the following default timing analyzer options for the Quartus process shown below.

```
#Timing Analyzer Options
set_global_assignment -name DO_MIN_TIMING OFF
set_global_assignment -name REPORT_IO_PATHS_SEPARATELY OFF
set_global_assignment -name CUT_OFF_PATHS_BETWEEN_CLOCK_DOMAINS ON
set_global_assignment -name CUT_OFF_READ_DURING_WRITE_PATHS ON
set_global_assignment -name CUT_OFF_IO_PIN_FEEDBACK ON
set_global_assignment -name DO_COMBINED_ANALYSIS OFF
set_global_assignment -name ANALYZE_LATCHES_AS_SYNCHRONOUS_ELEMENTS ON
set_global_assignment -name ENABLE_RECOVERY_REMOVAL_ANALYSIS OFF
set_global_assignment -name ENABLE_CLOCK_LATENCY OFF
```

Analysis & Synthesis Options

Edit the following default analysis and synthesis options for the Quartus process shown below.

```
#Analysis & Synthesis Options
set_global_assignment -name CYCLONE_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name CYCLONEII_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name STRATIX_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name MAXII_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name APEX20K_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name STRATIXII_OPTIMIZATION_TECHNIQUE BALANCED
set_global_assignment -name TRUE_WYSIWYG_FLOW OFF
set_global_assignment -name ADV_NETLIST_OPT_SYNTH_GATE_RETIME OFF
set_global_assignment -name ADV_NETLIST_OPT_RETIME_CORE_AND_IO ON
```

Specifying Xilinx Place-and-Route Options in a Tcl File

The following procedure shows you how to customize your Xilinx place-and-route run by specifying an ISE place-and-route Tcl file. You can also use a Tcl file to specify options for Vivado. For information about specifying Vivado place-and-route options in a Tcl file, see *Customizing Vivado Place and Route Options*, on page 860.

For the ISE options, you can use the default Tcl file or create a custom file. This file uses the Xilinx xtclsh flow, which is the recommended methodology. If you must use the old Xilinx xflow, refer to the procedure in *Specifying Xilinx Place-and-Route Options in an opt File*, on page 314.

1. To use the default place-and-route options, do the following:
   - Click the Add P&R Implementation button in the Project view and select Standard Options File in the dialog box.
   - Click OK.

   By default, the software uses the Tcl file located in the installation directory. This file is used by the Xilinx xtclsh executable to run the P&R tool.

2. To use an existing Tcl options file, do the following:

Copyright © 2013 Synopsys, Inc.

- Click the Add P&R Implementation button in the Project view.
- Click Existing Options File. Select the file you want, and click Open.
- Right-click the implementation, select Add New Place & Route Job, and make sure the correct options file is selected. Click OK.

3. To create a new place-and-route Tcl file, do this:
   - Click the Add P&R Implementation button in the Project view.
   - Click Create New Options File. Specify a file name and click Open. The tool generates a default Tcl file and automatically adds it to the project.

   A text window opens with the default Tcl file options.

4. Edit the file.
   - Edit the default options to customize this file.
   - Save the file.

5. Synthesize, place, and route the design.
   - Right-click the implementation, select Add New Place & Route Job, and make sure the Tcl file is selected.
   - Select Run Place & Route following synthesis. Click OK.

   The software uses the Tcl file to place and route the design after synthesis.

6. View the results.
   - Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
   - View the log file xflow.log for information about the run.

Sample run_ise.tcl File

The following example shows the place-and-route options file used in the Xilinx xtclsh (run_ise.tcl) flow.

```
##################################################
###  SET DESIGN VARIABLES                      ###
##################################################
set DesignName "adder"
set FamilyName "VIRTEX6"
set DeviceName XC6VLX760"
set PackageName FF1760"
```

```
set SpeedGrade "-1"
set TopModule ""
set EdifFile "D:/adder/adder.edf"

##################################################
###  SET FLOW                                  ###
##################################################
set Flow "Standard"

##################################################
###  SET POWER OPTION                          ###
##################################################
set Power "0"

##################################################
###  PROJECT SETUP                             ###
##################################################
if {![file exists $DesignName.xise]} {

   project new $DesignName.xise
   project set family $FamilyName
   project set device $DeviceName
   project set package $PackageName
   project set speed $SpeedGrade
   xfile add $EdifFile

   if {[file exists synplicity.ucf]} { xfile add synplicity.ucf }

} else {

   project open $DesignName.xise}

##################################################
###  STANDARD                                  ###
##################################################
if { $Flow == "Standard" } {

   project set "Netlist Translation Type" "Timestamp"
   project set "Other NGDBuild Command Line Options" "-verbose"
   project set "Generate Detailed MAP Report" TRUE
   project set {Place & Route Effort Level (Overall)} "High"}

##################################################
###  FAST                                      ###
##################################################
if { $Flow == "Fast" } {

   project set "Netlist Translation Type" "Timestamp"
   project set "Other NGDBuild Command Line Options" "-verbose"
   project set "Generate Detailed MAP Report" TRUE
   project set {Place & Route Effort Level (Overall)} "Standard"}

##################################################
###  SMARTGUIDE                                ###
##################################################
if { $Flow == "SmartGuide" } {
```


Copyright © 2013 Synopsys, Inc.

```
    project set "Use Smartguide" TRUE
    project set "SmartGuide Filename" $DesignName\_guide.ncd
    project set "Netlist Translation Type" "Timestamp"
    project set "Other NGDBuild Command Line Options" "-verbose"
    project set "Generate Detailed MAP Report" TRUE
    project set {Place & Route Effort Level (Overall)} "High"}

##################################################
###  SMARTGUIDE FAST                           ###
##################################################
if { $Flow == "SmartGuideFast" } {

    project set "Use Smartguide" TRUE
    project set "SmartGuide Filename" $DesignName\_guide.ncd
    project set "Netlist Translation Type""Timestamp"
    project set "Other NGDBuild Command Line Options" "-verbose"
    project set "Generate Detailed MAP Report" TRUE
    project set {Place & Route Effort Level (Overall)} "Standard"}

##################################################
###  EXECUTE ISE PLACE & ROUTE                 ###
##################################################
file delete -force $DesignName\_xdb
project open $DesignName.xise
process run "Implement Design" -force rerun_all
## process run      "Generate Programming File"

##################################################
###  EXECUTE POWER OPTION                      ###
##################################################
if { $Power == "1" } {

exec xpwr -v $DesignName.ncd $DesignName.pcf}

project close
```

Specifying Xilinx Place-and-Route Options in an opt File

If you are using the old Xilinx xflow, you specify P&R options in a `opt` file instead of a Tcl file, which is the recommended methodology (see *Specifying Xilinx Place-and-Route Options in a Tcl File*, on page 311.)

1. To use the default place-and-route options, do the following:
   - Click the Add P&R Implementation button in the Project view and select Standard Options File in the dialog box.
   - Click OK.

   By default, the software uses the `opt` file located in the installation directory, which is used by the xflow executable to run the P&R tool.

2. To use an existing options file, do the following:
   - Click Implementation Options, go to the Device tab, and enable Use Xilinx Xflow. By default, the synthesis tools use the xtclsh executable and the corresponding Tcl options file, so you must explicitly turn on the Use Xilinx Xflow option to use Xflow.
   - Click the Add P&R Implementation button in the Project view.
   - Click Existing Options File. Select the `opt` file you want to use and click Open.
   - Select the implementation in the project view, right-click and select Add Place & Route Job. In the dialog box, check that the correct options file is selected. Click OK.
3. To create a new place-and-route options file, do this:
   - Click the Add P&R Implementation button in the Project view.
   - Select File->New. Set the file type to Xilinx Option File, type a file name., enable the Add to Project option, and click OK. This file is automatically added to the project.

   A text window opens with the options file.

4. Edit the file.
   - Edit the default options to customize this options file. For more information about the contents of this file, see *Options in the Xilinx Place-and-Route Options File*, on page 316.
   - Save the file.
5. Synthesize, place, and route the design.
   - Return to the Add New Place & Route Task dialog box, and make sure the options file you created is selected.
   - Select Run Place & Route following synthesis. Click OK.

Copyright © 2013 Synopsys, Inc.

The software uses the options file to place and route the design after synthesis.

6. View the results.
   - Select the P&R implementation in the Project view. The result files are displayed in the Implementation Results view.
   - View the log file xflow.log for information about the run.

Options in the Xilinx Place-and-Route Options File

To customize the Xilinx place-and-route options file, you can edit the default options file (xilinx_par.opt or xilinx_par5.opt). This file contains the options for the following place-and-route processes:

- Translator Options
- Mapper Options
- Place-and-Route Options
- Post Place-and-Route Timing Report Options
- Bitgen Generation Options

Translator Options

Edit the following default translator options for the ngdbuild command as shown below.

```
#########################
## Translator Options ##
#########################
## Type "ngdbuild -h" for a detailed list of ngdbuild command line options
#########################
Program ngdbuild
-intstyle xflow; # Message Reporting Style: ise, xflow, or silent
-nt timestamp;# NGO File generation. Regenerate only when
   # source netlist is newer than existing
   # NGO file (default)
<userdesign>; # User design - pick from xflow command line
<design>.ngd; # Name of NGD file. Filebase same as design filebase
##-p <partname>;# Partname to use - picked from xflow command line
##-sd source_dir; #Add "source_dir" to the list of directories
## #to search when resolving netlist file references
-uc synplicity.ucf; #Use specified "User Constraint File".
```

```
## #The file <design_name>.ucf is used by default
## #if it is found in the local directory.
##-insert_keep_hierarchy; # Retain hierarchy identified by
   individual source input netlists
End Program ngdbuild
```

Mapper Options

Edit the following default mapper options for the map command as shown below.

```
####################
## Mapper Options ##
####################
## Type "map -h <architecture>" for a detailed list of map command
line options
####################
Program map
-intstyle xflow; # Message Reporting Style: ise, xflow, or silent
#-cm area; # Cover mode.
#-pr b; # Pack internal flops/latches into input(i), output (o),
   # or both (b) types of IOBs.
#-k 4; # Function size for covering combinational logic.
#-c 100; # Pack unrelated logic into clbs.
-t 1; # Timing-driven cost table entry.
-w;                      # overwrite existing ncd file
-o <design>_map.ncd;     # Output Mapped ncd file
<inputdir><design>.ngd;  # Input NGD file
<inputdir><design>.pcf;  # Physical constraints file
END Program map
```

Note that in the xilinx_par5.opt file for Virtex-5 devices and later, the following option is set:

```
-t 1     # Timing-driven cost table entry.
```

Place-and-Route Options

Edit the following default place-and-route options for the par command as shown below.

```
###########################
## Place & Route Options ##
###########################
## Type "par -h" for a detailed list of par command line options
###########################
Program par
-w; # Overwrite existing placed and routed ncd
-intstyle xflow; # Message Reporting Style: ise, xflow, or silent
-ol high; # Overall effort level
#-t 1; # Placer cost table entry.
```


Copyright © 2013 Synopsys, Inc.

```
#-xe c;
<design>_map.ncd;         # Input mapped NCD file
<inputdir><design>.ncd;   # Output placed and routed NCD
<inputdir><design>.pcf;   # Input physical constraints file
END Program par
```

Post Place-and-Route Timing Report Options

Edit the following default post place-and-route timing report options for the post_par_trce command as shown below.

```
####################################
## Post PAR Timing Report Options ##
####################################
## Type trce -h for detailed list of trce command line options
####################################
Program post_par_trce
-intstyle xflow;
-e 100;
<inputdir><design>.ncd;
<inputdir><design>.pcf;
-o <design>.twr;
END Program post_par_trce
```

Bitgen Generation Options

Edit the following default bit generation options for the bitgen command as shown below.

```
#######################
## Bitgen Generation ##
#######################
## Type "bitgen -h" for detailed list of bitgen command line options
## Uncomment following commands to generate a bitstream
#######################
##Program bitgen
##-intstyle xflow;
##<inputdir><design>.ncd;
##-l; # Create logic allocation file
##-w; # Overwrite existing output file
##-m; # Create mask file
##<inputdir><design>.bit;
##<inputdir><design>.pcf;
##END Program bitgen
```

Specifying Xilinx Global Placement Options

For graph-based physical synthesis in Xilinx designs, you can override the default values for global placement in the options file.

1. Create a file with your preferred global placement options.

2. Use the SYN_XILINX_GLOBAL_PLACE_OPT environment variable to point to your options file:

```
SYN_XILINX_GLOBAL_PLACE_OPT ="C:/Temp/test.opt"
```


Copyright © 2013 Synopsys, Inc.

Setting Constraints for Physical Synthesis

The following describe how to set up various constraints for physical synthesis. Some procedures are technology-specific or may apply only if you are using Design Planner.

- Using Design Planner Floorplan Constraints, on page 320
- Setting Physical Synthesis Constraints for Altera, on page 322

For additional information on translating Altera QSF and Xilinx UCF constraints to the synthesis sdc format, see *Translating Altera QSF Constraints*, on page 193 and *Converting and Using Xilinx UCF Constraints*, on page 200, respectively.

Using Design Planner Floorplan Constraints

For physical synthesis flows that use Design Planner (Altera and Xilinx technologies), use the floorplan constraints you generated to drive synthesis. You can use the following procedure before logic synthesis, because you only need a compiled design.

1. Start with a compiled design.

   The Design Planner flows are only supported for Altera and Xilinx technologies.

2. Floorplan the design with Design Planner, and assign RTL modules, paths or components to regions on the device. See Chapter 12, *Floorplanning with Design Planner* for details.

3. Add the floorplan file to the design.
   - After floorplanning, add the design plan file (`sfp`) to the project.
   - Enable the file in the Implementation Options ->Design Planning tab.

4. Run synthesis.

   You can run the preliminary logic synthesis step of the physical synthesis flows, or you can run physical synthesis. When you run

physical synthesis, the placement constraints from the `sfp` file are honored.

Translating Pin Location Files

You can automatically convert Altera and Xilinx place-and-route pin location constraint files to SCOPE constraint files (`sdc`) with the Run->Translate Vendor IO command.

1. Select Run ->Translate Vendor IO.
2. Enter the name of the Altera `pin` or Xilinx `pad` file you want to translate.
3. Type a name for the `sdc` constraint file you want to create.

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

   The pin locations from the files are translated into SDC constraints. If you add the constraint file to the project, it can be used for design planning and synthesis. Note that if there are pin assignment conflicts between SDC constraints and pin locations assigned in Design Planner, the SDC constraints take precedence. For another method to assign pins, see *Assigning Pins Interactively*, on page 619.

Copyright © 2013 Synopsys, Inc.

Setting Physical Synthesis Constraints for Altera

The following procedure describes the general procedure for setting up physical synthesis constraints for Altera designs.

1. Create a constraints file, as described in *Using the SCOPE Editor (Legacy)*, on page 173, and add it to your project.

   Use timing constraints to specify performance goals for the design and describe the environment.

2. Translate Altera QSF constraints, as described in *Translating Altera QSF Constraints*, on page 193.

3. Select Run->Constraint Check to check the constraints in the constraints file. See *Tcl Syntax Guidelines for Constraint Files*, on page 123.

   This command generates a report that checks the syntax and applicability of the timing constraints in the `sdc` files for your project. The report is written to the *project_name*_cck.rpt file.

4. If you are using Design Planner, create a floorplan file and add it to your project. See *Using Design Planner Floorplan Constraints*, on page 320 for details.

Forward-Annotating Physical Synthesis Constraints

For more information about forward-annotating Altera physical constraints, see:

- Forward Annotating Altera Constraints
- Limitations Using the Physical Analyst and Technology View

Forward Annotating Altera Constraints

You can choose to forward annotate physical constraints from the Synplify Premier physical synthesis tool, or else, let the Quartus II place-and-route tool determine how to handle the physical constraints. Forward-annotation is only available for Altera Stratix IV GT, Stratix IV, Stratix III, Stratix II GX, or Stratix II devices. Use the following procedure.

New Implementation

Do the following for a new implementation:

1. Enable the Physical Synthesis switch.

   If the switch is disabled, physical synthesis optimizations are performed but placement constraints will not be forward annotated.

2. Click on the Add P&R Implementation button from the Project view or right-click and select Add Place & Route Task from the popup menu to create a new P&R implementation.

3. On the Add New Place & Route Task dialog box, enable or disable the Use placement constraints from physical synthesis option. By default, this option is enabled.

   The Physical Synthesis switch must be enabled to use this option. When this option is disabled, physical synthesis optimizations are performed but placement constraints will not be forward annotated.

Copyright © 2013 Synopsys, Inc.

Existing Implementation

Do the following for an existing place-and-route implementation:

1. Enable the Physical Synthesis switch.

   If the switch is disabled, physical synthesis optimizations are performed but placement constraints will not be forward annotated.

2. In the Project view, select the place-and-route implementation, then right-click and select P&R Options from the popup menu.

3. Enable or disable the Use placement constraints from physical synthesis option from the popup dialog box. By default, this option is enabled.

Note: When implementing physical synthesis for Altera designs, only certain placement constraints such as RAMs, DSPs and critical paths are forward-annotated for placement and routing. See *Limitations Using the Physical Analyst and Technology View*, on page 325

Limitations Using the Physical Analyst and Technology View

Since all Altera placement constraints are *not* forward-annotated after running physical synthesis, the instance locations in the Quartus II place-and-route output file (tan.rpt) might differ from locations in the Physical Analyst and HDL Technology views. The place-and-route tool matches locations in the physical constraints file (plc.tcl), for which the Synplify Premier tool forward-annotates only RAMS, DSPs, and critical paths.

Copyright © 2013 Synopsys, Inc.

Backannotating Physical Synthesis Constraints

For more accurate results, you can resynthesize your design using place-and-route data from the back-end tools. Refer to the following topics:

- Backannotating Place-and-Route Data
- Generating a Xilinx Coreloc Placement File

Backannotating Place-and-Route Data

You can also choose to back annotate place-and-route data which provides accurate timing and placement information during physical synthesis. However, this option is only applicable for Xilinx technologies.

Use the following procedure to backannotate place-and-route data. Do not backannotate designs that contain IP cores.

New Implementation

Do the following for a new place-and-route implementation:

1. Click on the Add P&R Implementation button from the Project view.
2. On the Add New Place & Route Task dialog box, enable the Run Backannotation after Place & Route option.

Existing Implementation

Do the following for an existing place-and-route implementation:

1. In the Project view, select the place-and-route implementation, then right-click and select P&R Options from the popup menu.
2. Enable the Smart Guide option from the popup dialog box to run the SmartGuide flow.
3. Enable the Run Backannotation after Place & Route option from the popup dialog box.

Copyright © 2013 Synopsys, Inc.

Generating a Xilinx Coreloc Placement File

You can use the Synplify Premier tool to generate a coreloc constraint file that contains the placement for all “anchor” or “core” instances in the design, like I/Os and BUFs. You can pass this file to the P&R tool for backannotation.

Using core location constraints ensures that block and I/O placement are equivalent when you compare logic and physical synthesis. Include the `coreloc.sdc` file to both logic and physical synthesis runs. This placement information is also very useful for timing convergence. By stabilizing block and I/O locations, you eliminate any changes to results that stem from placement variations when running small designs.

Copyright © 2013 Synopsys, Inc.

To generate a coreloc file, do the following:

1. Before you run synthesis, make sure to enable the Run Backannotation after Place & Route switch when you create the place-and-route implementation before running synthesis.

   The *filename*_coreloc.sdc file is generated during place-and-route backannotation and is written to the results directory. The *filename* is the same base name as the EDIF output netlist (*filename*.edf).

2. After the synthesis run, when the file has been generated, add the *filename*_coreloc.sdc file to your project.

   To guarantee consistent and stable comparisons, include the coreloc.sdc file to both logic and physical synthesis runs.

3. Re-run synthesis.

Copyright © 2013 Synopsys, Inc.

CHAPTER 7

Inferring High-Level Objects

This chapter contains guidelines on how to structure your code or attach attributes so that the synthesis tools can automatically infer high-level objects like RAMs. See the following for more information:

- Defining Black Boxes for Synthesis, on page 330
- Defining State Machines for Synthesis, on page 340
- Implementing High-Reliability Designs, on page 345
- Inferring RAMs, on page 369
- Initializing RAMs, on page 394
- Inferring Shift Registers, on page 405
- Working with LPMs, on page 411

Copyright © 2013 Synopsys, Inc.

Defining Black Boxes for Synthesis

Black boxes are predefined components for which the interface is specified, but whose internal architectural statements are ignored. They are used as place holders for IP blocks, legacy designs, or a design under development.

This section discusses the following topics:

- Instantiating Black Boxes and I/Os in Verilog, on page 330
- Instantiating Black Boxes and I/Os in VHDL, on page 332
- Adding Black Box Timing Constraints, on page 334
- Adding Other Black Box Attributes, on page 338

The Clock Conversion option is only available in the Synplify Pro and Synplify Premier tools. For information about using black boxes with the Clock Conversion option, see *Working with Gated Clocks, on page 466*.

Instantiating Black Boxes and I/Os in Verilog

Verilog black boxes for macros and I/Os come from two sources: commonly-used or vendor-specific components that are predefined in Verilog macro libraries, or black boxes that are defined in another input source like a schematic. For information about instantiating black boxes in VHDL, see Instantiating Black Boxes and I/Os in VHDL, on page 332. Additional information about black boxes can be found in *Working with Gated Clocks, on page 466*, Instantiating CoreGen Cores, on page 801, and Instantiating Virtex PCI Cores, on page 802.

The following process shows you how to instantiate both types as black boxes. Refer to the *installDirectory*/examples directory for examples of instantiations of low-level resources.

1. To instantiate a predefined Verilog module as a black box:
   - Select the library file with the macro you need from the *installDirectory*/lib/*technology* directory. Files are named *technology*.v. Most vendor architectures provide macro libraries that predefine the black boxes for primitives and macros.
   - Make sure the library macro file is the first file in the source file list for your project.

2. To instantiate a module that has been defined in another input source as a black box:
   - Create an empty macro that only contains ports and port directions.
   - Put the syn_black_box synthesis directive just before the semicolon in the module declaration.

```
module myram (out, in, addr, we) /* synthesis syn_black_box */;
   output [15:0] out;
   input [15:0] in;
   input [4:0] addr;
   input we;
endmodule
```

   - Make an instance of the stub in your design.
   - Compile the stub along with the module containing the instantiation of the stub.
   - To simulate with a Verilog simulator, you must have a functional description of the black box. To make sure the synthesis software ignores the functional description and treats it as a black box, use the translate_off and translate_on constructs. For example:

```
module adder8(cout, sum, a, b, cin);
// Code that you want to synthesize
/* synthesis translate_off */
// Functional description.
/* synthesis translate_on */
// Other code that you want to synthesize.
endmodule
```

3. To instantiate a vendor-specific (black box) I/O that has been defined in another input source:
   - Create an empty macro that only contains ports and port directions.
   - Put the syn_black_box synthesis directive just before the semicolon in the module declaration.
   - Specify the external pad pin with the black_box_pad_pin directive, as in this example:

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
      /* synthesis syn_black_box black_box_pad_pin="PAD"
```

   - Make an instance of the stub in your design.

Copyright © 2013 Synopsys, Inc.

- Compile the stub along with the module containing the instantiation of the stub.

4. Add timing constraints and attributes as needed. See *Adding Black Box Timing Constraints*, on page 334 and *Adding Other Black Box Attributes*, on page 338.

5. After synthesis, merge the black box netlist and the synthesis results file using the method specified by your vendor.

Instantiating Black Boxes and I/Os in VHDL

VHDL black boxes for macros and I/Os come from two sources: commonly-used or vendor-specific components that are predefined in VHDL macro libraries, or black boxes that are defined in another input source like a schematic. For information about instantiating black boxes in VHDL, see *Instantiating Black Boxes and I/Os in Verilog*, on page 330.

The following process shows you how to instantiate both types as black boxes. Refer to the *installDirectory*/examples directory for examples of instantiations of low-level resources.

1. To instantiate a predefined VHDL macro (for a component or an I/O),
   - Select the library file with the macro you need from the *installDirectory*/lib/*vendor* directory. Files are named *family*.vhd. Most vendor architectures provide macro libraries that predefine the black boxes for primitives and macros.
   - Add the appropriate library and use clauses to the beginning of your design units that instantiate the macros.

     ```
     library family ;
     use family.components.all;
     ```

2. To create a black box for a component from another input source:
   - Create a component declaration for the black box.
   - Declare the syn_black_box attribute as a boolean attribute.
   - Set the attribute to true.

```
library synplify;
use synplify.attributes.all;
entity top is
   port (clk, rst, en, data: in bit; q: out bit);
end top;

architecture structural of top is
component bbox
   port(Q: out bit; D, C, CLR: in bit);
end component;

attribute syn_black_box of bbox: component is true;
...
```

– Instantiate the black box and connect the ports.

```
begin
my_bbox: bbox port map (
   Q => q,
   D => data,
   C => clk,
   CLR => rst);
```

– To simulate with a VHDL simulator, you must have the functional description of a black box. To make sure the synthesis software ignores the functional description and treats it as a black box, use the translate_off and translate_on constructs. For example:

```
architecture behave of ram4 is
begin
   -- synthesis translate_off
      stimulus: process (clk, a, b)
   -- Functional description
end process;
   -- synthesis translate_on

-- Other source code you WANT synthesized
```

3. To create a vendor-specific (black box) I/O for an I/O defined in another input source:
   – Create a component declaration for the I/O.
   – Declare the black_box_pad_pin attribute as a string attribute.
   – Set the attribute value on the component to be the external pin name for the pad.

Copyright © 2013 Synopsys, Inc.

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

component mybuf
   port(O: out bit; I: in bit);
end component;
attribute black_box_pad_pin of mybuf: component is "I";
```

– Instantiate the pad and connect the signals.

```
begin
data_pad: mybuf port map (
   O => data_core,
   I => data);
```

4. Add timing constraints and attributes. See Adding Black Box Timing Constraints, on page 334, *Using Gated Clocks for Black Boxes*, on page 489, and Adding Other Black Box Attributes, on page 338.

Adding Black Box Timing Constraints

A black box does not provide the software with any information about internal timing characteristics. You must characterize black box timing accurately, because it can critically affect the overall timing of the design. To do this, you add constraints in the source code or in the SCOPE interface.

You attach black box timing constraints to instances that have been defined as black boxes. There are three black box timing constraints, syn_tpd, syn_tsu, and syn_tco. There are additional attributes for black box pins and black boxes with gated clocks; see Adding Other Black Box Attributes, on page 338 and *Using Gated Clocks for Black Boxes*, on page 489.

1. Define the instance as a black box, as described in *Instantiating Black Boxes and I/Os in Verilog*, on page 330 or *Instantiating Black Boxes and I/Os in VHDL*, on page 332.

2. Determine the kind of constraint for the information you want to specify:

To define...	Use...
Propagation delay through the black box	syn_tpd
Setup delay (relative to the clock) for input pins	syn_tsu
Clock-to-output delay through the black box	syn_tco

3. In VHDL, use the following syntax for the constraints.
   - Use the predefined attributes package by adding this syntax

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

     In VHDL, you must use the predefined attributes package. For each directive, there are ten predeclared constraints in the attributes package, from *directive_name1* to *directive_name10*. If you need more constraints, declare the additional constraints using integers greater than 10. For example:

     ```
     attribute syn_tco11 : string;
     attribute syn_tco12 : string;
     ```

   - Define the constraints in either of these ways:

VHDL syntax	`attribute attributeName<n> : "att_value"`
Verilog-style notation	`attribute attributeName<n> of bbox_name : component is "att_value"`

The following table shows the appropriate syntax for att_value. See the *Reference Manual* for complete syntax information.

Copyright © 2013 Synopsys, Inc.

Attribute	Value Syntax
`syn_tsu<n>`	*bundle* **->** [**!**]*clock* **=** *value*
`syn_tco<n>`	[**!**]*clock* **->** *bundle* **=** *value*
`syn_tpd<n>`	*bundle* **->** *bundle* **=** *value*

- *<n>* is a numerical suffix.
- *bundle* is a comma-separated list of buses and scalar signals, with no intervening spaces. For example, A,B,C.
- ! indicates (optionally) a negative edge for a clock.
- *value* is in ns.

The following is an example of black box attributes, using VHDL signal notation:

```
architecture top of top is
component rcf16x4z port(
   ad0, ad1, ad2, ad3 : in std_logic;
   di0, di1, di2, di3 : in std_logic;
   wren, wpe : in std_logic;
   tri : in std_logic;
   do0, do1, do2 do3 : out std_logic;
end component

attribute syn_tpd1 of rcf16x4z : component is
   "ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is
   "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcf16x4z : component is
   "ad0,ad1,ad2,ad3 -> ck = 1.2";
attribute syn_tsu2 of rcf16x4z : component is
   "wren,wpe,do0,do1,do2,do3 -> ck = 0.0";
```

4. In Verilog, add the directives as comments, as shown in the following example. For explanations about the syntax, see the table in the previous step or the *Reference Manual*.

```
module ram32x4 (z, d, addr, we, clk)
   /* synthesis syn_black_box
   syn_tpd1="addr[3:0]->z[3:0]=8.0"
   syn_tsu1="addr[3:0]->clk=2.0"
   syn_tsu2="we->clk=3.0" */;
output [3:0[ z;
```

```
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

5. To add black box attributes through the SCOPE interface, do the following:
   - Open the SCOPE spreadsheet and select the Attributes panel.
   - In the Object column, select the name of the black-box module or component declaration from the pull-down list. Manually prefix the black box name with **v:** to apply the constraint to the view.
   - In the Attribute column, type the name of the timing attribute, followed by the numerical suffix, as shown in the following table. You cannot select timing attributes from the pull-down list.
   - In the Value column, type the appropriate value syntax, as shown in the table in step 3.
   - Save the constraint file, and add it to the project.

   The resulting constraint file contains syntax like this:

   define_attribute v:{*blackboxModule***}** *attribute<n>* **{***attributeValue***}**

6. Synthesize the design, and check black box timing.

Copyright © 2013 Synopsys, Inc.

Adding Other Black Box Attributes

Besides black box timing constraints, you can also add other attributes to define pin types on the black box or define gated clocks. You cannot use the attributes for all technologies. Check the *Reference Manual* for details about which technologies are supported. For information about black boxes with gated clocks, see *Using Gated Clocks for Black Boxes*, on page 489. The Clock Conversion option is only available in the Synplify Pro and Synplify Premier tools.

1. To specify that a clock pin on the black box has access to global clock routing resources, use syn_isclock.

   Depending on the technology, different clock resources are inserted. In Xilinx, the software inserts BUFG, for Microsemi it inserts CLKBUF, and for QuickLogic, it inserts Q_CKPAD.

2. To specify that the software need not insert a pad for a black box pin, use black_box_pad_pin.

   Use this for technologies that automatically insert pad buffers for the I/Os like Xilinx, some Altera families, Microsemi, QuickLogic, and some Lattice technologies.

3. To define a tristate pin so that you do not get a mixed driver error when there is another tristate buffer driving the same net, use black_box_tri_pins.

Copyright © 2013 Synopsys, Inc.

4. To ensure consistency between synthesized black box netlist names and the names generated by third party tools or IP cores, use the following attributes (Xilinx only):
   - syn_edif_bit_format
   - syn_edif_scalar_format
5. To specify that a port on a black box is connected to an internal STARTUP block in Xilinx XC4000 architectures, use the xc_isgr directive.

Copyright © 2013 Synopsys, Inc.

Defining State Machines for Synthesis

A finite state machine (FSM) is a piece of hardware that advances from state to state at a clock edge. The synthesis software recognizes and extracts the state machines from the HDL source code. For guidelines on setting up the source code, see the following:

- Defining State Machines in Verilog, on page 340
- Defining State Machines in VHDL, on page 341
- Specifying FSMs with Attributes and Directives, on page 343

For information about the attributes used to define state machines, see *Running the FSM Compiler*, on page 455. For information about implementing safe FSMs, see *Specifying Safe FSMs*, on page 359.

Defining State Machines in Verilog

The synthesis software recognizes and automatically extracts state machines from the Verilog source code if you follow these coding guidelines. The software attaches the syn_state_machine attribute to each extracted FSM.

For alternative ways to define state machines, see *Defining State Machines in VHDL*, on page 341 and *Specifying FSMs with Attributes and Directives*, on page 343.

- In Verilog, model the state machine with case, casex, or casez statements in always blocks. Check the current state to advance to the next state and then set output values. Do not use if statements.
- Always use a default assignment as the last assignment in the case statement, and set the state variable to 'bx. This is a "don't care" statement and ensures that the software can remove unnecessary decoding and gates.
- Make sure the state machines have a synchronous or asynchronous reset to set the hardware to a valid state after power-up, or to reset the hardware when you are operating.

Copyright © 2013 Synopsys, Inc.

- Use explicit state values for states using parameter or ‘define statements. This is an example of a parameter statement that sets the current state to 2’h2:

  ```
  parameter state1 = 2’h1, state2 = 2’h2;
  ...
  current_state = state2;
  ```

  This example shows how to set the current state value with `define statements:

  ```
  ‘define state1 2’h1
  ‘define state2 2’h2
  ...
  current_state = ‘state2;
  ```

  Make state assignments using parameter with symbolic state names.Use parameter over `define, because `define is applied globally whereas parameter definitions are local. Local definitions make it easier to reuse certain state names in multiple FSM designs. For example, you might want to reuse common state names like RESET, IDLE, READY, READ, WRITE, ERROR, and DONE. If you use `define to assign state names, you cannot reuse a state name because the name has already been taken in the global name space. To use the names multiple times, you have to `undef state names between modules and redefine them with `define state names in the new FSM modules. This method makes it difficult to probe the internal values of FSM state buses from a testbench and compare them to the state names.

Defining State Machines in VHDL

The synthesis software recognizes and automatically extracts state machines from the VHDL source code if you follow coding guidelines. For alternative ways to define state machines, see *Defining State Machines in Verilog*, on page 340 and *Specifying FSMs with Attributes and Directives*, on page 343.

The following are VHDL guidelines for coding. The software attaches the syn_state_machine attribute to each extracted FSM.

- Use case statements to check the current state at the clock edge, advance to the next state, and set output values. You can also use if-then-else statements, but case statements are preferable.

Copyright © 2013 Synopsys, Inc.

- If you do not cover all possible cases explicitly, include a when others assignment as the last assignment of the case statement, and set the state vector to some valid state.
- If you create implicit state machines with multiple `WAIT` statements, the software does not recognize them as state machines.
- Make sure the state machines have a synchronous or asynchronous reset to set the hardware to a valid state after power-up, or to reset the hardware when you are operating.
- To choose an encoding style, attach the syn_encoding attribute to the enumerated type. The software automatically encodes your state machine with the style you specified.

Specifying FSMs with Attributes and Directives

If your design has state machines, the software can extract them automatically with the FSM Compiler (see *Optimizing State Machines*, on page 454), or you can manually specify attributes to define the state machines. You attach the attributes to the state registers. For detailed information about the attributes and their syntax, see the *Reference Manual*.

The following steps show you how to use attributes to define FSMs for extraction. For alternative ways to define state machines, see Defining State Machines in Verilog, on page 340 and Defining State Machines in VHDL, on page 341.

1. To determine how state machines are extracted, set attributes in the source code as shown in the following table:

To...	Attribute
Specify a state machine for extraction and optimization	syn_state_machine=1
Prevent state machines from being extracted and optimized	syn_state_machine=0
Prevent the state machine from being optimized away	syn_preserve=1

   For information about how to add attributes, see *Specifying Attributes and Directives*, on page 245.

2. To determine the encoding style used for the state machine, set the syn_encoding attribute in the source code or in the SCOPE window. For VHDL users there are alternative methods, described in the next step.

   The FSM Compiler and the FSM Explorer honor this setting. The different values for this attribute are briefly described here:

Copyright © 2013 Synopsys, Inc.

Situation: If...	syn_encoding Value	Explanation
Area is important	sequential	One of the smallest encoding styles.
Speed is important	onehot	Usually the fastest style and suited to most FPGA styles.
Recovery from an invalid state is important	safe, with another style. For example: `/* synthesis syn_encoding = "safe, onehot" */`	Forces the state machine to reset. For example, where an alpha particle hit in a hostile operating environment causes a spontaneous register change, you can use safe to reset the state machine.
There are <5 states	sequential	Default encoding.
Large output decoder follows the FSM	sequential or gray	Could be faster than onehot, even though the value must be decoded to determine the state. For sequential, more than one bit can change at a time; for gray, only one bit changes at a time, but more than one bit can be hot.
There are a large number of flip-flops	onehot	Fastest style, because each state variable has one bit set, and only one bit of the state register changes at a time.

3. If you are using VHDL, you have two choices for defining encoding:
   - Use syn_encoding as described above, and enable the FSM compiler.
   - Use syn_enum_encoding to define the states (sequential, onehot, gray, and safe) and disable the FSM compiler. If you do not disable the FSM compiler, the syn_enum_encoding values are not implemented. This is because the FSM compiler, a mapper operation, overrides syn_enum_encoding, which is a compiler directive.

     Use this method for user-defined FSM encoding. For example:

     ```
     attribute syn_enum_encoding of state_type : type is "001 010 101";
     ```

Implementing High-Reliability Designs

As geometries shrink, the possibility of soft errors or radiation effects increase. This affects industries like aerospace most immediately, but many other applications increasingly require high reliability and built-in fault tolerance.

The synthesis software provides different ways to implement high reliability, and the following describe methods to implement high reliability in Altera and Xilinx designs:

- Implementing Distributed TMR, on page 346
- Implementing Duplication with Comparison (DWC), on page 350
- Using TMR or ECC for RAMs, on page 356
- Specifying Safe FSMs, on page 359
- Error Monitoring for High Reliability Features, on page 361

For Microsemi designs, see *Working with Radhard Designs*, on page 790 and *Specifying syn_radhardlevel in the Source Code*, on page 791 for information about implementing high reliability features.

About Triple Modular Redundancy (TMR)

One of the ways to ensure high reliability is to use triple modular redundancy or TMR. TMR is a method of ensuring fault tolerance by automatically adding redundancy to provide immunity for single event transient (SET) and single event upset (SEU) faults.

- A SET fault is a temporary change in output of combinational logic caused by ions or electro-magnetic radiation. This fault usually corrects itself within several nanoseconds, but can become more problematic at higher operating frequencies or if the fault propagates through the logic.
- A SEU fault is a change of state caused by ions or electro-magnetic radiation that affects sequential elements.

With TMR, three redundant systems execute a process and a voting system produces a single output that reflects the majority vote from the three systems. If any one system fails, the voter chooses the majority from the other two systems, so that the system avoids failure due to SET and SEU faults. The synthesis tools use attributes to implement TMR.

Copyright © 2013 Synopsys, Inc.

TMR can be local, distributed, or global, and the following table shows support for the different kinds of TMR in the synthesis tools. The synthesis tools also support Duplication with Comparison (DWC).

Local TMR	Microsemi Altera/Xilinx RAMs	*Working with Radhard Designs*, on page 790 *Specifying Local TMR for RAMs*, on page 356
Distributed TMR	Altera, Xilinx	*Implementing Distributed TMR*, on page 346
DWC	Altera, Xilinx	*Implementing Duplication with Comparison (DWC)*, on page 350
Global TMR	Microsemi	*Specifying syn_radhardlevel in the Source Code*, on page 791

Implementing Distributed TMR

Distributed TMR protects against SETs in combinatorial logic. It triplicates the sequential logic and the combinatorial logic, including the voters. It is primarily aimed at commercial/military-aerospace implementations, especially those that require detailed control over the area, speed, or reliability of specific blocks.

The following procedure specifies how to implement distributed TMR.

1. Select a supported technology family in the Synplify Premier tool.

   See *Vendor Support for Distributed TMR and DWC*, on page 351 for a list.

2. Enable the Enhanced Optimization option.

3. From the Attributes tab of the SCOPE editor, specify the syn_radhardlevel attribute with a value of distributed_tmr for the modules you want to triplicate.

   For the syn_radhardlevel attribute syntax, see *syn_radhardlevel*, on page 1462 in the *Reference Manual*.

   You must set the attribute on views or modules. You can specify hierarchical modules that require increased reliability to prevent SEU and SET faults. If the specified module contains an FSM, TMR is also applied to the FSM. In this case it could override a syn_fsm_correction directive specified for the FSM. Safe correcting FSMs are converted to safe detecting

FSMs. For a list of restrictions to using this attribute, see *Limitations*, on page 347.

The voter system to resolve the three systems can be implemented in different ways. See *Voter Insertion Examples*, on page 348 for details.

4. Add redundancy for I/Os connected to distributed TMR modules.

   See *Using Redundancy for I/O Connectors*, on page 353 for details.

   Note: You can optionally add I/O redundancy to TMR modules and only when it is required.

5. Provide error monitoring.

   See *Error Monitoring for High Reliability Features*, on page 361 for details.

   Note: You can optionally specify the error monitoring Tcl commands for distributed TMR.

6. Run synthesis.

   The entire module with the specified attribute is triplicated. In addition to the inserted TMR elements in the netlist, you can view them in the HDL Analyst tool.

Limitations

Distributed TMR support includes the following limitations:

- Cross-boundary optimizations, such as retiming, packing, or constant propagation are not allowed across a distributed TMR module boundary.
- TMR is always applied to an FSM in the parent module when it is tagged for distributed TMR. Safe correcting FSM is converted to safe detecting.
- Distributed TMR does not support the following:
  - Module with tristates
  - Module that includes an instantiated module
  - Module that includes a BUFG for Xilinx devices
  - Top-level module
  - Conflicting syn_ramstyle values of ecc or tmr

Copyright © 2013 Synopsys, Inc.

Voter Insertion Examples

The tool implements voter logic as synchronous voters (in the case of loops) and asynchronous or output voters (to resolve TMR logic). You can control voter insertion for synchronous voters with the syn_vote_loops attribute. The following examples illustrate different cases of voter insertion:

- Example 1: Standard Distributed TMR, on page 348
- Example 2: Distributed TMR with Cyclic Module, on page 348
- Example 3: Distributed TMR with Multiple Modules, on page 349

Example 1: Standard Distributed TMR

This example has three separate instances created for the specified module. The inputs to the original module are connected to all three instances. The outputs are fed to a majority voter, which produces a single output connected to the fanout of the original module.

Example 2: Distributed TMR with Cyclic Module

This example has a loop in the specified module. To restore the state after an SEU fault, the loop in each of the three instances must insert a voter. Any of the TMR module outputs can be the final output.

If you used a mitigation mechanism to avoid the accumulation of faults in a sequential feedback path and do not want the overhead of voter logic, you can turn off voter insertion by setting the syn_vote_loops attribute to false. For details, see *syn_vote_loops*, on page 1591 in the *Reference Manual*.

Example 3: Distributed TMR with Multiple Modules

This example shows a case where multiple hierarchical modules have distributed TMR applied. The modules are triplicated in each path with three voters, until the last module is reached. The last module converges the three paths to one, and then drives the non-TMR module.

Copyright © 2013 Synopsys, Inc.

Implementing Duplication with Comparison (DWC)

Duplication with Comparison (DWC) is a mitigation methodology that protects against SEU failures in a design. It duplicates modules and compares the outputs of this circuit to determine if an error occurs. Signal nets are also duplicated to be the same as the original connectivity. Since this option limits the redundancy performed, it has less impact on area and QoR than distributed TMR.

The following procedure specifies how to implement DWC.

1. Select a supported technology family in the Synplify Premier tool.

   See *Vendor Support for Distributed TMR and DWC*, on page 351 for a list.

2. Enable the Enhanced Optimization option.

3. From the Attributes tab of the SCOPE editor, specify the syn_radhardlevel attribute with a value of duplicate_with_compare for the modules you want to duplicate.

   For the syn_radhardlevel attribute syntax, see *syn_radhardlevel*, on page 1462 in the *Reference Manual*.

   You must set the attribute on views or modules. You can specify hierarchical modules that require increased reliability to prevent SEU and SET

faults. You cannot apply this attribute on the top-level module or globally.

4. Add redundancy for I/Os connected to distributed DWC modules.

   See *Using Redundancy for I/O Connectors*, on page 353 for details.

   Note: You can optionally add I/O redundancy to DWC modules and only when it is required.

5. Provide error monitoring.

   See *Error Monitoring for High Reliability Features*, on page 361 for details.

   Note: You must specify the error monitoring Tcl commands for DWC. Otherwise, the mapper generates an error message.

6. Run synthesis.

   The modules with the specified attribute are duplicated. In addition to the inserted DWC elements in the netlist, you can view them in the HDL Analyst tool.

Vendor Support for Distributed TMR and DWC

The following technology families support the specification of distributed TMR and DWC at the module level:

Altera	Arria V, Cyclone III, Cyclone IV, Cyclone V, Stratix III, Stratix IV, Stratix V
Xilinx	Artix-7, Kintex-7, and Virtex families

DWC Examples

The tool implements DWC with comparator logic (XOR/OR) that is connected to the error monitor port. The following examples illustrate different cases of DWC:

- Example 1: DWC with Single Module, on page 352
- Example 2: DWC with Multiple Modules, on page 352

Copyright © 2013 Synopsys, Inc.

Example 1: DWC with Single Module

This example has two separate instances created for the specified module. The outputs are fed to comparator logic that is connected to the error monitoring port (ERROR_FLAG). Output from the redundant copy is connected to the original output OUT.

Example 2: DWC with Multiple Modules

This example shows a case where multiple hierarchical modules have DWC applied. The modules are duplicated and their outputs are fed to comparator logic (XOR/OR) that connects to the corresponding error monitoring ports, respectively (ERROR_FLAG_A and (ERROR_FLAG_B). Outputs for the redundant copies of module A are connected to the corresponding inputs of the redundant copies of module B. Output from the redundant copy of module B is connected to the original output OUT.

Using Redundancy for I/O Connectors

I/O connectors that interface to modules with distributed TMR or DWC allow you to access the signals within its boundaries. The synthesis software implements connections to copies of modules that have been specified for distributed TMR or DWC and ensures that their inputs and outputs are not a single-point of failure.

The following procedure specifies how to implement redundancy for I/O connectors.

1. Select a supported technology family in the Synplify Premier tool.

   See *Vendor Support for Distributed TMR and DWC*, on page 351 for a list.

2. Enable the Enhanced Optimization option.

3. Instantiate the I/O connector modules in the RTL that are to be connected to these I/O connectors.

4. From the Attributes tab of the SCOPE editor, specify the syn_highrel_ioconnector attribute with a value of 1 along with connections to I/Os on one side and the DTMR/DWC on the other side.

Copyright © 2013 Synopsys, Inc.

The following logic must be included in the module definition to qualify as an I/O connector:

- DWC ip connector: AND/OR logic
- DWC op connector: feedthrough connection
- DTMR ip connector: any majority voter logic
- DTMR op connector: feedthrough connection

For the syn_highrel_ioconnector attribute syntax, see *syn_highrel_ioconnector*, on page 1352 in the *Reference Manual*.

I/O Connector Redundancy Examples

Create redundant ports and instantiate connector objects with the attribute syn_highrel_ioconnector for DWC or DTMR modules. The following examples illustrate cases for I/O connector redundancy:

- Example 1: I/O Connectors with DWC, on page 354
- Example 2: I/O Connectors with DTMR, on page 355

Example 1: I/O Connectors with DWC

This example creates redundant ports and instantiates connector objects with the syn_highrel_ioconnector attribute. The following are specified:

- Valid input connector for 2 inputs and 1 output with AND/OR logic.
- Valid output connector for 1 input and 2 outputs with feedthrough logic.

Use the "syn_radhardlevel=duplicate_with_compare" attribute with the module for which DWC is applied and that is connected to error monitoring. Two copies of the modules (A0, A1) with their outputs fed to comparator logic (XOR/OR) are connected to the error monitoring port (ERROR_FLAG).

The inputs (IN0, IN1) and outputs (OUT0, OUT1) are connected to the corresponding redundant copies (A0, A1) with their hierarchies dissolved.

Example 2: I/O Connectors with DTMR

This example creates redundant ports and instantiates connector objects with the syn_highrel_ioconnector attribute. The following are specified:

- Valid input connector for 3 inputs and 1 output with voter logic.
- Valid output connector for 1 input and 3 outputs with feedthrough logic.

Use the "syn_radhardlevel=distributed_tmr" attribute with the module for which distributed TMR is applied. The inputs (IN0, IN1, IN2) are fed to the corresponding redundant copies (A0, A1, A2) and their outputs are connected to the outputs (OUT0, OUT1, OUT2) through individual voter logic.

Copyright © 2013 Synopsys, Inc.

Using TMR or ECC for RAMs

You can specify fault tolerance for Altera and Xilinx RAM architectures, using local TMR or Error Correction Code (ECC). The details are described here:

- Specifying Local TMR for RAMs
- Specifying ECC for RAMs

Specifying Local TMR for RAMs

For some Altera and Xilinx architectures, you can specify local TMR for RAM to increase fault tolerance. Local TMR protects sequential elements like RAMs from SEUs by tripling and then voting on registers.

Do the following to implement local TMR for a RAM:

1. Select a supported device in the Synplify Premier tool.

   See *Vendor Support for Local TMR RAMs*, on page 357 for a list.

2. Enable the Enhanced Optimization option.

3. Specify the syn_ramstyle attribute with the tmr value.

You can specify the tmr value in combination with other syn_ramstyle values, like block_ram. For the attribute syntax, see *syn_ramstyle*, on page 1467 in the *Reference Manual*.

4. Provide error monitoring.

   See *Error Monitoring for High Reliability Features*, on page 361 for details.

   Note: You can optionally specify the error monitoring Tcl commands for local TMR RAM.

5. Run synthesis.

   The software implements the inferred RAM primitive in triplicate for the design.

6. Check the log file for the TMR RAM.

Vendor Support for Local TMR RAMs

The following technology families support local TMR for RAMs:

Altera	Arria V, Cyclone-III, Cyclone-IV, Cyclone-V, Stratix-III, Stratix-IV, Stratix-V
Xilinx	Artix-7, Kintex-7, and Virtex families

Specifying ECC for RAMs

Using Error Correction Code (ECC) RAM lets you detect and correct single-bit errors on RAM. Some Altera and Xilinx architectures offer ECC memories that automatically detect and correct single-bit errors. The synthesis software can infer and connect these vendor-specific memories.

The following procedure shows you how to ensure high reliability by specifying ECC RAMs. You can also combine the error-mitigating ECC RAMs with TMR to prevent false data from being captured by the memory and propagated to other parts of the circuitry.

1. Select a supported device in the Synplify Premier tool.

   For a list of supported technologies, see *Vendor Support for ECC for RAMs*, on page 358. Note that some Xilinx ECC primitives are not supported.

2. Enable the Enhanced Optimization option.

Copyright © 2013 Synopsys, Inc.

3. Specify the syn_ramstyle attribute with a value of ecc.

   If the ecc value is used in combination with other syn_ramstyle values like select_ram, it has a higher priority. This means that if ecc and mlab are both specified for an Altera device, mlab is ignored because ecc has higher priority. For more about the attribute syntax, see *syn_ramstyle*, on page 1467 in the *Reference Manual*.

4. For Altera RAM modes that do not support an ECC implementation, either specify the syn_ramstyle attribute with the ecc and no_rw_check values, or disable read-write check for the RAM with the set_option -RWCheckOnRam 0 command.

5. Provide error monitoring.

   See *Error Monitoring for High Reliability Features*, on page 361 for details.

   Note: You can optionally specify the error monitoring Tcl commands for ECC RAM.

6. Run synthesis.

   The software infers the built-in ECC RAM primitive in the design. The synthesis tool creates the ECC RAM block primitive and its associated glue logic (for example, address decoders/encoders or comparators), while keeping the RAM interface the same.

7. Check the log file for the ECC RAM.

Vendor Support for ECC for RAMs

The following technology families support the specification of TMR for RAMs:

Altera	Stratix-V
Xilinx	Artix-7, Kintex-7, Virtex-5, Virtex-6, and Virtex-7 The following ECC technology primitives are not supported: • Byte-wide write enable RAM • RAM with synchronous set/reset signal • RAM with output register clock enable • Explicitly instantiated block RAM with parity pins for wider configuration

Copyright © 2013 Synopsys, Inc.

Specifying Safe FSMs

Typically, unspecified or unreachable FSM states are optimized away during synthesis, so if an SEU causes a bit to be inverted, the FSM can be put into an undefined, invalid state, and lock up the circuit. An SEU fault is a change of state caused by ions or electro-magnetic radiation that affects sequential elements. The basic principle of a safe FSM is to prevent the state machine from getting stuck in an unknown state because of an SEU.

Safe FSMs are primarily required by commercial or military-aerospace users, especially those who want to ensure that their FSMs are tolerant of single event upset (SEU) faults and continue to function correctly.

The following procedures describe ways to ensure high reliability and fault tolerance for FSMs:

- Implementing Safe Case FSMs, on page 359
- Implementing FSMs with Hamming 3 Encoding, on page 360

Vendor Support for Safe FSMs

The following technology families support the specification of safe case and Hamming Distance 3 for FSMs:

Altera	Cyclone-IV, Stratix-V
Xilinx	Virtex-5, Virtex-6, Virtex-7

Implementing Safe Case FSMs

To implement safe case FSMs, follow this procedure:

1. Select a supported device in the Synplify Pro or Synplify Premier synthesis tools.

   See *Vendor Support for Safe FSMs*, on page 359 for a list.

2. To globally implement safe case FSMs, go to the Implementation Options-> High Reliability tab, and enable the Preserve and Decode Unreachable States (FSM, Counters, Sequential Logic) option.

Copyright © 2013 Synopsys, Inc.

The high reliability safe case option turns off sequential optimizations that would otherwise optimize away some FSM states.

3. To apply safe case on an individual module or architecture, set the syn_safe_case directive on a module.

   This is a Verilog example: *module* **/* syn_safe_case =1*/**

   For further details about this directive, see *syn_safe_case*, on page 1517 in the *Reference Manual*.

4. Provide error monitoring.

   See *Error Monitoring for High Reliability Features*, on page 361 for details.

   Note: You can optionally specify the error monitoring Tcl commands for safe FSM.

Implementing FSMs with Hamming 3 Encoding

Generating fault-tolerant FSMs with Hamming-3 encoding is an effective error mitigation technique. Hamming 3 encoding detects and corrects single-bit errors in the FSM state registers with a Hamming distance of 3, and ensures that the FSM continues to operate correctly after the error is corrected.

Parity bits are added to the FSM state register to generate Hamming distance 3 encoding. Error correction logic is used to automatically detect and correct an SEU in the register bits.

Copyright © 2013 Synopsys, Inc.

To provide immunity against single-bit errors and implement FSMs with Hamming 3 encoding, do the following:

1. Select a supported device in the Synplify Premier tools.

   See *Vendor Support for Safe FSMs*, on page 359 for a list of supported devices.

2. To enable Hamming 3 encoding globally, go to the Implementation Options-> High Reliability tab, and enable the FSM Error Correction Using Hamming Distance 3 option.

3. To apply Hamming 3 encoding to an individual FSM, use the syn_fsm_correction directive.

   For a description of this directive, see *syn_fsm_correction*, on page 1332 in the *Reference Manual*.

When Hamming distance 3 is applied to an FSM that is inside a module to which distributed TMR is applied, then TMR takes precedence and the FSM is triplicated along with the rest of the module and the Hamming distance 3 specification ignored. This also generates a warning message in the log file.

Error Monitoring for High Reliability Features

The software can inform you when it detects that an error has occurred. Based on the type of error, you may want to take corrective action (such as, reset the design or enable/disable scrubbing). Error monitoring can occur for the following:

- Modules specified for distributed TMR
- Modules specified for distributed DWC

Copyright © 2013 Synopsys, Inc.

- RTL RAM specified for ECC; provide error status bit
- Local RAM TMR
- Finite state machines with unreachable states
- I/O connectors with distributed TMR or DWC

Error Monitoring Procedure

Error monitoring requires that you do the following:

1. As a prerequisite, a top-level port or user instantiated error monitoring IP (EMIP) must be provided in the RTL. You must specify syn_keep on the signal/wire feeding into the top-level port or EMIP port.
2. Specify the connectivity between the module/instance being monitored for the error with the error monitoring IP port or top-level port for the error monitoring module. You must define the control signals of the source and elements for error monitoring with the following Tcl commands:
   - syn_create_err_net
   - syn_connect

   Add these commands in the constraint file. For a descriptions of the syntax, see *syn_connect*, on page 1723 and *syn_create_err_net*, on page 1724.

   Here are some examples of specifying these commands:

```
# Unregistered error flag connected to top-level error port
syn_create_err_net {-name {error_flag} -inst {i:inst_A}}

syn_connect {-from {n:error_flag} -to {p:emp}}

# Error flag with 4 stage pipeline registers with asynchronous
   reset connected to EMIP port

syn_create_err_net {-name {error_flag} -inst {i:inst_A}
   -err_pipe_num {4} -err_clk {n:inst_A.clk} -err_reset
   {n:inst_A.rst} -err_synch {false}}

syn_connect {-from {n:error_flag} -to {t:inst_emip.emp}}
```

The usage model for setting up the error monitoring varies slightly depending on the type of high reliability feature you are implementing: DWC, DTMR, or ECC.

Error Monitoring Examples

Error monitoring is applied to an instance. You can have a 1-bit error port for each instance. To do this, use the following Tcl commands:

- syn_create_err_net – Creates logic (XOR/OR) that compares the outputs for the selected module and connects it to the source of the new net that you specified. You can also specify pipeline registers along with clock and other control signals to improve timing through the comparator circuitry.
- syn_connect – Connects the new net that you specified to an existing net, top-level port, or input port of the instantiated Error Monitoring IP (EMIP).

See the following examples:

- Example 1: Error Monitoring with DWC, on page 363
- Example 2: Error Monitoring with DTMR, on page 364
- Example 3: Error Monitoring with ECC/TMR RAM, on page 365
- Example 4: Error Monitoring with FSM, on page 367

Example 1: Error Monitoring with DWC

To set up the error monitoring for a DWC module, specify the following:

1. On the module:

```
define_attribute {v:A} syn_radhardlevel {duplicate_with_compare}
```

2. On the instance:

```
– syn_create_err_net -name {error_flag} -inst {i:A_1}
  -err_pipe_num  {1}
  -err_clk {n: hierarchical path to the input pin of clk}
  -err_reset {n: hierarchical path to the input pin of reset}
  -err_set {n: hierarchical path to the input pin of set}
  -err_enable {n: hierarchical path to the input pin of enable}
  -err_synch {false}
```


Copyright © 2013 Synopsys, Inc.

- `syn_connect -from {n:error_flag} -to {t:EMIP.err_port}`

For an example, see *Example 1: DWC with Single Module*, on page 352.

Example 2: Error Monitoring with DTMR

To set up the error monitoring for a DTMR module and provide access to error bits, specify the following:

1. On the module:

   ```
   define_attribute {v:A} syn_radhardlevel {distributed_tmr}
   ```

2. On the instance:

   - ```
     syn_create_err_net -name {error_flag} -inst {i:A_}
     -err_pipe_num {1}
     -err_clk {n: hierarchical path to the input pin of clk}
     -err_reset {n: hierarchical path to the input pin of reset}
     -err_set {n: hierarchical path to the input pin of set}
     -err_enable {n: hierarchical path to the input pin of enable}
     -err_synch {false}
     ```
   - `syn_connect -from {n:error_flag} -to {t:EMIP.err_port}`

In this example, module A is specified for distributed TMR and error monitoring. Three copies of the module (A0, A1, A2) are created, for which their outputs are fed through the majority voter logic. The outputs of A0, A1, and A2 are connected to the comparator logic (XOR/OR) and the error flag is connected to the EMP port.

Copyright © 2013 Synopsys, Inc.
     ```

Example 3: Error Monitoring with ECC/TMR RAM

To set up the error monitoring for a DTMR module and provide access to error bits, specify the following:

1. On the instance:

   ```
   define_attribute {i:RAM} syn_ramstyle {ecc|tmr}
   ```

2. On the instance:

   – ```
     syn_create_err_net -name {error_flag} -inst {i:RAM}
     -err_pipe_num {1}
     -err_clk {n: hierarchical path to the input pin of clk}
     -err_reset {n: hierarchical path to the input pin of reset}
     -err_set {n: hierarchical path to the input pin of set}
     -err_enable {n: hierarchical path to the input pin of enable}
     -err_synch {false}
     ```

   – ```
     syn_connect -from {n:error_flag} -to {t:EMIP.err_port}
     ```


Copyright © 2013 Synopsys, Inc.

In this example, the "syn_ramstyle=ecc" attribute is applied globally to the RAM, for which error monitoring is implemented. The built-in ECC is inferred for the block RAM. Error bits are OR'ed and the error flag is connected to the EMP port. For

- Altera – ECCSTATUS
- Xilinx – SBITERR and DBITERR

In this example, the "syn_ramstyle=tmr" attribute is applied globally to the RAM, for which error monitoring is implemented. Three copies of the inferred block RAM primitive are created with outputs voted through majority voter logic. The outputs (TMR0, TMR1, TMR2) are compared with (XOR/OR) logic to the error flag connected to the EMP port.

Example 4: Error Monitoring with FSM

To set up the error monitoring for an FSM and provide access to error bits, specify the following:

1. Enable Preserve and Decode Unreachable States on the High Reliability tab of the Implementation Options panel for the state machine.
2. On the instance:
   - `syn_create_err_net {-name {error_flag} -inst {i:state[1:3]}}`
   - `syn_connect -from {{n:error_flag} -to {t:EMIP.err_port}}`

In this example, the Preserve and Decode Unreachable States option is enabled on the High Reliability tab of the Implementation Options panel for the compiler to implement recovery logic by inferring the stateerrordetect IP. The Tcl commands connect the output of this IP to the EMP port for error monitoring of the FSM to occur.

Copyright © 2013 Synopsys, Inc.

statemachine
clk
rst
enable
Q[3:0]
state[0:3]
EMP
[0]
[1]
[2]
OUT0
OUT1
OUT2
stateerrdetect
I[3:0]
OUT
un1_state

Inferring RAMs

There are two methods of handling RAMs: instantiation and inference. The software can automatically infer RAMs if they are structured correctly in your source code. For details, see the following sections:

- Inference Versus Instantiation, on page 369
- Basic Guidelines for Coding RAMs, on page 370
- Specifying RAM Implementation Styles, on page 375
- Implementing Altera RAMs Automatically, on page 376
- Implementing Xilinx RAMs Automatically, on page 379
- Implementing Altera Stratix Multi-Port RAMs, on page 382
- Inferring Altera Stratix III LUTRAMs, on page 383
- Inferring Xilinx Block RAMs Using Registered Addresses, on page 385
- Inferring Xilinx Block RAMs Using Registered Output, on page 387
- Initializing Xilinx RAM, on page 398
- Mapping Xilinx ROM to Block RAM, on page 392

For information about generating RAMs with SYNCore, see *Specifying RAMs with SYNCore*, on page 939.

Inference Versus Instantiation

There are two methods to handle RAMs: instantiation and inference. Many FPGA families provide technology-specific RAMs that you can instantiate in your HDL source code. The software supports instantiation, but you can also set up your source code so that it infers the RAMs. The following table sums up the pros and cons of the two approaches.

Inference in Synthesis	Instantiation
Advantages	**Advantages**
Portable coding style	Most efficient use of the RAM primitives of a specific technology
Automatic timing-driven synthesis	Supports all kinds of RAMs
No additional tool dependencies	

Copyright © 2013 Synopsys, Inc.

Inference in Synthesis	Instantiation
Limitations	**Limitations**
Glue logic to implement the RAM might result in a sub-optimal implementation.	Source code is not portable because it is technology-dependent.
Can only infer synchronous RAMs	Limited or no access to timing and area data if the RAM is a black box.
No support for address wrapping	Inter-tool access issues, if the RAM is a black box created with another tool.
Pin name limitations means some pins are always active or inactive	

Basic Guidelines for Coding RAMs

Read through the limitations before you start. See *Inference Versus Instantiation*, on page 369 for information. The following steps describe general rules for coding RAMs so that the compiler infers them; to ensure that they are mapped to the vendor-specific implementation you want, see *Specifying RAM Implementation Styles*, on page 375, *Implementing Altera RAMs Automatically*, on page 376, and *Implementing Xilinx RAMs Automatically*, on page 379.

1. Make sure that the RAM meets minimum size and address width requirements for your technology. The software implements RAMs that are smaller than the minimum as registers.

2. Structure the assignment to a VHDL signal/Verilog register as follows:
   - To infer a RAM, structure the code as an indexed array or a case structure. Code it as a two-dimensional array (VHDL) or memory (Verilog) with writes to one process.
   - Control the structure with a clock edge and a write enable.

   The software extracts RAMs even if write enables are tied to true (VCC), if you have complex write enables coded in nested if statements, or if you have RAMs with synchronous resets.

3. For a single-port RAM, make the address for indexing the write-to the same as the address for the read-from. The following code and figure illustrate how the software infers a single-port RAM.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
```

```
port (q : out std_logic_vector(3 downto 0);

      d : in std_logic_vector(3 downto 0);
      addr : in std_logic_vector(2 downto 0);
      we : in std_logic;
      clk : in std_logic);
end ramtest;

architecture rtl of ramtest is

type mem_type is array (7 downto 0) of std_logic_vector
      (3 downto 0);
signal mem : mem_type;

begin
q <= mem(conv_integer(addr));

process (clk) begin
if rising_edge(clk) then
   if (we = '1') then
      mem(conv_integer(addr)) <= d;
   end if;
end if;
end process;

end rtl;
```

For technology-specific details, see *Implementing Altera RAMs Automatically*, on page 376 and *Implementing Xilinx RAMs Automatically*, on page 379.

Copyright © 2013 Synopsys, Inc.

4. For a dual-port RAM, make the write-to and read-from addresses different. The following figure and code example illustrate how the software infers a dual-port RAM.

```
module ram16x8(z, raddr, d, waddr, we, clk);
output [7:0] z;
input [7:0] d;
input [3:0] raddr, waddr;
input we;
input clk;
reg [7:0] z;
reg [7:0] mem0, mem1, mem2, mem3, mem4, mem5, mem6, mem7;
reg [7:0] mem8, mem9, mem10, mem11, mem12, mem13, mem14, mem15;
always @(mem0 or mem1 or mem2 or mem3 or mem4 or mem5 or mem6 or
   mem7 or mem8 or mem9 or mem10 or mem11 or mem12 or mem13 or
   mem14 or mem15 or raddr)
begin
   case (raddr[3:0])
   4'b0000: z = mem0;
   4'b0001: z = mem1;
   4'b0010: z = mem2;
   4'b0011: z = mem3;
   4'b0100: z = mem4;
   4'b0101: z = mem5;
   4'b0110: z = mem6;
   4'b0111: z = mem7;
   4'b1000: z = mem8;
   4'b1001: z = mem9;
   4'b1010: z = mem10;
   4'b1011: z = mem11;
   4'b1100: z = mem12;
```

Copyright © 2013 Synopsys, Inc.

```
      4'b1101: z = mem13;
      4'b1110: z = mem14;
      4'b1111: z = mem15;
   endcase
   end

   always @(posedge clk) begin
      if(we) begin
         case (waddr[3:0])
         4'b0000: mem0 = d;
         4'b0001: mem1 = d;
         4'b0010: mem2 = d;
         4'b0011: mem3 = d;
         4'b0100: mem4 = d;
         4'b0101: mem5 = d;
         4'b0110: mem6 = d;
         4'b0111: mem7 = d;
         4'b1000: mem8 = d;
         4'b1001: mem9 = d;
         4'b1010: mem10 = d;
         4'b1011: mem11 = d;
         4'b1100: mem12 = d;
         4'b1101: mem13 = d;
         4'b1110: mem14 = d;
         4'b1111: mem15 = d;
      endcase
   end
   end
   endmodule
```

For technology-specific details, see *Implementing Altera RAMs Automatically*, on page 376 and *Implementing Xilinx RAMs Automatically*, on page 379.

5. To infer multi-port RAMs or nrams (certain technologies only), do the following:
   - Target a technology that supports multi-port RAMs.
   - Register the read address.
   - Add the syn_ramstyle attribute with a value of no_rw_check. If you do not do this, the compiler errors out.
   - Make sure that the writes are to one process. If the writes are to multiple processes, use the syn_ramstyle attribute to specify a RAM.

Copyright © 2013 Synopsys, Inc.

6. For RAMs where inference is not the best solution, use either one of these approaches:
   - Implement them as regular logic using the syn_ramstyle attribute with a value of registers. You might want to do this if you have to conserve RAM resources.
   - Instantiate RAMs using the black box methodology. Use this method in cases where RAM is implemented in two cells instead of one because the RAM address range spans the word limit of the primitive and the software does not currently support address wrapping. If the address range is 8 to 23 and the RAM primitive is 16 words deep, the software implements the RAM as two cells, even though the address range is only 16 words deep. Refer to the list of limitations in *Inference Versus Instantiation*, on page 369 and the vendor-specific information referred to in the previous step to determine whether you should instantiate RAMs.

7. Synthesize your design.

   The compiler infers one of the following RAMs from the source code. You can view them in the RTL view:

RAM1	RAM
RAM2	Resettable RAM
NRAM	Multi-port RAM

   If the number of words in the RAM primitive is less than the required address range, the compiler generates two RAMs instead of one, leaving any extra addresses unused.

   Once the compiler has inferred the RAMs, the mapper implements the inferred RAMs in the technology you specified. For details of how to map the RAM inferred by the compiler to the implementation you want, see *Specifying RAM Implementation Styles*, on page 375, *Implementing Altera RAMs Automatically*, on page 376, and *Implementing Xilinx RAMs Automatically*, on page 379.

Specifying RAM Implementation Styles

You can manually influence how RAMs are implemented with the syn_ramstyle attribute, as described in the following procedure. The valid values vary slightly, depending on the technology you use. Check the *Reference Manual* for the values that apply to the technology you choose.

If you would rather set up your design so that the software automatically maps the RAMs to the components you want, see *Implementing Altera RAMs Automatically*, on page 376 and *Implementing Xilinx RAMs Automatically*, on page 379 for some vendor-specific details.

1. If you do not want to use RAM resources, attach the syn_ramstyle attribute with a value of registers to the RAM instance name or to the signal driven by the RAM.

   Use this value for small RAMs. The software implements the RAMs according to the technology. They can be implemented as registers, LPMs, dedicated RAM resources (QuickLogic) or synchronous dual-port memory cells (some Lattice technologies).

2. To use the dedicated memory resources on the FPGA (Altera technologies), do the following:
   - Set syn_ramstyle to block_ram.
   - For Altera technologies like Stratix, specify mapping to TriMatrix memories by setting syn_ramstyle to M512, M4K, or M-RAM.
   - If you do not want glue logic created, register the RAM output. For Altera Stratix designs, you can set syn_ramstyle to no_rw_check.

3. To implement RAMs using dedicated Block SelectRAM+ in Xilinx Virtex technologies, do the following. To use distributed memory, see the next step.
   - Set syn_ramstyle to block_ram.
   - Register the read address, because the technology is fully synchronous.
   - If you do not want to generate glue logic for dual-port RAMs, either register the RAM output or set syn_ramstyle to no_rw_check. Use this attribute value only if you do not care about a read/write check.

4. To implement RAMs using distributed memory in Xilinx technologies, set syn_ramstyle to select_ram. Set syn_ramstyle explicitly, because by

Copyright © 2013 Synopsys, Inc.

default the software first implements block RAM, and select RAM only if it cannot implement block RAM.

Implementing Altera RAMs Automatically

The following procedure shows you how to implement various Altera RAMs automatically. You can always override the automatic implementation by specifying the syn_ramstyle attribute, as described in *Specifying RAM Implementation Styles*, on page 375 or instantiate LPMs instead of using RAMs.

1. Follow the guidelines described for RAM inference by the compiler (*Basic Guidelines for Coding RAMs*, on page 370).

   The Altera mapper does not implement any RAMs that are not first inferred by the compiler.

2. To implement Stratix block RAM, follow these guidelines:
   - If you are a Verilog user, avoid using blocking statements when you model the RAMs because not all blocking assignments are mapped to block RAM.
   - Synchronize the read and write addresses by registering either the read address or output. RAMs with asynchronous read and write are mapped to logic.
   - Use syn_ramstyle with a value of no_rw_check to disable the creation of glue logic in dual-port mode.
   - Make sure the Altera Quartus tool is installed, for best results with Stratix II devices. When Quartus is installed, the synthesis software takes advantages of the Stratix II RAMs and MACs and writes out Stratix II primitives that you can view in the Technology view. If you do not install Quartus, the synthesis software infers LPMs.

   During synthesis, the mapper maps Altera Stratix RAM to ALTSYNCRAM in the following modes:

Single-port	One address bus
Dual-port	One address bus (where old data cannot be obtained in single-port mode), or Two buses: one each for read and write.
Bidirectional	Two buses: one for read/write and one for read only

3. To implement Stratix single-port RAMs, ensure the following:
   - The read and write addresses share a single address.
   - There is only one data input.
   - There is only RAM output.
   - Either the read address or the output is registered.
   - For multiple clocks, both the read address and the output must be registered.

   The mapper maps the RAM to the dedicated memory resource, ALTSYNCRAM, which is fully synchronous. It is mapped in SINGLE_PORT mode, and all ports are registered. The ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

4. To implement Stratix dual-port RAMs, make sure of the following:
   - The code is written so that the hardware exactly matches the RTL behavior. For example, if your code allows simultaneous reads and writes to the same address, it can result in a mismatch between the RTL and hardware behaviors. In such a case, the mapper does not map the RAM inferred by the compiler to the dedicated ALTSYNCRAM resources and you get a warning message. For Verilog examples, see *Stratix, Arria and Cyclone Dual-Port Verilog RAM Examples,* on page 720 and for VHDL examples, see *Stratix, Arria and Cyclone Dual-Port VHDL RAM Example,* on page 948 in the *Reference Manual*.
   - The design has different read and write addresses.
   - There is only one data input.
   - There is only RAM output.
   - Either the read address or the output is registered.

Copyright © 2013 Synopsys, Inc.

- The read and write addresses can have different clocks. However if you register the read, write, and output, at least two of them must share a clock.
- For multiple clocks, both the read address and the output must be registered.

The mapper maps the RAM to ALTSYNCRAM in DUAL_PORT mode, which is fully synchronous. The actual ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

The following figure shows one dual-port RAM implementation:

5. To implement Stratix dual-port RAMs in bidirectional mode, make sure of the following:
   - The code must be written so that there are no mismatches between the hardware and RTL behaviors. For Verilog examples, see *Stratix, Arria and Cyclone Dual-Port Verilog RAM Examples*, on page 720 and for VHDL examples, see*Stratix, Arria and Cyclone Dual-Port VHDL RAM Example*, on page 948 in the *Reference Manual*.
   - The design has different read and write addresses. There are two read addresses.
   - There is only one data input.
   - There are two RAM outputs.
   - Either the read address or the output is registered.
   - The read and write addresses can have different clocks. However if you register the read, write, and output, at least two of them must share a clock.

– For multiple clocks, both the read address and the output must be registered.

The mapper maps the RAM to ALTSYNCRAM in BIDIR_DUAL_PORT mode, which is fully synchronous. The actual ALTSYNCRAM implementation is determined by the Quartus place-and-route tool.

6. To implement Stratix multi-ports RAMs automatically, see *Implementing Altera Stratix Multi-Port RAMs*, on page 382.

Implementing Xilinx RAMs Automatically

The following procedure shows you how to implement various Xilinx RAMs automatically. You can always override the automatic implementation by specifying the syn_ramstyle attribute, as described in *Specifying RAM Implementation Styles*, on page 375.

1. Follow the guidelines described for RAM inference by the compiler (*Basic Guidelines for Coding RAMs*, on page 370).

   The Xilinx mapper does not implement any RAMs that are not first inferred by the compiler.

2. To automatically implement distributed RAM, make sure the write operation is synchronous and the read operation is asynchronous.

   The Xilinx mapper implements RAMs inferred by the compiler as asynchronous RAMs, using the CLB resources.

3. To implement block SelectRAM+, do the following:
   – Make the write port synchronous. The read port can be asynchronous.
   – Register the read address (see *Inferring Xilinx Block RAMs Using Registered Addresses*, on page 385).
   – Make sure the RAM is a minimum of 2K bits.

   The Xilinx mapper automatically implements RAMs inferred by the compiler as Block SelectRAM+, using the dedicated memory resources on the FPGA. The enable pin is tied to active and the reset pin is tied to inactive.

4. To implement single-port block RAM automatically, do the following:

Copyright © 2013 Synopsys, Inc.

- Register the output. (see *Inferring Xilinx Block RAMs Using Registered Output*, on page 387).
- Make the read and write addresses the same.
- Make sure that the read and write clocks are the same.
- Make sure the read and write enables are the same.

The Xilinx mapper automatically implements RAMs inferred by the compiler as single-port Block SelectRAM+, using the dedicated memory resources on the FPGA. The enable signal has the highest priority. Where applicable, the tool uses the parity bus to infer data bus widths. The mapper also uses the Write modes in some Xilinx architectures, as described in the next step.

5. To implement dual-port block RAM automatically, do the following:
   - Register the output. (see *Inferring Xilinx Block RAMs Using Registered Output*, on page 387).
   - Your design can have different read and write addresses, multiple clocks, and different read and write enables.

   The Xilinx mapper implements RAMs inferred by the compiler as dual-port block SelectRAM+, using the dedicated memory resources on the FPGA. The dual-port RAM has only one write port. The software automatically inserts glue logic for address collision and recovery, unless you specify otherwise with the syn_ramstyle attribute.

The mapper also implements the Write modes available with certain Xilinx architectures to indicate the output value when the write enable is active. The RAM implementations are shown here:

Write Mode	Xilinx Architecture	RAM Implementation
Writefirst (data_in goes to data_out)	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3 Automotive, and Spartan-3E	Block SelectRAM+ (single-port or dual-port) Distributed RAM
	Spartan-II, Spartan-IIE, and Spartan-IIE Automotive	Block SelectRAM+ (single-port or dual-port) Distributed RAM
Readfirst (memory goes to data_out)	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3 Automotive, and Spartan-3E	Block SelectRAM+ (single-port) Distributed RAM
	Spartan-II, Spartan-IIE, and Spartan-IIE Automotive	Distributed RAM
Nochange (data_out is unchanged)	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3 Automotive, and Spartan-3E	Block SelectRAM+ (single-port) Distributed RAM
	Spartan-II, Spartan-IIE, and Spartan-IIE Automotive	Distributed RAM

6. To implement true dual-port block (multi-port) RAM automatically, make sure the design meets the following conditions:
   - The compiler has inferred multi-port RAMs (nrams). See *Basic Guidelines for Coding RAMs*, on page 370 for details.
   - The inferred nram has two writes and one read. The read shares an address with only one of the write ports, or two inferred RAMs share the same write addresses, clocks, and enables, but have different read addresses. In the latter case, the mapper pairs the RAMs together and maps them to true dual-port RAM.

   The Xilinx mapper implements RAMs inferred by the compiler as true dual-port block SelectRAM+, using the dedicated memory resources on the FPGA. The dual-port RAM has one read port and multiple write

Copyright © 2013 Synopsys, Inc.

ports. Each write port has its own write clock, write enable, data in, and write address.

Implementing Altera Stratix Multi-Port RAMs

The software can infer true multi-port RAMs, where both ports are used to read and write simultaneously. Implementing Stratix ALTSYNCRAM components is a two-step process: first the synthesis compiler infers the RAM primitive, and then the mapper maps the primitive to ALTSYNCRAM.

1. Make sure the compiler infers an nram, by following the guidelines in *Basic Guidelines for Coding RAMs*, on page 370.

   For multi-port RAMs, the compiler infers an *n*ram primitive, where *n* is the number of write ports. You can view this in the RTL view.

2. To map the nram automatically to ALTSYNCRAM, ensure that it follows these guidelines:
   - The nram has two writes and one read. The read shares an address with only one of the write ports.
   - Make sure there are only two clocks, one for each port.
   - You cannot have more than two write ports; *n*ram primitives with more than two ports are mapped to logic.
   - The read address is registered.
   - If the output is registered, the mapper retimes and infers block RAM.

   The software maps *n*ram primitives as follows:

Primitive Description	Mapping
2 write ports, 1 read. The read shares an address with only one of the write ports	ALTSYNCRAM in bidir mode
2 nrams each with 2 write addresses and 1 read, which share the same write addresses, clocks, and enables, but different read addresses	Paired together and mapped to ALTSYNCRAM
> 2 write ports	Logic
> 2 clocks	Logic

After synthesis, the software writes out the following for the place-and-route tool:

```
defparam mem_1_1_Z.lpm_type = "altsyncram";
```

Inferring Altera Stratix III LUTRAMs

The Altera Stratix III technology has LUTRAM memory components. MLAB (Memory LAB) resources are configured as LUTRAM. MLABs can be configured as single-port RAM or ROM, or simple dual-port RAM. LUTRAM writes occur on the falling edge of the clock and can be configured to have synchronous or asynchronous read.

The following procedure shows you how to set up the synthesis tool to map memory to MLABs and LUTRAMs. Note that you cannot currently map to a LUTRAM ROM, nor can you initialize asynchronous memory.

1. Start with a Stratix III design.
2. Enable the Clearbox flow option.

   If this option is not enabled, the memories are mapped to ALTSYNCRAM or ALTDPRAM instead of LUTRAM.
3. Set the syn_ramstyle attribute to MLAB.

   This automatically maps the RAM to MLAB resources, which can be configured as LUTRAMs. If you do not want to infer LUTRAM, set syn_ramstyle to registers.

   For Verilog code examples that implement LUTRAM, see *Stratix III LUTRAM Verilog Examples*, on page 717 in the *Reference Manual*.
4. Synthesize your design.

   The software maps aysnchronous RAMs to LUTRAM, and reports resource utilization in the log file, like this example:

```
Memory ALUTs:   10  (0% of 19000)
```


Copyright © 2013 Synopsys, Inc.

The following shows how the software maps an SDPRAM with registered output and asynchronous read to a simple dual-port RAM in the RTL view:

The following shows how the same memory is mapped in the Technology view to a stratixiii_mlab_cell LUTRAM component:

Inferring Xilinx Block RAMs Using Registered Addresses

There are two ways to infer block RAMs in Xilinx Virtex device families: using registered addresses and using registered output. For information about the latter, see *Inferring Xilinx Block RAMs Using Registered Output*, on page 387. The following procedure shows you how to set up your code with an explicit read address register.

The software does not currently infer block RAMs for Virtex family designs automatically; you have to use an attribute. It inserts bypass logic to resolve a read/write behavior difference between the RTL and post-synthesis gate-level simulations. It inserts the glue logic because it does not know the output at the read port when the read address and the write address access the same memory location.

1. Use instantiation instead of inference in the following cases where the software currently does not infer the RAMs:
   - RAMs with enable signals, RAM resets, or initialization settings.
   - Inaccessible pins: read enable pins are always active, and reset pins are always inactive.
   - Dual-port RAMs with read/write on a port.
2. For single-port RAM, do the following:
   - Make sure the read and write clocks are the same.
   - Make sure the read and write addresses are the same.
   - Make sure the enable signals are the same. Use only write enable signals.
   - Register the address, as shown in the following code:

```
always @(posedge clk)
   if(we)
      mem[addr] = din;

always @(posedge clk)
   addr_reg = addr;

assign dout = mem[addr_reg];
```


Copyright © 2013 Synopsys, Inc.

– To forward-annotate initialization values, use the Xilinx INIT property, as described in *Initializing Xilinx RAM*, on page 398.

3. For dual-port RAM, do the following:

   – Register the address as shown in this code:

```
always @(posedge clk)
   if(we)
      mem[waddr] = din;

always @(posedge clk)
   raddr_reg = raddr;

assign dout = mem[raddr_reg];
```

– To forward-annotate initialization values, see *Initializing Xilinx RAM, on page 398*.

4. To prevent the insertion of glue logic, add the syn_ramstyle="no_rw_check" attribute.

   By default, the software inserts glue logic when the read and write addresses access the same memory location, because it does not know the output of the read port. The glue logic prevents a mismatch between the RTL and post-synthesis simulation results. See *Specifying RAM Implementation Styles, on page 375* or the *Reference Manual* for more information about this attribute.

5. To infer Virtex block RAM, add the syn_ramstyle="block_ram" attribute to the register signal in your source code, or to the output signal of the RAM in the SCOPE window. See *Specifying RAM Implementation Styles, on page 375* or the *Reference Manual* for more information about this attribute.

6. Run synthesis.

   The software implements the circuit using Xilinx RAMB4_S<*n*>_-S<*n*> primitives. The Xilinx dual-port block RAM is implemented with one write port.

Inferring Xilinx Block RAMs Using Registered Output

For Virtex-II and Virtex-II Pro designs, you can code block RAMs with registered output as described here, or with registered addresses (see *Inferring Xilinx Block RAMs Using Registered Addresses, on page 385*). For information about forward-annotating initialization values, see *Initializing Xilinx RAM, on page 398*.

This information is organized into these subtopics:

- Advantages of Using Registered Output, on page 388
- Block RAM Mapping for Virtex-II Write Modes, on page 388
- Xilinx Single-Port Example with Registered Output, on page 389
- Xilinx Single-Output Dual-Port Example with Registered Output, on page 392

Copyright © 2013 Synopsys, Inc.

Advantages of Using Registered Output

The registered output method allows you to use reset and enable lines as well as the different write modes in Virtex architectures. The following table shows the advantages of using registered output instead of registered addresses:

Registered Read Address	Registered Output
Read and write clocks must be the same	Can have different clocks
No enable, reset supported	Supports enable and reset signals

Block RAM Mapping for Virtex-II Write Modes

The following table summarizes how the software implements Block RAM for the write modes in different Virtex families when you register the outputs. See *Xilinx Single-Port Example with Registered Output*, on page 389 and *Xilinx Single-Output Dual-Port Example with Registered Output*, on page 392 for examples.

	Virtex-II	Virtex-II Pro
WRITEFIRST Mode		
With enable and reset, enable takes precedence	SP	SP
With enable and reset, reset takes precedence	SP	SP
Without enable	SP	SP
Without reset	SP	SP
Without enable or reset	SP	SP
READFIRST Mode		
With enable and reset, enable takes precedence	SP	SP
With enable and reset, reset takes precedence	SP	SP

SP: Single-port block RAM

DP: Single-output, dual-port block RAM

	Virtex-II	Virtex-II Pro
Without enable	SP	SP
Without reset	SP	SP
Without enable or reset	SP	SP
NOCHANGE Mode		
With enable and reset, enable takes precedence	SP	SP
With enable and reset, reset takes precedence	SP	SP
Without enable	SP	SP
Without reset	SP	SP
Without enable or reset	SP	SP

SP: Single-port block RAM

DP: Single-output, dual-port block RAM

Xilinx Single-Port Example with Registered Output

This example shows the single-port 257x9 RAM with reset and enable extracted from the following code, where the output is registered. To forward-annotate initialization values, use the Xilinx INIT property as described in *Initializing Xilinx RAM*, on page 398.

Copyright © 2013 Synopsys, Inc.

un7_en
un6_en
do_4[8:0]
do[8:0]
ram1
RADDR[8:0]
DATA[8:0]
WADDR[8:0]
WE
CLK
DOUT[8:0]
mem[8:0]
un12_en
di[8:0]
addr[8:0]
clk
[8:0]

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ramtest is port(
do : out std_logic_vector(8 downto 0);
addr : in std_logic_vector(8 downto 0);
di : in std_logic_vector(8 downto 0);
en,clk,we,rst : in std_logic);

end ramtest;

architecture beh of ramtest is
type memtype is array (256 downto 0) of std_logic_vector(8 downto
0);
signal mem : memtype;
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "block_ram";

begin
process(clk)
begin
   if clk'event and clk='1' then
   if(en='1') then
      if (rst='1') then
         do <= "000000000";
      elsif (we='1') then
         do <= di;
      else
         do <= mem(CONV_INTEGER(addr));
      end if;
   end if;
   end if;
end process;

process(clk)
begin
   if clk'event and clk='1' then
   if (en='1' and we='1') then
      mem(CONV_INTEGER(addr)) <= di;
   end if;
   end if;
end process;

end beh;
```


Copyright © 2013 Synopsys, Inc.

Xilinx Single-Output Dual-Port Example with Registered Output

For Virtex architectures, the software maps components to single-output dual-port block RAMs when the RAMs are coded with different read and write addresses, different read and write clocks, and different enable signals.

To forward-annotate initialization values, see *Initializing Xilinx RAM*, on page 398.

In the following example, the read port has no enable, and the component is mapped to single-output dual-port block RAM:

```
always@(posedge clk_r)
   if(rst == 1)
      data_out = 0;
   else
      data_out = mem[addr_out];
always @(posedge clk_w)
   if (we) mem[addr_in] = data_in;
```

Mapping Xilinx ROM to Block RAM

For Xilinx Virtex architectures, the software can map ROM into block RAM, provided you follow the guidelines in this procedure.

1. Place a dff register in front of the ROM, or place one of the following after the ROM:

Asynchronous	Synchronous
dff, dffe	
dffr, dffre	sdffr, sdffre
dffs, dffse	sdffs, sdffse
dffpatr, dffpatre	sdffpatr, sdffpatre

where dffe is an enabled flip-flop, dffre is an enabled flip-flop with asynchronous reset, dffse is an enabled flip-flop with asynchronous set, and dffpatre is an enabled, vectored flip-flop with asynchronous reset pattern.

2. Ensure that the registers and ROMs are within the same hierarchy.
3. Make sure that at least half the addresses possess assigned values. For example, in a ROM with ten address bits (1024 unique addresses), at least 512 of those unique addresses must be assigned values.
4. Specify the syn_romstyle attribute with the value set to block_rom.
5. Synthesize the design.

   The software maps the ROM into block RAM.

Copyright © 2013 Synopsys, Inc.

Initializing RAMs

You can specify startup values for RAMs and pass them on to the place-and-route tools. See the following for ways to set the initial values:

- Initializing RAMs in Verilog, on page 394
- Initializing RAMs in VHDL, on page 395
- Initializing Xilinx RAM, on page 398

Initializing RAMs in Verilog

In Verilog, you specify startup values using initial statements, which are procedural `assign` statements guaranteed by the language to be executed by the simulator at the start of the simulation. This means that any assignment to a variable within the body of the initial statement is treated as if the variable was initialized with the corresponding LHS value. You can initialize memories using the built-in load memory system tasks $readmemb (binary) and $readmemh (hex). For Xilinx RAMs, you can alternatively initialize RAMs using the INIT property, as described in *Specifying the INIT Property for Xilinx RAMs (Verilog)*, on page 399 and *Specifying the INIT Property with Attributes*, on page 403.

The following procedure is the recommended method for specifying initial values:

1. Create a data file with an initial value for every address in the memory array. This file can be a binary file or a hex file. See *Initialization Data File*, on page 680 in the *Reference Manual* for details of the formats for these files.

2. Do the following in the Verilog file to define the module:

   – Include the appropriate task enable statement, $readmemb or $readmemh, in the initial statement for the module:

     $readmemb ("*fileName***", *memoryName* [, *startAddress* [, *stopAddress*]]);**

     $readmemh ("*fileName***", *memoryName* [, *startAddress* [, *stopAddress*]]);**

     Use $readmemb for a binary file and use $readmemh for a hex file. For descriptions of the syntax, see *Initial Values in Verilog*, on page 676 in the *Reference Manual*.

– Make sure the array declaration matches the order in the initial value data file you specified. As the file is read, each number encountered is assigned to a successive word element of the memory. The software starts with the left-hand address in the memory declaration, and loads consecutive words until the memory is full or the data file has been completely read. The loading order is the order in the declaration. For example, with the following memory definition, the first line in the data file corresponds to address 0:

```
reg [7:0] mem_up [0:63]
```

With this next definition, the first line in the data file applies to address 63:

```
reg [7:0] mem_down [63:0]
```

3. To forward-annotate initial values, use the $readmemb or $readmemh statements, as described in *Initializing RAMs with $readmemb and $readmemh*, on page 399.

   See *RAM Initialization Example*, on page 679 in the *Reference Manual* for an example of a Verilog single-port RAM.

Initializing RAMs in VHDL

There are two ways to initialize RAMs in the VHDL code: with signal declarations or with variable declarations. For Xilinx RAMs, you can also initialize RAMs using the INIT property (see *Specifying the INIT Property for Xilinx RAMs (VHDL)*, on page 402 and *Specifying the INIT Property with Attributes*, on page 403).

Initializing VHDL Rams with Signal Declarations

The following example shows a single-port RAM that is initialized with signal initialization statements. For alternative methods, see *Initializing VHDL Rams with Variable Declarations*, on page 397 and *Specifying the INIT Property for Xilinx RAMs (VHDL)*, on page 402.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity w_r2048x28 is
port (
   clk : in  std_logic;
   adr : in  std_logic_vector(10 downto 0);
   di : in  std_logic_vector(26 downto 0);
   we : in  std_logic;
   dout : out std_logic_vector(26 downto 0));
end;

architecture arch of w_r2048x28 is

-- Signal  Declaration --

type MEM is array(0 to 2047) of std_logic_vector (26 downto 0);
signal memory : MEM := (
"11111111111111111000000000000"
,"11111001101110101001111 0001"
,"11100111100011110010110 0111"
,"11001011001110111001111 0001"
,"10100111100011111110011 0111"
,"10000000000000111111111 1111"
,"01011000011100111110011 0111"
,"00110100110001111001111 0001"
,"00011000011100110010110 0111"
,"00000110010001101001111 0001"
,"00000000000000100000000 0000"
,"00000110010001010110000 1110"
,"00011000011100001101001 1000"
,"00110100110001000110000 1110"
,"01011000011100000001100 1000"
,"01111111111111000000000 0000"
,"10100111100011000001100 1000"
,"11001011001110000110000 1110"
,"11100111100011001101001 1000"
,"11111001101110010110000 1110"
,"11111111111111101111111 1111"
,"11111001101110101001111 0001"
,"11100111100011110010110 0111"
,"11001011001110111001111 0001"
,"10100111100011111110011 0111"
,"10000000000000111111111 1111"
,others => (others => '0'));

begin
process(clk)
```

```
begin
   if rising_edge(clk) then
      if (we = '1') then
         memory(conv_integer(adr)) <= di;
      end if;
         dout <= memory(conv_integer(adr));
   end if;
end process;

end arch;
```

Initializing VHDL Rams with Variable Declarations

The following example shows a RAM that is initialized with variable declarations. For alternative methods, see *Initializing VHDL Rams with Signal Declarations*, on page 395 and *Initializing RAMs with $readmemb and $readmemh*, on page 399.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity one is
generic (data_width    : integer := 6;
   address_width :integer  := 3
);

port ( data_a    :in std_logic_vector(data_width-1 downto 0);
   raddr1    :in unsigned(address_width-2 downto 0);
      waddr1 :in unsigned(address_width-1 downto 0);
      we1    :in std_logic;
      clk :in std_logic;
      out1 :out std_logic_vector(data_width-1 downto 0) );
end;

architecture rtl of one is
   type mem_array is array(0 to 2**(address_width) -1) of
      std_logic_vector(data_width-1 downto 0);
begin

WRITE1_RAM :  process (clk)
   variable mem : mem_array := (1 => "111101", others => (1=>'1',
      others => '0'));
begin
   if rising_edge(clk) then
      out1  <= mem(to_integer(raddr1));
      if (we1 = '1') then
```


Copyright © 2013 Synopsys, Inc.

```
            mem(to_integer(waddr1)) := data_a;
         end if;
      end if;
   end process WRITE1_RAM;
   end rtl;
```

Initializing Xilinx RAM

In addition to the methods described in *Initializing RAMs in Verilog*, on page 394 and *Initializing RAMs in VHDL*, on page 395, you can also define and forward-annotate Xilinx RAM initialization values as summarized in this table:

Verilog

- `$readmemebh` or `$readmemh`
  See *Initializing RAMs with $readmemb and $readmemh*, on page 399.
- `INIT` property in `defparam` statement
  See *Specifying the INIT Property for Xilinx RAMs (Verilog)*, on page 399.
- `INIT` property in global comment (`/* synthesis INIT` or `/*synthesis` *`INIT_xx=<value>`*
  See *Specifying the INIT Property for Xilinx RAMs (Verilog)*, on page 399.

VHDL

- `INIT` property on label
  See *Specifying the INIT Property for Xilinx RAMs (VHDL)*, on page 402.

Attributes

- `INIT` property in SCOPE
  See *Specifying the INIT Property with Attributes*, on page 403.
- `define_attribute` statements in the `sdc` file
  See *Specifying the INIT Property with Attributes*, on page 403.

Note the following differences between the above methods:

- You can use the INIT property with any code. The $readmemb and $readmemh system tasks are only applicable in Verilog.
- The Verilog initial values only affect the output of the compiler, not the mapper. They ensure that the synthesis results match the simulation results, and are not forward-annotated.

Initializing RAMs with $readmemb and $readmemh

1. Create a data file with an initial value for every address in the memory array. This file can be a binary file or a hex file. See *Initialization Data File*, on page 680 in the *Reference Manual* for details.

2. Include one of the task enable statements, $readmemb or $readmemh, in the initial statement for the module:

   $readmemb ("*fileName***",** *memoryName* [**,** *startAddress* [**,** *stopAddress*]]**) ;**
   $readmemh ("*fileName***",** *memoryName* [**,** *startAddress* [**,** *stopAddress*]]**) ;**

   Use $readmemb for a binary file and $readmemh for a hex file. For details about the syntax, see *Initial Values in Verilog*, on page 676 in the *Reference Manual*.

Specifying the INIT Property for Xilinx RAMs (Verilog)

You can initialize and forward-annotate the values for Xilinx Verilog RAMs by specifying the INIT property. In Verilog, you can do this by using the `defparams` statement or by specifying the property in a global comment. The following examples illustrate these two methods.

- Using defparam to Specify Initialization Values for Xilinx RAMs, on page 399
- Using Global Comments to Specify Initialization Values for Xilinx RAMs, on page 400

Using defparam to Specify Initialization Values for Xilinx RAMs

1. Include defparam statements in the Verilog file, using one statement for each word. Use the following syntax for the INIT property:

   defparam *name***.INIT_***xx***=***value***;**

name	Is the name of the RAM.
xx	Indicates the part of the RAM you are initializing. It can be any hex number from 00 to FF.
value	Sets the initialization value in hex.

Copyright © 2013 Synopsys, Inc.

The following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64 x 4 and 256 x 16 = 4K).

```
RAMB4_S4 pkt_len_ram_lo (
   .CLK  (clock),
   .RST  (1'b0),
   .EN   (1'b1),
   .WE   (we),
   .ADDR (address),
   .DI   (data),
   .DO   (q)
   );
defparam pkt_len_ram_lo.INIT_00=
"0017001600150014001300120011001000 0f000e000d000c000b000a00090008 ";
defparam pkt_len_ram_lo.INIT_01=
"0027002600250024002300220021002000 1f001e001d001c001b001a00190018';
defparam pkt_len_ram_lo.INIT_02=
"0037003600350034003300320031003000 2f002e002d002c002b002a00290028";
...
defparam pkt_len_ram_lo.INIT_0F=
"0107010601050104010301020101010000 ff00fe00fd00fc00fb00fa00f900f8";
```

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx P&R software in the EDIF netlist. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Using Global Comments to Specify Initialization Values for Xilinx RAMs

1. Add the INIT property.
   - Attach the INIT property to the instance as shown:

RAM	`/* synthesis INIT = "value" */`
Block RAM	`/* synthesis INIT_xx = "value" */`

   - Define the INIT_xx=*value* property as follows:

xx	Indicate the part of the RAM you are initializing with a number from 00 to FF.
value	Set the initialization value, in hex. You have 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

– Keep the entire statement on one line. Let your editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

2. For RAM, specify a hex value for the INIT statement as shown here:

```
RAM16X1S RAM1(...) /* synthesis INIT = "0000" */;
```

3. For Virtex block RAM, specify 16 different INIT statements. End the initialization data with a semicolon.

   All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

   he following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64 x 4 and 256 x 16 = 4K).

```
RAMB4_S4 pkt_len_ram_lo (
   .CLK  (clock),
   .RST  (1'b0),
   .EN   (1'b1),
   .WE   (we),
   .ADDR (address),
   .DI   (data),
   .DO   (q)
   )

/* synthesis
INIT_00="0017001600150014001300120011001000 0f000e000d000c000b000a00090008"
INIT_01="0027002600250024002300220021002000 1f001e001d001c001b001a00190018"
INIT_02="0037003600350034003300320031003000 2f002e002d002c002b002a00290028"
...
INIT_0F="0107010601050104010301020101010000 ff00fe00fd00fc00fb00fa00f900f8"
*/;
```

   When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software.

Copyright © 2013 Synopsys, Inc.

The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Specifying the INIT Property for Xilinx RAMs (VHDL)

1. Add the INIT property.
   - Attach the INIT property to the label as shown:

RAM	`attribute INIT of` *object* `: label is "`*value*`";`
Block RAM	`attribute INIT_xx of` *object* `: label is "`*value*`";`

   - Keep the entire statement on one line. Let the editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

2. For RAM, specify hex values for the INIT statement as shown:

```
attribute INIT of RAM1 : label is "0000";:
```

3. For Virtex block RAM, specify 16 different INIT statements.
   - Define the INIT_xx=*value* property as follows:

xx	Indicate the part of the RAM you are initializing with a number from 00 to FF.
value	Set the initialization value, in hex. You have 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

   All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

   - End the initialization data with a semicolon.

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Copyright © 2013 Synopsys, Inc.

Specifying the INIT Property with Attributes

When you set the INIT property in the source code, it is difficult to pass on the values if the RAM instances are mapped to registers. When you specify INIT as an attribute, either in the SCOPE window or the constraint file, you are working with a mapped RAM, and the values are passed to the P&R tool. You can use this method to specify the initialization values for a RAM whether you are using Verilog or VHDL.

1. Compile and map the design.
2. Select the RAM in the SCOPE window.
   - Open SCOPE and go to the Attributes panel.
   - Open the Technology view. Drag and drop the RAM into the window.
3. Define the INIT (RAM) or INIT_*xx* = *value* (Block RAM) property in SCOPE. Alternatively you can edit the `sdc` file using define_attribute statements.

xx	Indicates the part of the RAM you are initializing with a number from 00 to FF.
value	Sets the initialization value, in hex. You have 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

When you synthesize the design, the software forward-annotates the initialization values as constraints in the `sdc` file. The following example shows a value of ABBADABAABBADABA defined for INIT_00 and INIT_01 on mem.mem_0_0 in the `sdc` file:

```
define_attribute {i:mem.mem_0_0} INIT_00 {ABBADABAABBADABA}
define_attribute {i:mem.mem_0_0} INIT_01 {ABBADABAABBADABA}
```

These initialization values are forward-annotated as constraints to the Xilinx place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Copyright © 2013 Synopsys, Inc.

Initializing Xilinx Select RAM

For select RAM in some Virtex architectures, you can specify an initial value and have it honored.

1. Specify the initial value in the RTL using $readmemb or another supported mechanism.

   See *Initializing Xilinx RAM*, on page 398 for details.

   If the tool maps the ram1 to a supported select RAM in Virtex 4 and later designs, it automatically distributes the initial value you specified. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

Inferring Shift Registers

The software infers shift registers for Xilinx Virtex and Altera Stratix architectures. For Altera Stratix designs, you can implement the shift register as an altshift_tap with tap points. Use the following procedure.

1. Set up the HDL code for the sequential shift components. See *Shift Register Examples*, on page 407 for examples.

   Note the following:

   Altera
   - The registers must be dff or dffe registers.
   - All registers must have the same type and use the same control signals.

   Xilinx
   - The new component represents a set of three or more registers that can be shifted left (from a low address to a higher address).
   - The contents of only one register can be seen at a time, based on the read address.
   - For static components, the software only taps the output of the last register. The read address of the inferred component is set to a constant.

2. Specify the implementation style with the syn_srlstyle attribute.

syn_srlstyle Value	Implemented as...
registers	registers
select_srl	Xilinx SRL16 primitives
no_extractff_srl	Xilinx SRL16 primitives without output flip-flops
altshift_tap	Altera Altshift_tap components

   You can set the value globally or on individual registers. For example, if you do not want the components automatically mapped to shift registers, set it globally to registers. You can then override this with specific settings on individual registers as needed.

Copyright © 2013 Synopsys, Inc.

3. For Altera shift registers, use attributes to control how the registers are packed:

To...	Attach...
Prevent a register from being packed into shift registers	syn_useioff or syn_noprune to the register. You can also use syn_srlstyle with a value of registers.
Prevent two registers from being packed into the same shift registers	syn_keep between the two registers. The algorithm slices the chain vertically, and packs the two registers into separate shift registers.
Specify that two registers be packed in different shift registers	syn_srlstyle with different group names for the registers you want to separate (syn_srlstyle= altshift_tap, *groupName*)

If you do not specify anything, registers are packed across hierarchy. In all cases, the registers are not packed if doing so violates DRC restrictions.

4. Run synthesis

After compilation, the software displays the components as seqShift components in the RTL view. The following figure shows the components in the RTL view.

In the technology view, the components are implemented as Xilinx SRL16 or Altera altshift_tap primitives or registers, depending on the attribute values you set.

5. Check the results in the log file and the technology file.

   The log file reports the shift registers and the number of registers packed in them. The following is an Altera example, showing the number of registers packed and taps inferred:

```
ShiftTap:       1 (10100 registers)
```

Shift Register Examples

Altera Shift Register (VHDL)

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (
clk : in std_logic;
din : in std_logic_vector(31 downto 0);
dout : out std_logic_vector(31 downto 0);
tap7 : out std_logic_vector(31 downto 0);
tap6 : out std_logic_vector(31 downto 0);
tap5 : out std_logic_vector(31 downto 0);
tap4 : out std_logic_vector(31 downto 0);
tap3 : out std_logic_vector(31 downto 0);
tap2 : out std_logic_vector(31 downto 0);
tap1 : out std_logic_vector(31 downto 0)
);
end test;

architecture rtl of test is
type dataAryType is array(31 downto 0) of std_logic_vector(31
downto 0);
signal q : dataAryType;

begin
process (Clk)
begin
   if (Clk'Event And Clk = '1') then
      q <= (q(30 DOWNTO 0) & din);
   end if;
end process;
```


Copyright © 2013 Synopsys, Inc.

```
dout <= q(31);
tap7 <= q(27);
tap6 <= q(23);
tap5 <= q(19);
tap4 <= q(15);
tap3 <= q(11);
tap2 <= q(7);
tap1 <= q(3);

end rtl;
```

Altera Shift Register (Verilog)

```
module
test(dout,tap7,tap6,tap5,tap4,tap3,tap2,tap1,din,shift,clk);

output [7:0] dout;
output [7:0] tap7;
output [7:0] tap6;
output [7:0] tap5;
output [7:0] tap4;
output [7:0] tap3;
output [7:0] tap2;
output [7:0] tap1;
input [7:0] din;
input shift, clk;
reg [7:0] q[63:0];

integer n;

assign dout = q[63];
assign tap7 = q[55];
assign tap6 = q[47];
assign tap5 = q[39];
assign tap4 = q[31];
assign tap3 = q[23];
assign tap2 = q[15];
assign tap1 = q[7];

always @(posedge clk)
      if (shift)
      begin
         q[0] <= din;
         for (n=0; n<63; n=n+1)
            begin
```

```
                q[n+1] <= q[n];
            end
        end

endmodule
```

Xilinx Shift Register (VHDL)

This is a VHDL example of a shift register with no resets. It has four 8-bit wide registers and a 2-bit wide read address. Registers shift when the write enable is 1.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity srltest is
   port ( inData: std_logic_vector(7 downto 0);
      clk, en : in std_logic;
      outStage : in integer range 3 downto 0;
      outData: out std_logic_vector(7 downto 0)
);
end srltest;

architecture rtl of srltest is
   type dataAryType is array(3 downto 0) of std_logic_vector(7
downto 0);
   signal regBank : dataAryType;

begin
   outData <= regBank(outStage);
   process(clk, inData)
      begin
         if (clk'event and clk = '1') then
            if (en='1') then
            regBank <= (regBank(2 downto 0) & inData);
            end if;
         end if;
   end process;
end rtl;
```


Copyright © 2013 Synopsys, Inc.

Xilinx Shift Register (Verilog)

```
module test_srl(clk, enable, dataIn, result, addr);
input clk, enable;
input [3:0] dataIn;
input [3:0] addr;
output [3:0] result;
reg [3:0] regBank[15:0];
integer i;

always @(posedge clk) begin
   if (enable == 1) begin
      for (i=15; i>0; i=i-1) begin
         regBank[i] <= regBank[i-1];
      end
      regBank[0] <= dataIn;
   end
end

assign result = regBank[addr];

endmodule
```

Working with LPMs

Some technologies support LPMs (Library of Parameterized Modules), which are technology-independent logic functions that are parameterized for scalability and adaptability. There are two ways to instantiate LPMs in your source code: as black boxes, or by using prepared components.

The following table compares the methods for instantiating LPMs.

	Black Box Method	**Verilog Library/VHDL Prepared Component Method**
Applies to any LPM	Yes	No
Synthesis LPM timing support	No	Yes
Synthesis procedure	More coding	Simple

See the following for more information about instantiating LPMs:

- Instantiating Altera LPMs as Black Boxes, on page 412
- Instantiating Altera LPMs Using VHDL Prepared Components, on page 416
- Instantiating Altera LPMs Using a Verilog Library, on page 418

Copyright © 2013 Synopsys, Inc.

Instantiating Altera LPMs as Black Boxes

The method described here uses either Verilog or VHDL LPMs in the Altera-prescribed megafunction format. Alternatively, you can use the methods described in *Instantiating Altera LPMs Using a Verilog Library*, on page 418 or *Instantiating Altera LPMs Using VHDL Prepared Components*, on page 416. For information about using Clearbox in Synplify Pro Stratix designs, see *Implementing Megafunctions with Clearbox Models*, on page 1002.

1. Generate the LPM using the Altera MegaWizard Plug-in Manager. If you generate the file using another method, make sure to use the same MegaWizard format, where ALTSYNCRAM is instantiated.

   For examples of coding style, see *LPM Megafunction Example (Verilog)*, on page 412 and *LPM Megafunction Example (VHDL)*, on page 414.

2. Manually edit the LPM file and add the syn_black_box attribute to make the LPM a black box for synthesis.

   See the examples in *LPM Megafunction Example (Verilog)*, on page 412 and *LPM Megafunction Example (VHDL)*, on page 414.

3. Instantiate the LPM in your design so that the LPM is not the top level. Synthesize the design.

   The synthesis software treats the LPM as a black box. After synthesis, the software writes out a `vqm` file where the module is a black box.

4. Add the original LPM file to the results directory and use it along with the `vqm` file to place and route your design.

   The place-and-route software uses the synthesized design information from the `vqm` file and adds in the ALTSYNCRAM parameter information from the original megafunction file to place and route the LPM RAM correctly.

LPM Megafunction Example (Verilog)

The following file shows the coding style the Altera MegaWizard uses to generate a Verilog LPM file, with the syn_black_box attribute added for synthesis.

```
module mylpm (
   data,
   wren,
   wraddress,
   rdaddress,
   clock,
   q)/* synthesis syn_black_box */;

   input    [7:0] data;
   input      wren;
   input    [4:0] wraddress;
   input    [4:0] rdaddress;
   input      clock;
   output    [7:0] q;

   wire [7:0] sub_wire0;
   wire [7:0] q = sub_wire0[7:0];

   altsyncram    altsyncram_component (
      .wren_a (wren),
      .clock0 (clock),
      .address_a (wraddress),
      .address_b (rdaddress),
      .data_a (data),
      .q_b (sub_wire0));

   defparam
      altsyncram_component.operation_mode = "DUAL_PORT",
      altsyncram_component.width_a = 8,
      altsyncram_component.widthad_a = 5,
      altsyncram_component.numwords_a = 32,
      altsyncram_component.width_b = 8,
      altsyncram_component.widthad_b = 5,
      altsyncram_component.numwords_b = 32,
      altsyncram_component.lpm_type = "altsyncram",
      altsyncram_component.width_byteena_a = 1,
      altsyncram_component.outdata_reg_b = "UNREGISTERED",
      altsyncram_component.indata_aclr_a = "NONE",
      altsyncram_component.wrcontrol_aclr_a = "NONE",
      altsyncram_component.address_aclr_a = "NONE",
      altsyncram_component.address_reg_b = "CLOCK0",
      altsyncram_component.address_aclr_b = "NONE",
```


Copyright © 2013 Synopsys, Inc.

```
        altsyncram_component.outdata_aclr_b = "NONE",
        altsyncram_component.read_during_write_mode_mixed_ports
           = "DONT_CARE",
        altsyncram_component.ram_block_type = "AUTO",
        altsyncram_component.intended_device_family = "Stratix";

endmodule
```

LPM Megafunction Example (VHDL)

Instantiate a file like this one at the top level, and include it in the project file, as shown in the preceding figure.

```
ENTITY myram IS
   PORT(
   data        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
   wren        : IN STD_LOGIC := '1';
   wraddress : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
   rdaddress : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
   clock        : IN STD_LOGIC ;
   q        : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );
END myram;

ARCHITECTURE SYN OF mylpram IS
   SIGNAL sub_wire0    : STD_LOGIC_VECTOR (7 DOWNTO 0);

   COMPONENT altsyncram
   GENERIC (
      operation_mode        : STRING;
      width_a        : NATURAL;
      widthad_a        : NATURAL;
      numwords_a        : NATURAL;
      width_b        : NATURAL;
      widthad_b        : NATURAL;
      numwords_b        : NATURAL;
      lpm_type        : STRING;
      width_byteena_a        : NATURAL;
      outdata_reg_b        : STRING;
      indata_aclr_a        : STRING;
      wrcontrol_aclr_a        : STRING;
      address_aclr_a        : STRING;
      address_reg_b        : STRING;
      address_aclr_b        : STRING;
```

```
      outdata_aclr_b        : STRING;
      read_during_write_mode_mixed_ports        : STRING;
      ram_block_type        : STRING;
      intended_device_family        : STRING
   );

   PORT (
      wren_a    : IN STD_LOGIC ;
      clock0    : IN STD_LOGIC ;
      address_a    : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
      address_b    : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
      q_b    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      data_a    : IN STD_LOGIC_VECTOR (7 DOWNTO 0)
   );

    END COMPONENT;

BEGIN
   <= sub_wire0(7 DOWNTO 0);

   altsyncram_component : altsyncram
   GENERIC MAP (
      operation_mode => "DUAL_PORT",
      width_a => 8,
      widthad_a => 5,
      numwords_a => 32,
      width_b => 8,
      widthad_b => 5,
      numwords_b => 32,
      lpm_type => "altsyncram",
      width_byteena_a => 1,
      outdata_reg_b => "CLOCK0",
      indata_aclr_a => "NONE",
      wrcontrol_aclr_a => "NONE",
      address_aclr_a => "NONE",
      address_reg_b => "CLOCK0",
      address_aclr_b => "NONE",
      outdata_aclr_b => "NONE",
      read_during_write_mode_mixed_ports => "DONT_CARE",
      ram_block_type => "AUTO",
      intended_device_family => "Stratix"
   )
```


Copyright © 2013 Synopsys, Inc.

```
    PORT MAP (
       wren_a => wren,
       clock0 => clock,
       address_a => wraddress,
       address_b => rdaddress,
       data_a => data,
       q_b => sub_wire0
    );

END SYN;
```

Instantiating Altera LPMs Using VHDL Prepared Components

Prepared LPM components, available as component declarations in LPM_COMPONENTS, use generics instead of attributes to specify different design parameters. After you specify library and use clauses, instantiate the components and assign (map) the ports and the values for the generics. Refer to the Quartus II documentation for ports and generics that require mapping. See *Prepared LPM Components Provided by Synopsys (VHDL)*, on page 1898 in the *Reference Manual* for a list of available LPM components.

The prepared components method is the simplest to use, but it does not cover all available LPMs. For other methods, see *Instantiating Altera LPMs as Black Boxes*, on page 412), or *Instantiating Altera LPMs Using a Verilog Library*, on page 418 (Altera only). The prepared components method uses generics instead of attributes to specify design parameters. You specify the library, instantiate the components, and assign (map) the ports and the values for the generics.

1. In the higher-level entity, specify the appropriate `library` and `use` clauses.

   ```
   library lpm;
   use lpm.components.all;
   ```

   The prepared components are in the *installDirectory*\lib\vhd directory. The Altera LPM prepared components are in `lpm`.

2. Instantiate the prepared component.

3. Assign the ports and values for the generics. These assignments override the generic values in the library. Refer to the vendor documentation for details about ports and values for generics.

This is an example of an LPM instantiated at a higher level:

```
library ieee, lpm;
use ieee.std_logic_1164.all;
use lpm.components.all;

entity test is
   port(data : in std_logic_vector (5 downto 0);
      distance : in std_logic_vector (7 downto 0);
      result : out std_logic_vector (5 downto 0);
end test;

architecture arch1 of test is
begin
u1 : lpm_clshift
   generic map (LPM_WIDTH=>6, LPM_WIDTHDIST =>8)
   port map (data=>data, distance=>distance, result=>result);
end arch1;
```

Prepared Components LPM Example (Altera)

This example shows the instantiation of the prepared component lpm_ram_dq:

```
library lpm;
use lpm.lpm_components.all;
library ieee;
use ieee.std_logic_1164.all;

entity lpm_inst is
   port (clock, we: in std_logic;
         data : in std_logic_vector(3 downto 0);
         address : in std_logic_vector(3 downto 0);
         q : out std_logic_vector (3 downto 0));
end lpm_inst;

architecture arch1 of lpm_inst is
begin
   I0 : lpm_ram_dq
      generic map (LPM_WIDTH => 4,
                   LPM_WIDTHAD => 4,
                   LPM_TYPE => "LPM_RAM_DQ")
      port map (data => data,
                   address => address,
                   we => we,
                   inclock => clock,
                   outclock => clock,
                   q => q);
end arch1;
```


Copyright © 2013 Synopsys, Inc.

Instantiating Altera LPMs Using a Verilog Library

For Altera LPMs, you can also instantiate LPMs from a Verilog library. For other methods of instantiating LPMs, see *Instantiating Altera LPMs as Black Boxes*, on page 412 and *Instantiating Altera LPMs Using VHDL Prepared Components*, on page 416.

1. Add the Verilog library file *installDirectory*/lib/altera/altera_lpm.v to your project. The following shows the code for lpm_ram_dp.

```
module lpm_ram_dp (q, data, wraddress, rdaddress, rdclock,
wrclock,
   rdclken, wrclken, rden, wren) /*synthesis syn_black_box*/;

parameter lpm_type = "lpm_ram_dp";
parameter lpm_width = 1;
parameter lpm_widthad = 1;
parameter numwords = 1<<lpm_widthad;
parameter lpm_indata = "REGISTERED";
parameter lpm_outdata = "REGISTERED";
parameter lpm_rdaddress_control = "REGISTERED";
parameter lpm_wraddress_control = "REGISTERED";
parameter lpm_file = "UNUSED";
parameter lpm_hint = "UNUSED";

input [lpm_width-1:0] data;
input [lpm_widthad-1:0] rdaddress, wraddress;
input rdclock, wrclock, rdclken, wrclken, wren, rden;
output [lpm_width-1:0] q;
endmodule //lpm_ram_dp
```

2. Instantiate the LPM in the higher-level module. For example:

```
module top(d, q1, wclk, rclk, wraddr, raddr, wren, rden,
   wrclken, rdclken) ;
parameter AWIDTH = 4;
parameter DWIDTH = 8;
parameter WDEPTH = 1<<AWIDTH;

input [AWIDTH-1:0] wraddr, rdaddr;
input [DWIDTH-1:0] d;
input wclk, rclk, wren, rden;
input wrclken, rdclken;
output [DWIDTH-1:0] q1;
```

Copyright © 2013 Synopsys, Inc.

```
lpm_ram_dp u1(.data(d), .wrclock(wclk), .rdclock(rclk), .q(q1),
   .wraddress(wraddr), .rdaddress(rdaddr), .wren(wren),
   .rden(rden), .wrclken(wrclken), .rdclken(rdclken));
defparam u1.lpm_width = DWIDTH;
defparam u1.lpm_widthad = AWIDTH;
defparam u1.lpm_indata = "REGISTERED";
defparam u1.lpm_outdata = "REGISTERED";
defparam u1.lpm_wraddress_control = "REGISTERED";
defparam u1.lpm_rdaddress_control = "REGISTERED";
endmodule
```

For information about using the LPMs in Altera simulation flows, see Using LPMs in Simulation Flows, on page 767.

Copyright © 2013 Synopsys, Inc.

CHAPTER 8

Specifying Design-Level Optimizations

This chapter covers techniques for optimizing your design using built-in tools or attributes. For vendor-specific optimizations, see Chapter 15, *Optimizing for Specific Targets*. It describes the following:

- Tips for Optimization, on page 422
- Pipelining, on page 426
- Retiming, on page 430
- Preserving Objects from Optimization, on page 438
- Optimizing Fanout, on page 444
- Sharing Resources, on page 448
- Inserting I/Os, on page 452
- Optimizing State Machines, on page 454
- Inserting Probes, on page 462

Copyright © 2013 Synopsys, Inc.

Tips for Optimization

The software automatically makes efficient trade-offs to achieve the best results. However, you can optimize your results by using the appropriate control parameters. This section describes general design guidelines for optimization. The topics have been categorized as follows:

- General Optimization Tips, on page 422
- Optimizing for Area, on page 423
- Optimizing for Timing, on page 424

General Optimization Tips

This section contains general optimization tips that are not directly area or timing-related. For area optimization tips, see *Optimizing for Area*, on page 423. For timing optimization, see *Optimizing for Timing*, on page 424.

- In your source code, remove any unnecessary priority structures in timing-critical designs. For example, use `CASE` statements instead of nested `IF-THEN-ELSE` statements for priority-independent logic.
- If your design includes safe state machines, use the syn_encoding attribute with a value of safe. This ensures that the synthesized state machines never lock in an illegal state.
- For FSMs coded in VHDL using enumerated types, use the same encoding style (syn_enum_encoding attribute value) on both the state machine enumerated type and the state signal. This ensures that there are no discrepancies in the type of encoding to negatively affect the final circuit.
- Make sure that the source code supports inferencing or instantiation by using architecture-specific resources like memory blocks.
- Some designs benefit from hierarchical optimization techniques. To enable hierarchical optimization on your design, set the syn_hier attribute to firm.
- For accurate results with timing-driven synthesis, explicitly define clock frequencies with a constraint, instead of using a global clock frequency.

Copyright © 2013 Synopsys, Inc.

Optimizing for Area

This section contains information on optimizing to reduce area. Optimizing for area often means larger delays, and you will have to weigh your performance needs against your area needs to determine what works best for your design. For tips on optimizing for performance, see *Optimizing for Timing*, on page 424. General optimization tips are in *General Optimization Tips*, on page 422.

- Increase the fanout limit when you set the implementation options. A higher limit means less replicated logic and fewer buffers inserted during synthesis, and a consequently smaller area. In addition, as P&R tools typically buffer high fanout nets, there is no need for excessive buffering during synthesis. See *Setting Fanout Limits*, on page 444 for more information.
- Enable the Resource Sharing option when you set implementation options. With this option checked, the software shares hardware resources like adders, multipliers, and counters wherever possible, and minimizes area. This is a global setting, but you can also specify resource sharing on an individual basis for lower-level modules. See *Sharing Resources*, on page 448 for details.
- For designs with large FSMs, use the gray or sequential encoding styles, because they typically use the least area. For details, see *Specifying FSMs with Attributes and Directives*, on page 343.
- If you are mapping into a CPLD and do not meet area requirements, set the default encoding style for FSMs to sequential instead of onehot. For details, see *Specifying FSMs with Attributes and Directives*, on page 343.
- For small CPLD designs (less than 20K gates), you might improve area by using the syn_hier attribute with a value of flatten. When specified, the software optimizes across hierarchical boundaries and creates smaller designs.

Copyright © 2013 Synopsys, Inc.

Optimizing for Timing

This section contains information on optimizing to meet timing requirements. Optimizing for timing is often at the expense of area, and you will have to balance the two to determine what works best for your design. For tips on optimizing for area, see *Optimizing for Area*, on page 423. General optimization tips are in *General Optimization Tips*, on page 422.

- Use realistic design constraints, about 10 to 15 percent of the real goal. Over-constraining your design can be counter-productive because you can get poor implementations. Typically, you set timing constraints like clock frequency, clock-to-clock delay paths, I/O delays, register I/O delays and other miscellaneous path delays. Use clock, false path, and multi-cycle path constraints to make the constraints realistic.
- Enable the Retiming option. This optimization moves registers into I/O buffers if this is permitted by the technology and the design. However, it may add extra registers when clouds of logic are balanced across more than one register-to-register timing path. Extra registers are only added in parallel within the timing path and only if no extra latency is added by the additional registers. For example, if registers are moved across a 2x1 multiplexer, the tool adds two new registers to accommodate the select and data paths.

  You can set this option globally or on specific registers. When it is enabled, it automatically enables pipelining as well. See *Retiming*, on page 430 for details.
- Enable the Pipelining option. With this optimization enabled, the tool moves existing registers into a ROM or multiplier. Unlike retiming, it does not add any new logic. Pipelining reduces routing and delay and the extra instance delay of the external register by moving it into the ROM or multiplier and making it a built-in register.
- Select a balanced fanout constraint. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated logic. See *Setting Fanout Limits*, on page 444 for information about setting limits and using the syn_maxfan attribute. You can use this in conjunction with the syn_replicate attribute that controls register duplication and buffering.
- Control register duplication and buffering criteria with the syn_replicate attribute. The tool automatically replicates registers during optimization, and you can use this attribute globally or locally on a specific register to turn off register duplication. See *Controlling Buffering and Replication*,

on page 446 for a description. Use syn_replicate in conjunction with the syn_maxfan attribute that controls fanout.

- If the critical path goes through arithmetic components, try disabling Resource Sharing. You can get faster times at the expense of increased area, but use this technique carefully. Adding too many resources can cause longer delays and defeat your purpose.
- If the P&R and synthesis tools report different critical paths, use a timing constraint with the -route option. With this option, the software adds route delay to its calculations when trying to meet the clock frequency goal. Use realistic values for the constraints.
- For FSMs, use the onehot encoding style, because it is often the fastest implementation. If a large output decoder follows an FSM, gray or sequential encoding could be faster.
- For designs with black boxes, characterize the timing models accurately, using the syn_tpd, syn_tco, and syn_tso directives.
- If you see warnings about feedback muxes being created for signals when you compile your source code, make sure to assign set/resets for the signals. This improves performance by eliminating the extra mux delay on the input of the register.
- Make sure that you pass your timing constraints to the place-and-route tools, so that they can use the constraints to optimize timing.
- If you are working in the Synplify Premier tool and performance and quality of results (QoR) are not essential to the application (as with early prototyping and “what if” scenarios), use the Fast Synthesis option. For details, refer to Chapter 19, *Fast Synthesis*.

  The Fast Synthesis option reduces the amount and number of mapper optimizations performed so that you get faster synthesis runtimes. Once the design has been evaluated with fast synthesis, the mapper optimization effort can be returned to its normal, default level for optimum performance/QoR evaluations. This option is only available with the Synplify Premier tool for devices from the Xilinx Virtex and Spartan families or the Altera Stratix families.

Copyright © 2013 Synopsys, Inc.

Pipelining

Synplify Pro and Synplify Premier tools
Pipelining is the process of splitting logic into stages so that the first stage can begin processing new inputs while the last stage is finishing the previous inputs. This ensures better throughput and faster circuit performance. If you are using selected technologies which use pipelining, you can also use the related technique of retiming to improve performance. See *Retiming*, on page 430 for details.

For pipelining, The software splits the logic by moving registers into the multiplier or ROM:

This section discusses the following pipelining topics:

- Prerequisites for Pipelining, on page 426
- Pipelining the Design, on page 427

Prerequisites for Pipelining

The pipelining feature is only available in the Synplify Pro and Synplify Premier tools.

- Currently, pipelining is only supported for certain Microsemi, Altera, Lattice, SiliconBlue, and Xilinx technologies.
- In Xilinx Virtex device families, you can pipeline ROMs and multipliers. In Altera designs, you can pipeline multipliers, but not ROMs.
- For Xilinx Virtex device families, you can only pipeline multipliers if the adjacent register has a synchronous reset.
- ROMs to be pipelined must be at least 512 words. Anything below this limit is too small.
- For Xilinx Virtex device families, you can push any kind of flip-flop into the module, as long as all the flip-flops in the pipeline have the same clock, the same set/reset signal or lack of it, and the same enable control or lack of it. For Altera designs, you must have asynchronous set/resets if you want to do pipelining.

Copyright © 2013 Synopsys, Inc.

Pipelining the Design

The following procedure shows you techniques for pipelining.

1. Make sure the design meets the criteria described in *Prerequisites for Pipelining*, on page 426.

2. To enable pipelining for the whole design, check the Pipelining check box. from the button panel in the Project window, or with the Project->Implementation Options command (Options tab). The option is only available in the appropriate technologies.

   Use this approach as a first pass to get a feel for which modules you can pipeline. If you know exactly which registers you want to pipeline, add the attribute to the registers in the source code or interactively using the SCOPE interface.

3. To check whether individual registers are suitable for pipelining, do the following:
   - Open the RTL view of the design.
   - Select the register and press F12 to filter the schematic view.

Copyright © 2013 Synopsys, Inc.

– In the new schematic view, select the output and type e (or select Expand from the popup menu. Check that the register is suitable for pipelining.

4. To enable pipelining on selected registers, use either of the following techniques:

   – Check the Pipelining checkbox and attach the syn_pipeline attribute with a value of 0 or false to any registers you do not want the software to move. This attribute specifies that the register cannot be moved for pipelining.

   – Do not check the Pipelining checkbox. Attach the syn_pipeline attribute with a value of 1 or true to any registers you want the software to consider for retiming. This attribute marks the register as one that can be moved during retiming, but does not necessarily force it to be moved during retiming.

   The following are examples of the attribute:

   SCOPE Interface:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Descr
1	✔	register	res[15:0]	syn_pipeline	1	boolean	Controls pipelining c

   Verilog Example:

```
reg [`lefta:0] a_aux;
reg [`leftb:0] b_aux;
reg [`lefta+`leftb+1:0] res /* synthesis syn_pipeline=1 */;
reg [`lefta+`leftb+1:0] res1;
```

   VHDL Example:

```
architecture beh of onereg is
signal temp1, temp2, temp3,
   std_logic_vector(31 downto 0);
attribute syn_pipeline : boolean;
attribute syn_pipeline of temp1 : signal is true;
attribute syn_pipeline of temp2 : signal is true;
attribute syn_pipeline of temp3 : signal is true;
```

5. Click Run.

   The software looks for registers where all the flip-flops of the same row have the same clock, no control signal, or the same unique control signal, and pushes them inside the module. It attaches the syn_pipeline attribute to all these registers. If there already is a syn_pipeline attribute on a register, the software implements it.

6. Check the log file (`*.srr`). You can use the Find command for occurrences of the word pipelining to find out which modules got pipelined.

   The log file entries look like this:

```
@N:|Pipelining module res_out1
@N:|res_i is level 1 of the pipelined module res_out1
@N:|r is level 2 of the pipelined module res_out1
```


Copyright © 2013 Synopsys, Inc.

Retiming

(Synplify Pro, Synplify Premier). Some Altera, Lattice, Microsemi, and Xilinx technologies.
Retiming improves the timing performance of sequential circuits without modifying the source code. It automatically moves registers (register balancing) across combinatorial gates or LUTs to improve timing while maintaining the original behavior as seen from the primary inputs and outputs of the design. Retiming moves registers across gates or LUTs, but does not change the number of registers in a cycle or path from a primary input to a primary output. However, it can change the total number of registers in a design.

The retiming algorithm retimes only edge-triggered registers. It does not retime level-sensitive latches. Note that registers associated with RAMS, DSPs, and the mapping for generated clocks may be moved, regardless of the Retiming option setting. The Retiming option is not available if it does not apply to the family you are using.

These sections contain details about using retiming.

- Controlling Retiming, on page 430
- Retiming Example, on page 432
- Retiming Report, on page 433
- How Retiming Works, on page 434

Controlling Retiming

The following procedure shows you how to use retiming.

1. To enable retiming for the whole design, check the Retiming check box.

   You can set the Retiming option from the button panel in the Project window, or with the Project->Implementation Options command (Options tab). The option is only available in certain technologies.

For Altera, SiliconBlue, and Xilinx designs, retiming is a superset of pipelining, so when you select Retiming, you automatically select Pipelining. See *Pipelining*, on page 426 for more information. For Microsemi designs, retiming does not include pipelining.

Retiming works globally on the design, and moves edge-triggered registers as needed to balance timing.

2. To enable retiming on selected registers, use either of the following techniques:
   - Check the Retiming checkbox and attach the syn_allow_retiming attribute with a value of 0 or false to any registers you do not want the software to move. This attribute specifies that the register cannot be moved for retiming. Refer to *How Retiming Works*, on page 434 for a list of the components the retiming algorithm will move.
   - Do not check the Retiming checkbox. Attach the syn_allow_retiming attribute with a value of 1 or true to any registers you want the software to consider for retiming. You can do this in the SCOPE interface or in the source code. This attribute marks the register as one that can be moved during retiming, but does not necessarily force it to be moved during retiming. If you apply the attribute to an FSM, RAM or SRL that is decomposed into flip-flops and logic, the software applies the attribute to all the resulting flip-flops

Copyright © 2013 Synopsys, Inc.

Retiming is a superset of pipelining; therefore adding syn_allow_retiming=1 on any registers implies that syn_pipeline =1.

3. You can also fine-tune retiming using attributes:
   - To preserve the power-on state of flip-flops without sets or resets (FD or FDE) during retiming, set syn_preserve=1 or syn_allow_retiming=0 on these flip-flops.
   - To force flip-flops to be packed in I/O pads, set syn_useioff=1 as a global attribute. This will prevent the flip-flops from being moved during retiming.
4. Set other options for the run. Retiming might affect some constraints and attributes. See *How Retiming Works*, on page 434 for details.
5. Click Run to start synthesis.

   After the LUTs are mapped, the software moves registers to optimize timing. See *Retiming Example*, on page 432 for an example. The software honors other attributes you set, like syn_preserve, syn_useioff, and syn_ramstyle. See *How Retiming Works*, on page 434 for details.

   Note that the tool might retime registers associated with RAMs, DSPs, and generated clocks, regardless of whether the Retiming option is on or off.

   The log file includes a retiming report that you can analyze to understand the retiming changes. It contains a list of all the registers added or removed because of retiming. Retimed registers have a _ret suffix added to their names. See *Retiming Report*, on page 433 for more information about the report.

Retiming Example

The following example shows a design with retiming disabled and enabled.

The top figure shows two levels of logic between the registers and the output, and no levels of logic between the inputs and the registers.

The bottom figure shows the results of retiming the three registers at the input of the OR gate. The levels of logic from the register to the output are reduced from two to one. The retimed circuit has better performance than the original circuit. Timing is improved by transferring one level of logic from the critical part of the path (register to output) to the non-critical part (input to register).

Retiming Report

The retiming report is part of the log file, and includes the following:

- The number of registers added, removed, or untouched by retiming.
- Names of the original registers that were moved by retiming and which no longer exist in the Technology view.
- Names of the registers created as a result of retiming, and which did not exist in the RTL view. The added registers have a _ret suffix.

Copyright © 2013 Synopsys, Inc.

How Retiming Works

This section describes how retiming works when it moves sequential components (flip-flops). Registers associated with RAMs, DSPs, and the mapping for fixing generated clocks might be moved, whether Retiming is enabled or not. Here are some implications and results of retiming:

- Flip-flops with no control signals (resets, presets, and clock enables) are moved. Flip-flops with minimal control logic can also be retimed. Multiple flip-flops with reset, set or enable signals that need to be retimed together are only retimed if they have exactly the same control logic.
- The software does not retime the following combinatorial sequential elements: flip-flops with both set and reset, flip-flops with attributes like syn_preserve, flip-flops packed in I/O pads, level-sensitive latches, registers that are instantiated in the code, SRLs, and RAMs. If a RAM with combinatorial logic has syn_ramstyle set to registers, the registers can be retimed into the combinatorial logic.
- Retimed flip-flops are only moved through combinatorial logic. The software does not move flip-flops across the following objects: black boxes, sequential components, tristates, I/O pads, instantiated components, carry and cascade chains, and keepbufs. For Altera designs, registers that are in counter modes are not retimed to preserve the performance benefit of the counter mode.
- You might not be able to crossprobe retimed registers between the RTL and the Technology view, because there may not be a one-to-one correspondence between the registers in these two views after retiming. A single register in the RTL view might now correspond to multiple registers in the Technology view.
- Retiming affects or is affected by, these attributes and constraints:

Copyright © 2013 Synopsys, Inc.

Attribute/Constraint	Effect
False path constraint	Does not retime flip-flops with different false path constraints. Retimed registers affect timing constraints.
Multicycle constraint	Does not retime flip-flops with different multicycle constraints. Retimed registers affect timing constraints.
Register constraint	Does not maintain define_reg_input_delay and define_reg_output_delay constraints. Retimed registers affect timing constraints.
from/to timing exceptions	If you set a timing constraint using a from/to specification on a register, it is not retimed. The exception is when using a max_delay constraint. In this case, retiming is performed but the constraint is not forward annotated. (The max_delay value would no longer be valid.)
syn_hier=macro	Does not retime registers in a macro with this attribute.
syn_keep	Does not retime across keepbufs generated because of this attribute.
syn_hier=macro	Does not retime registers in a macro with this attribute.
syn_pipeline	Automatically enabled if retiming is enabled.
syn_preserve	Does not retime flip-flops with this attribute set.
syn_probe	Does not retime net drivers with this attribute. If the net driver is a LUT or gate, no flip-flops are retimed across it.
syn_reference_clock	On a critical path, does not retime registers with different syn_reference_clock values together, because the path effectively has two different clock domains.
syn_useioff	Does not override attribute-specified packing of registers in I/O pads. If the attribute value is false, the registers can be retimed. If the attribute is not specified, the timing engine determines whether the register is packed into the I/O block.
syn_allow_retiming	Registers are not retimed if the value is 0.

Copyright © 2013 Synopsys, Inc.

- Retiming does not change the simulation behavior (as observed from primary inputs and outputs) of your design, However if you are monitoring (probing) values on individual registers inside the design, you might need to modify your test bench if the probe registers are retimed.
- Beginning with the C-2009.09-SP1 release, the behavior for retiming unconstrained I/O pads has changed. If retiming is enabled, registers connected to unconstrained I/O pins are not retimed by default. If you want to revert back to how retiming I/O paths was previously implemented, you can:
  - Globally turn on the Use clock period for unconstrained IO switch from the Constraints tab of the Implementation Options panel.
  - Add constraints to all input/output ports.
  - Separately constrain each I/O pin as required.

Copyright © 2013 Synopsys, Inc.

How Retiming Works With Synplify Premier Regions

The following conditions can occur after a register has been retimed:

- If the retimed register and its driver and load remain in a Synplify Premier-specific region, then the register will remain in the region.
- If the retimed register is moved outside of a Synplify Premier-specific region but its load remains in the region, then the register will remain in the region.
- If the retimed register and its driver and load are moved outside a Synplify Premier-specific region, then the register will be moved outside the region.
- If the retimed register is moved to the boundary of a Synplify Premier-specific region, then tunneling can occur.
- Retiming may move a register across a Synplify Premier-specific region but not across combinatorial logic.

Copyright © 2013 Synopsys, Inc.

Preserving Objects from Optimization

Synthesis can collapse or remove nets during optimization. If you want to retain a net for simulation, probing, or for a different synthesis implementation, you must specify this with an attribute. Similarly, the software removes duplicate registers or instances with unused output. If you want to preserve this logic for simulation or analysis, you must use an attribute. The following table lists the attributes to use in each situation. For details about the attributes and their syntax, see the *Reference Manual*.

To Preserve...	Attach...	Result
Nets	syn_keep on wire or reg (Verilog), or signal (VHDL). For Microsemi designs (except 500K and PA), use alspreserve as well as syn_keep.	Keeps net for simulation, a different synthesis implementation, or for passing to the place-and-route tool.
Nets for probing	syn_probe on wire or reg (Verilog), or signal (VHDL)	Preserves internal net for probing. This attribute is only applicable to the Synplify Pro and Synplify Premier software.
Shared registers	syn_keep on input wire or signal of shared registers	Preserves duplicate driver cells, prevents sharing. See *Using syn_keep for Preservation or Replication*, on page 439 for details on the effects of applying syn_keep to different objects.
Sequential components	syn_preserve on reg or module (Verilog), signal or architecture (VHDL)	Preserves logic of constant-driven registers, keeps registers for simulation, prevents sharing
FSMs	syn_preserve on reg or module (Verilog), signal (VHDL)	Prevents the output port or internal signal that holds the value of the state register from being optimized
Instantiated components	syn_noprune on module or component (Verilog), architecture or instance (VHDL)	Keeps instance for analysis, preserves instances with unused outputs

Copyright © 2013 Synopsys, Inc.

See the following for more information:

- Using syn_keep for Preservation or Replication, on page 439
- Controlling Hierarchy Flattening, on page 442
- Preserving Hierarchy, on page 442

Using syn_keep for Preservation or Replication

By default the tool considers replicated logic redundant, and optimizes it away. If you want to maintain the redundant logic, use syn_keep to preserve the logic that would otherwise be optimized away.

The following Verilog code specifies a replicated AND gate:

```
module redundant1(ina,inb,out1);
input  ina,inb;
output out1,out2;
wire out1;
wire out2;

assign out1 = ina & inb;
assign out2 = ina & inb;;
endmodule
```

The compiler implements the AND function by replicating the outputs out1 and out2, but optimizes away the second AND gate because it is redundant.

Copyright © 2013 Synopsys, Inc.

To replicate the AND gate in the previous example, apply syn_keep to the input wires, as shown below:

```
module redundant1d(ina,inb,out1,out2);
input ina,inb;
output out1,out2;
wire out1;
wire out2;

wire in1a /*synthesis syn_keep = 1*/;
wire in1b /*synthesis syn_keep = 1*/;
wire in2a /*synthesis syn_keep = 1*/;
wire in2b /*synthesis syn_keep = 1 */;

assign in1a = ina ;
assign in1b = inb  ;
assign in2a = ina;
assign in2b = inb;
assign out1 = in1a & in1b;
assign out2 = in2a & in2b;
endmodule
```

Setting syn_keep on the input wires ensures that the second AND gate is preserved:

You must set syn_keep on the input wires of an instance if you want to preserve the logic, as in the replication of this AND gate. If you set it on the outputs, the instance is not replicated, because syn_keep preserves the nets but not the function driving the net. If you set syn_keep on the outputs in the example, you get only one AND gate, as shown in the next figure.

Copyright © 2013 Synopsys, Inc.

Controlling Hierarchy Flattening

Optimization flattens hierarchy. To control the flattening, use the syn_hier attribute as described here. You can also use the attribute to prevent flattening, as described in *Preserving Hierarchy*, on page 442.

1. Attach the syn_hier attribute with the value you want to the module or architecture you want to preserve.

To...	Value...
Flatten all levels below, but not the current level	flatten
Remove the current level of hierarchy without affecting the lower levels	remove
Remove the current level of hierarchy and the lower levels	flatten, remove
Flatten the current level (if needed for optimization)	soft

   You can also add the attribute in SCOPE instead of the HDL code. If you use SCOPE to enter the attribute, make sure to use the v: syntax. For details, see *syn_hier*, on page 1344 in the *Reference Manual*.

   The software flattens the design as directed. If there is a lower-level syn_hier attribute, it takes precedence over a higher-level one.

2. If you want to flatten the entire design, use the syn_netlist_hierarchy attribute set to false, instead of the syn_hier attribute.

   This flattens the entire netlist and does not preserve any hierarchical boundaries. See *syn_netlist_hierarchy*, on page 1413 in the *Reference Manual* for the syntax.

Preserving Hierarchy

The synthesis process includes cross-boundary optimizations that can flatten hierarchy. To override these optimizations, use the syn_hier attribute as described here. You can also use this attribute to direct the flattening process as described in *Controlling Hierarchy Flattening*, on page 442.

1. Attach the syn_hier attribute to the module or architecture you want to preserve. You can also add the attribute in SCOPE. If you use SCOPE to enter the attribute, make sure to use the v: syntax.

2. Set the attribute value:

To...	Value...
Preserve the interface but allow cell packing across the boundary	firm
Preserve the interface with no exceptions (Microsemi, Altera, and Xilinx only)	hard
Preserve the interface and contents with no exceptions (Microsemi (except PA, 500K, and ProASIC3 families), Altera, Lattice, and QuickLogic only)	macro
Flatten lower levels but preserve the interface of the specified design unit	flatten, firm

The software flattens the design as directed. If there is a lower-level syn_hier attribute, it takes precedence over a higher-level one.

Copyright © 2013 Synopsys, Inc.

Optimizing Fanout

You can optimize your results with attributes and directives, some of which are specific to the technology you are using. Similarly, you can use specify objects or hierarchy that you want to preserve during synthesis. For a complete list of all the directives and attributes, see the *Reference Manual*. This section describes the following:

- Setting Fanout Limits, on page 444
- Controlling Buffering and Replication, on page 446

Setting Fanout Limits

Optimization affects net fanout. If your design has critical nets with high fanout, you can set fanout limits. You can only do this in certain technologies. For details specific to individual technologies, see the *Reference Manual*.

1. To set a global fanout limit for the whole design, do either of the following:
   - Select Project-> Implementation Options->Device and type a value for the Fanout Guide option.
   - Apply the syn_maxfan attribute to the top-level view or module.

   The value sets the number of fanouts for a given driver, and affects all the nets in the design. The defaults vary, depending on the technology. Select a balanced fanout value. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated or buffered logic. Both extremes affect routing and design performance. The right value depends on your design. The same value of 32 might result in fanouts of 11 or 12 and large delays on the critical path in one design or in excessive replication in another design.

   The software uses the value as a soft limit, or a guide. It traverses the inverters and buffers to identify the fanout, and tries to ensure that all fanouts are under the limit by replicating or buffering where needed (see *Controlling Buffering and Replication*, on page 446 for details). However, the synthesis tool does not respect the fanout limit absolutely; it ignores the limit if the limit imposes constraints that interfere with optimization.

Copyright © 2013 Synopsys, Inc.

2. For certain Microsemi technologies, you can set a global hard fanout limit by doing the following:
   - Select Project-> Implementation Options->Device and type a value for the Fanout Guide option, as described in the previous step.
   - On the same tab, check the Hard Fanout Limit option.

   This makes the specified value a global hard fanout limit for the design.

3. To override the global fanout guideline and set a soft fanout limit at a lower level, set the syn_maxfan attribute on modules, views, or non-primitive instances.

   These limits override the more global limits for that object (including a global hard limit in Microsemi technologies). However, these limits still function as soft limits, and are replicated or buffered, as described in *Controlling Buffering and Replication*, on page 446.

Attribute specified on...	Effect
Module or view	Soft limit for the module; overrides the global setting.
Non-primitive instance	Soft limit; overrides global and module settings
Clock nets or asynchronous control nets	Soft limit.

4. To set a hard or absolute limit, set the syn_maxfan attribute on a port, net, register, or primitive instance.

   Fanouts that exceed the hard limit are buffered or replicated, as described in *Controlling Buffering and Replication*, on page 446.

5. To preserve net drivers from being optimized, attach the syn_keep or syn_preserve attributes.

   For example, the software does not traverse a syn_keep buffer (inserted as a result of the attribute), and does not optimize it. However, the software can optimize implicit buffers created as a result of other operations; for example, it does not respect an implicit buffer created as a result of syn_direct_enable.

Copyright © 2013 Synopsys, Inc.

6. Check the results of buffering and replication in the following:
   - The log file (click View Log). The log file reports the number of buffered and replicated objects and the number of segments created for the net.
   - The HDL Analyst views. The software might not follow DRC rules when buffering or replicating objects, or when obeying hard fanout limits.

Controlling Buffering and Replication

To honor fanout limits (see *Setting Fanout Limits*, on page 444) and reduce fanout, the software either replicates components or adds buffers. The tool uses buffering to reduce fanout on input ports, and uses replication to reduce fanout on nets driven by registers or combinatorial logic. The software first tries replication, replicating the net driver and splitting the net into segments. This increases the number of register bits in the design. When replication is not possible, the software buffers the signals. Buffering is more expensive in terms of intrinsic delay and resource consumption. The following table summarizes the behavior.

Replicates When...	Creates Buffers When...
syn_maxfan is set on a register output	syn_maxfan is set on input ports in Altera Apex, Microsemi ProASIC (500K), ProASIC PLUS (PA) and ProASIC3 families, and QuickLogic pASIC3 designs
syn_replicate is 1	syn_replicate is 0. Note that the syn_replicate attribute must be used in conjunction with the syn_maxfan attribute for Microsemi families. The syn_replicate attribute is used only to turn off the replication.
	syn_maxfan is set on a port/net that is driven by a port or I/O pad
	The net driver has a syn_keep or syn_preserve attribute
	The net driver is not a primitive gate or register

You can control whether high fanout nets are buffered or replicated, using the techniques described here:

- To use buffering instead of replication, set syn_replicate with a value of 0 globally, or on modules or registers. The syn_replicate attribute prevents replication, so that the software uses buffering to satisfy the fanout limit. For example, you can prevent replication between clock boundaries for a register that is clocked by clk1 but whose fanin cone is driven by clk2, even though clk2 is an unrelated clock in another clock group.
- To specify that high-fanout clock ports should not be buffered, set syn_noclockbuf globally, or on individual input ports. Use this if you want to save clock buffer resources for nets with lower fanouts but tighter constraints.
- Inverters merged with fanout loads increase fanout on the driver during placement and routing. A distinction is made between a keep buffer created as the result of the syn_keep attribute being applied by the user (explicit keep buffer) and a keep buffer that exists as the result of another attribute (implicit keep buffer). For example, the syn_direct_enable attribute inserts a keep buffer. When a syn_maxfan attribute is applied to the output of an explicit keep buffer, the signal is buffered (the keep buffer is not traversed so that the driver is not replicated). When the syn_maxfan attribute is applied to the output of an implicit keep buffer, the keep buffer is traversed and the driver is replicated.
- In Xilinx designs, you can handle extremely large clock fanout nets by inserting a global buffer (BUFG) in your design. A global buffer reduces delay for a large fanout net and can free up routing resources for other signals.
- Turn off buffering and replication entirely, by setting syn_maxfan to a very high number, like 1000.

Copyright © 2013 Synopsys, Inc.

Sharing Resources

One of the ways you can optimize area is to use resource sharing. With resource sharing, the software uses the same arithmetic operators for mutually exclusive statements; for example, with the branches of a case statement. Conversely, you can improve timing by disabling resource sharing, but at the expense of increased area.

1. Specify resource sharing globally for the whole design, using one of the methods below. Enable the option to improve area; disable it to improve timing.
   - Select Project->Implementation Options->Options, and enable or disable Resource Sharing. Alternatively, enable the Resource Sharing button on the left side of the Project view.
   - Apply the syn_sharing directive to the top-level module or architecture in the source code. See syn_sharing, on page 1518 of the *Reference Manual* for details of the syntax.

Verilog
```
module top(out, in, clk_in) /* synthesis syn_sharing = "on" */;
```

VHDL
```
architecture rtl of top is
  attribute syn_sharing : string;
  attribute syn_sharing of rtl : architecture is "off";
```

   - Edit your project file and include the following command. 0 disables and 1 enables resource sharing:

     ```
     set_option -resource_sharing 1|0
     ```

     When you save the project file, it includes the Tcl `set_option -resource_sharing` command.

     You cannot specify syn_sharing from the SCOPE interface, because it is a compiler directive, and works during the compilation stage of synthesis. The mapper does not consider the resource sharing setting, so even if resource sharing is disabled, it can perform resource sharing optimizations to improve results.

2. To specify resource sharing on an individual basis, or to override the global setting, specify the syn_sharing attribute for the lower-level module/architecture, using the syntax described in the previous step.

The following examples illustrate resource sharing.

Copyright © 2013 Synopsys, Inc.

Verilog Example

The following example illustrates resource sharing in Verilog. The first diagram shows the equivalent logic with resource sharing enabled, and the second diagram shows the same logic with resource sharing disabled.

```
module add (a, b, x, y, out1, out2, sel, en, clk);
input a, b, x, y, sel, en, clk;
output out1, out2;
wire tmp1, tmp2;
assign tmp1 = a * b;
assign tmp2 = x * y;
reg out1, out2;

always@(posedge clk)
   if (en)
      begin
         out1 <= sel ?  tmp1: tmp2;
      end
   else
      begin
         out2 <= sel ? tmp1: tmp2;
   end
endmodule
```

Copyright © 2013 Synopsys, Inc.

VHDL Example

The following example illustrates resource sharing in VHDL. The first diagram shows the equivalent logic with resource sharing enabled, and the second diagram shows the same logic with resource sharing disabled.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add is
   port (a, b : in std_logic_vector(1 downto 0);
         x, y : in std_logic_vector(1 downto 0);
         clk, sel, en: in std_logic;
         out1 : out std_logic_vector(3 downto 0);
         out2 : out std_logic_vector(3 downto 0)
   );
end add;

architecture rtl of add is
signal tmp1, tmp2: std_logic_vector(3 downto 0);
begin
   tmp1 <= a * b;
   tmp2 <= x * y;
```

```
process(clk) begin
   if clk'event and clk='1' then
      if (en='1') then
         if (sel='1') then
            out1 <= tmp1;
         else
            out1 <= tmp2;
         end if;
      else
         if (sel='1') then
            out2 <= tmp1;
         else
            out2 <= tmp2;
         end if;
      end if;
   end if;
end process;
end rtl;
```

Copyright © 2013 Synopsys, Inc.

Inserting I/Os

You can control I/O insertion globally, or on a port-by-port basis.

1. To control the insertion of I/O pads at the top level of the design, use the Disable I/O Insertion option as follows:
   - Select Project->Implementation Options and click the Device panel.
   - Enable the option (checkbox on) if you want to do a preliminary run and check the area taken up by logic blocks, before synthesizing the entire design.

     Do this if you want to check the area your blocks of logic take up, before you synthesize an entire FPGA. If you disable automatic I/O insertion, you do not get *any* I/O pads in your design, unless you manually instantiate them.
   - Leave the Disable I/O Insertion checkbox empty (disabled) if you want to automatically insert I/O pads for all the inputs, outputs and bidirectionals.

     When this option is set, the software inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist. Once inserted, you can override the I/O pad inserted by directly instantiating another I/O pad.

– For the most control, enable the option and then manually instantiate the I/O pads for specific pins, as needed.

For more information about using the Disable I/O Insertion option with SiliconBlue and Xilinx devices, see *syn_insert_pad*, on page 1376.

2. For Lattice designs, you can force I/O pads to be inserted for input ports that do not drive logic with the syn_force_pads attribute:

   – To force I/O pad insertion at the module level, set the syn_force_pads attribute on the module. Set the attribute value to 1. To disable I/O pad insertion at the module level, set the syn_force_pads attribute for the module to 0.

   – To force I/O pad insertion on an individual port, set the syn_force_pads attribute on the port with a value to 1. To disable I/O insertion for a port, set the attribute on the port with a value of 0.

   Enable this attribute to preserve user-instantiated pads, insert pads on unconnected ports, insert bi-directional pads on bi-directional ports instead of converting them to input ports, or insert output pads on unconnected outputs.

   If you do not set the syn_force_pads attribute, the synthesis design optimizes any unconnected I/O buffers away.

Copyright © 2013 Synopsys, Inc.

Optimizing State Machines

You can optimize state machines with the symbolic FSM Compiler and the FSM Explorer tools.

- The Symbolic FSM Compiler
  An advanced state machine optimizer, it automatically recognizes state machines in your design and optimizes them. Unlike other synthesis tools that treat state machines as regular logic, the FSM Compiler extracts the state machines as symbolic graphs, and then optimizes them by re-encoding the state representations and generating a better logic optimization starting point for the state machines.
- The FSM Explorer
  A specialized state machine optimizer that explores different encoding styles before selecting the best style. It uses the FSM Compiler to extract state machines, and runs the FSM Compiler automatically if it has not been run. The FSM Explorer functionality is only available in the Synplify Pro and Synplify Premier tools.

For more information, see the following:

- Deciding when to Optimize State Machines, on page 454
- Running the FSM Compiler, on page 455
- Running the FSM Explorer, on page 459

Deciding when to Optimize State Machines

The FSM Explorer and the FSM Compiler are automatic tools for encoding state machines, but you can also specify FSMs manually with attributes. For more information about using attributes, see *Specifying FSMs with Attributes and Directives*, on page 343.

Here are the main reasons to use the FSM Compiler:

- To generate better results for your state machines

  The software uses optimization techniques that are specifically tuned for FSMs, like reachability analysis for example. The FSM Compiler also lets you convert an encoded state machine to another encoding style (to improve speed and area utilization) without changing the source. For example, you can use a onehot style to improve results.

- To debug the state machines

  State machine description errors result in unreachable states, so if you have errors, you will have fewer states. You can check whether your source code describes your state machines correctly. You can also use the FSM Viewer to see a high-level bubble diagram and crossprobe from there. The FSM Viewer is only available in the Synplify Pro and Synplify Premier tools. For information about the FSM Viewer, see *Using the FSM Viewer*, on page 606.

- To run the FSM Explorer

  The FSM Explorer is a tool that examines all the encoding styles before selecting the best option, based on the state machine extraction done by the FSM Compiler. If the FSM Compiler has not been run previously, the Explorer automatically runs it. For more information about using the FSM Explorer, see *Running the FSM Explorer*, on page 459.

If you are trying to decide whether to use the FSM Compiler or the FSM Explorer to optimize your state machines, remember these points:

- The FSM Explorer runs the FSM Compiler if it has not already been run, because it picks encoding styles based on the state machines that the FSM Compiler extracts.
- Like the FSM Compiler, you use the FSM Explorer to generate better results for your state machines. Unlike the FSM Compiler, which picks an encoding style based on the number of states, the FSM Explorer tries out different encoding styles and picks the best style for the state machine based on overall design constraints.
- The trade-off is that the FSM Explorer takes longer to run than the FSM Compiler.

Running the FSM Compiler

You can run the FSM Compiler tool on the whole design or on individual FSMs. See the following:

- Running the FSM Compiler on the Whole Design, on page 456
- Running the FSM Compiler on Individual FSMs, on page 457

Copyright © 2013 Synopsys, Inc.

Running the FSM Compiler on the Whole Design

1. Enable the compiler by checking the Symbolic FSM Compiler box in one of these places:
   - The main panel on the left side of the project window
   - The Options tab of the dialog box that comes up when you click the Add Implementation/New Impl or Implementation Options buttons
2. To set a specific encoding style for a state machine, define the style with the syn_encoding attribute, as described in *Specifying FSMs with Attributes and Directives*, on page 343.

   If you do not specify a style, the FSM Compiler picks an encoding style based on the number of states.
3. Click Run to run synthesis.

   The software automatically recognizes and extracts the state machines in your design, and instantiates a state machine primitive in the netlist for each FSM it extracts. It then optimizes all the state machines in the design, using techniques like reachability analysis, next state logic optimization, state machine re-encoding and proprietary optimization algorithms. Unless you specified an encoding style, the tool automatically selects the encoding style. If you did specify a style, the tool uses that style.

   In the log file, the FSM Compiler writes a report that includes a description of each state machine extracted and the set of reachable states for each state machine.
4. Select View->View Log File and check the log file for descriptions of the state machines and the set of reachable states for each one. You see text like the following:

```
Extracted state machine for register cur_state
State machine has 7 reachable states with original encodings of:
   0000001
   0000010
   0000100
   0001000
   0010000
   0100000
   1000000
....
original code -> new code
```

```
0000001 -> 0000001
0000010 -> 0000010
0000100 -> 0000100
0001000 -> 0001000
0010000 -> 0010000
0100000 -> 0100000
1000000 -> 1000000
```

5. Check the state machine implementation in the RTL and Technology views and in the FSM viewer.
   - In the RTL view you see the FSM primitive with one output for each state.
   - In the Technology view, you see a level of hierarchy that contains the FSM, with the registers and logic that implement the final encoding.
   - In the FSM viewer you see a bubble diagram and mapping information. For information about the FSM viewer, see *Using the FSM Viewer*, on page 606.
   - In the `statemachine.info` text file, you see the state transition information.

Running the FSM Compiler on Individual FSMs

If you have state machines that you do not want automatically optimized by the FSM Compiler, you can use one of these techniques, depending on the number of FSMs to be optimized. You might want to exclude state machines from automatic optimization because you want them implemented with a specific encoding or because you do not want them extracted as state machines. The following procedure shows you how to work with both cases.

1. If you have just a few state machines you do not want to optimize, do the following:
   - Enable the FSM Compiler by checking the box in the button panel of the Project window.
   - If you do not want to optimize the state machine, add the syn_state_machine directive to the registers in the Verilog or VHDL code. Set the value to 0. When synthesized, these registers are not extracted as state machines.

Copyright © 2013 Synopsys, Inc.

Verilog	`reg [3:0] curstate /* synthesis syn_state_machine=0 */ ;`

VHDL	`signal curstate : state_type;`
	`attribute syn_state_machine : boolean;`
	`attribute syn_state_machine of curstate : signal is false;v`

- If you want to specify a particular encoding style for a state machine, use the syn_encoding attribute, as described in *Specifying FSMs with Attributes and Directives*, on page 343. When synthesized, these registers have the specified encoding style.
- Run synthesis.

The software automatically recognizes and extracts all the state machines, except the ones you marked. It optimizes the FSMs it extracted from the design, honoring the syn_encoding attribute. It writes out a log file that contains a description of each state machine extracted, and the set of reachable states for each FSM.

2. If you have many state machines you do not want optimized, do this:
   - Disable the compiler by disabling the Symbolic FSM Compiler box in one of these places: the main panel on the left side of the project window or the Options tab of the dialog box that comes up when you click the Add Implementation or Implementation Options buttons. This disables the compiler from optimizing any state machine in the design. You can now selectively turn on the FSM compiler for individual FSMs.
   - For state machines you want the FSM Compiler to optimize automatically, add the syn_state_machine directive to the individual state registers in the VHDL or Verilog code. Set the value to 1. When synthesized, the FSM Compiler extracts these registers with the default encoding styles according to the number of states.

Verilog	`reg [3:0] curstate /* synthesis syn_state_machine=1 */ ;`

VHDL	`signal curstate : state_type;`
	`attribute syn_state_machine : boolean;`
	`attribute syn_state_machine of curstate : signal is true;`

   - For state machines with specific encoding styles, set the encoding style with the syn_encoding attribute, as described in *Specifying FSMs*

with Attributes and Directives, on page 343. When synthesized, these registers have the specified encoding style.

– Run synthesis.

The software automatically recognizes and extracts only the state machines you marked. It automatically assigns encoding styles to the state machines with the syn_state_machine attribute, and honors the encoding styles set with the syn_encoding attribute. It writes out a log file that contains a description of each state machine extracted, and the set of reachable states for each state machine.

3. Check the state machine in the log file, the RTL and technology views, and the FSM viewer, which is not available to Synplify users. For information about the FSM viewer, see *Using the FSM Viewer*, on page 606.

Running the FSM Explorer

1. If you need to customize the extraction process, set attributes.

   – Use syn_state_machine=0 to specify state machines you do not want to extract and optimize.

Verilog
```
reg [3:0] curstate /* synthesis state_machine */ ;
```

VHDL
```
signal curstate : state_type;
attribute syn_state_machine : boolean;
attribute syn_state_machine of curstate : signal is true;
```

   – Use syn_encoding if you want to set a specific encoding style.

Verilog
```
reg [3:0] curstate /* synthesis syn_encoding="gray"*/ ;
```

VHDL
```
signal curstate : state_type;
attribute syn_encoding : string;
attribute syn_encoding of curstate : signal is "gray";
```

The FSM Compiler honors the syn_state_machine attribute when it extracts state machines, and the FSM Explorer honors the syn_encoding attribute when it sets encoding styles. See *Specifying FSMs with Attributes and Directives*, on page 343 for details.

Copyright © 2013 Synopsys, Inc.

2. Enable the FSM Explorer by checking the FSM Explorer box in one of these places:
   - The main panel on the left side of the project window
   - The Options tab of the dialog box that comes up when you click the Add Implementation or Implementation Options buttons.

   If you have not checked the FSM Compiler option, checking the FSM Explorer option automatically selects the FSM Compiler option.

3. Click Run to run synthesis.

   The FSM Explorer uses the state machines extracted by the FSM Compiler. If you have not run the FSM Compiler, the FSM Explorer invokes the compiler automatically to extract the state machines, instantiate state machine primitives, and optimize them. Then, the FSM Explorer runs through each encoding style for each state machine that does not have a syn_encoding attribute and picks the best style. If you have defined an encoding style with syn_encoding, it uses that style.

   The FSM Compiler writes a description of each state machine extracted and the set of reachable states for each state machine in the log file. The FSM Explorer adds the selected encoding styles. The FSM Explorer also generates a `<design>_fsm.sdc` file that contains the encodings and which is used for mapping.

4. Select View->View Log File and check the log file for the descriptions. The following extract shows the state machine and the reachable states as well as the encoding style, gray, set by FSM Explorer.

```
Extracted state machine for register cur_state
State machine has 7 reachable states with original encodings of:
   0000001
   0000010
   0000100
   0001000
   0010000
   0100000
   1000000
....
Adding property syn_encoding, value "gray", to instance
cur_state[6:0]
List of partitions to map:
   view:work.Control(verilog)
```

```
Encoding state machine work.Control(verilog)-
cur_state_h.cur_state[6:0]
original code -> new code
   0000001 -> 000
   0000010 -> 001
   0000100 -> 011
   0001000 -> 010
   0010000 -> 110
   0100000 -> 111
   1000000 -> 101
```

5. Check the state machine implementation in the RTL and Technology views and in the FSM viewer.

   For information about the FSM viewer, see *Using the FSM Viewer*, on page 606.

Copyright © 2013 Synopsys, Inc.

Inserting Probes

The probe insertion feature is only available with the Synplify Pro and Synplify Premier tools. Probes are extra wires that you insert into the design for debugging. When you insert a probe, the signal is represented as an output port at the top level. You can specify probes in the source code or by interactively attaching an attribute.

Specifying Probes in the Source Code

To specify probes in the source code, you must add the syn_probe attribute to the net. You can also add probes interactively, using the procedure described in *Adding Probe Attributes Interactively*, on page 463.

1. Open the source code file.

2. For Verilog source code, attach the syn_probe attribute as a comment on any internal signal declaration:

```
module alu(out, opcode, a, b, sel);
   output [7:0] out;
   input [2:0] opcode;
   input [7:0 a, b;
   input sel;
   reg [7:0] alu_tmp /* synthesis syn_probe=1 */;
   reg [7:0] out;
//Other code
```

   The value 1 indicates that probe insertion is turned on. For detailed information about Verilog attributes and examples of the files, see the *Reference Manual*.

   To define probes for part of a bus, specify where you want to attach the probes; for example, if you specify reg [1:0] in the previous code, the software only inserts two probes.

3. For VHDL source code, add the syn_probe attribute as follows:

```
architecture rtl of alu is
   signal alu_tmp : std_logic_vector(7 downto 0) ;
   attribute syn_probe : boolean;
   attribute syn_probe of alu_tmp : signal is true;
   --other code;
```

For detailed information about VHDL attributes and sample files, see the *Reference Manual*.

4. Run synthesis.

   The software looks for nets with the syn_probe attribute and creates probes and I/O pads for them.

5. Check the probes in the log file (`*.srr`) and the Technology view.

   This figure shows some probes and probe entries in the log file.

```
Adding property syn_probe, value 1, to net pc[0]
Adding property syn_probe, value 1, to net pc[1]
Adding property syn_probe, value 1, to net pc[2]
Adding property syn_probe, value 1, to net pc[3]
....
@N|Added probe pc_keep_probe_1[0] on pc_keep[0] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_1[0]
@N|Added probe pc_keep_probe_2[1] on pc_keep[1] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_2[1]
@N|Added probe pc_keep_probe_3[2] on pc_keep[2] in
eight_bit_uc
```

Adding Probe Attributes Interactively

The following procedure shows you how to insert probes by adding the syn_probe attribute through the SCOPE interface. Alternatively, you can add the attribute in the source code, as described in *Specifying Probes in the Source Code*, on page 462.

1. Open the SCOPE window and click Attributes.

2. Push down as necessary in an RTL view, and select the net for which you want to insert a probe point.

   Do not insert probes for output or bidirectional signals. If you do, you see warning messages in the log file.

3. Do the following to add the attribute:

   – Drag the net into a SCOPE cell.

Copyright © 2013 Synopsys, Inc.

- Add the prefix n: to the net name in the SCOPE window. If you are adding a probe to a lower-level module, the name is created by concatenating the names of the hierarchical instances.
- If you want to attach probes to part but not all of a bus, make the change in the Object column. For example, if you enter n:UC_ALU.longq[4:0] instead of n:UC_ALU.longq[8:0], the software only inserts probes where specified.
- Select syn_probe in the Attribute column, and type 1 in the Value column.
- Add the constraint file to the project list.

4. Rerun synthesis.
5. Open a Technology view and check the probe wires that have been inserted. You can use the Ports tab of the Find form to locate the probes.

   The software adds I/O pads for the probes. The following figure shows some of the pads in the Technology view and the log file entries.

```
Adding property syn_probe, value 1, to net pc[0]
Adding property syn_probe, value 1, to net pc[1]
Adding property syn_probe, value 1, to net pc[2]
Adding property syn_probe, value 1, to net pc[3]
....
@N|Added probe pc_keep_probe_1[0] on pc_keep[0] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_1[0]
@N|Added probe pc_keep_probe_2[1] on pc_keep[1] in
eight_bit_uc
@N|Also padding probe pc_keep_probe_2[1]
@N|Added probe pc_keep_probe_3[2] on pc_keep[2] in
eight_bit_uc
```

CHAPTER 9

Clock Conversion

ASICs are often implemented in FPGAs for prototyping. The inherent differences between ASICs and FPGAs can make this conversion difficult. One particular source of problems can be attributed to the complex clocking circuitry of FPGAs that often includes a large number of gated and internally generated clocks. The Synopsys FPGA synthesis tools provide two features, *Gated Clock Conversion* and *Generated Clock Conversion*, to address the complex clocking schemes. These features move the generated-clock and gated-clock logic from the clock pin of a sequential element to its enable pin.

Relocating these clocks allows the sequential elements to be tied directly to the skew-free source clock and also reduces the number of clock sources in the design which frees up routing resources and expedites placement and routing. Dedicated FPGA clock trees are routed to every sequential device on the die and are designed with low skew to avoid hold-time violations. Using these global clock trees allows the programmable routing resources of the FPGA to be used primarily for logic interconnect and simplifies static timing analysis because checks for hold-time violations based on minimum delays are unnecessary.

Copyright © 2013 Synopsys, Inc.

Working with Gated Clocks

The gated-clock feature is only available in the Synplify Pro and Synplify Premier tools. This section first describes the gated-clock solution, which is available for certain Altera, Lattice, and Xilinx technology families. The information is organized into the following sections:

- Obstacles to Conversion, on page 468
- Prerequisites for Gated Clock Conversion, on page 468
- Synthesizing a Gated-Clock Design, on page 475
- Accessing the Clock Conversion Report, on page 476
- Analyzing the Clock Conversion Report, on page 477
- Interpreting Gated Clock Error Messages, on page 479
- Disabling Individual Gated Clock Conversions, on page 482
- Using Gated Clocks for Black Boxes, on page 489
- OR Gates Driving Latches, on page 490
- Restrictions on Using Gated Clocks, on page 494

The clock conversion solution separates the gating from the clock inputs, and combines individual clock trees on the dedicated FPGA global clock trees. The software logically separates the gating from the clock and routes the gating to the clock enables on the sequential devices, using the programmable routing resources of the FPGA. The software separates a clock net going through an AND, NAND, OR, or NOR gate by doing one of the following:

- Inserting a multiplexer in front of the input pin of the synchronous element and connecting the clock net directly to the clock pin
- Moving the gating from the clock input pin to the dedicated enable pin, when this pin is available.

The ungated or base clock is routed to the clock inputs of the sequential devices using the global FPGA clock resources. Typically, many gated clocks are derived from the same base clock, so separating the gating from the clock allows a single global clock tree to be used for all gated clocks that reference that base clock.

Copyright © 2013 Synopsys, Inc.

It is not always necessary to convert all clocks from an ASIC design – fitting the design into the FPGA device and meeting timing are the ultimate goals. However, some situations require unconverted clock structures to be modified such as:

- Clock structures with a large number of loads resulting in congestion issues
- Clock structures resulting in a large clock skew between the starting and ending clocks on a path that does not meet timing

The following figure illustrates several cases of how gated clocks are converted.

Copyright © 2013 Synopsys, Inc.

Obstacles to Conversion

The following is a summary of issues to look for that can prevent a clock structure from being converted. More details about each condition are described later.

- Are the clocks defined properly?
- Is the Clock Conversion option enabled?
- Does the clock structure contain instantiated cells?
- Does the clock structure include MUX or XOR logic (MUX/XOR clock conversion is only supported in Synplify Premier for the HAPS flow)?
- Is something in the clock logic blocking optimization?
- Are there any unsupported constructs?

Prerequisites for Gated Clock Conversion

For a gated clock to be converted successfully, the design must meet these requirements:

Condition	Description
Combinational logic only	The gated clock logic must consist only of combinational logic. A derived clock that is the output of a register is not converted.
Single base clock	Identify only one input to the combinational logic for the gated clock as a base clock. To identify a net as a clock, specify a period or frequency constraint for either the gate or the clock in the constraint file. This example defines the clk input as the base clock: `create_clock -name {clk} -freq 10.000 -clockgroup default_clkgroup`
Supported primitives	The sequential primitive clocked by the gated clock must be a supported object. The tools support gated-clock conversion for most sequential primitives. Black-box modules driven by gated clocks can be converted if special synthesis directives are used to define the clock and clock enable inputs to the black box. See *Using Gated Clocks for Black Boxes*, on page 489.
Correct logic format	See *Correct Logic Format*, on page 469 for an example of the correct logic format.

Correct Logic Format

Specifically, the combinational logic for the gated clock must satisfy the following two conditions to have the correct format:

- For at least one set of gating input values, the value output for the gated clock must be constant and not change as the base clock changes.
- For at least one value of the base clock, changes in the gating input must not change the value output for the gated clock.

The correct logic format requirements are illustrated with the simple gates shown in the following figures. When the software synthesizes a design with the Gated Clocks option enabled, clock enables for the AND gate and OR gate are converted, but the exclusive-OR gate shown in the second figure is not converted. The following table explains.

AND gate `gclks[1]`	If either gate[1] or gate[2] is 0, then gclks[1] is 0, independent of the value of clk which satisfies the first condition. Also, if clk is 0, then gclks[1] is 0, independent of the values of gate[1] and gate[2] which satisfies the second condition. Because gclks[1] satisfies both conditions, it is successfully converted to the clock-enable format.
OR gate `gclks[2]`	If either gate[1] or gate[2] is 1, then gclks[2] is 1 independent of the value of clk which satisfies the first condition. Also, if clk is 1, then gclks[2] is 1 independent of the value of gate[1] or gate[2] which satisfies the second condition. Because gclks[2] satisfies both conditions, it is successfully converted to the clock-enable format.
Exclusive-OR gate `gclks[3]`	Irrespective of the value of gate[3], gclks[3] continues to toggle. The exclusive-OR function causes gclks[3] to fail both conditions which prevents gclks[3] from being converted.

Copyright © 2013 Synopsys, Inc.

Before Gated Clock Conversion

After Gated Clock Conversion

The clock enables for the AND and OR gates are converted, but the clock enable for the exclusive OR remains unchanged.

Defining Clocks Properly

The most common problems preventing clock conversion from occurring automatically are related to improperly defined clocks. When defining gated and generated clocks:

- Gated clocks should be defined at the nodes to be connected to the clock pins of the sequential cells.
- Use create_generated_clock constraints to define the relationship between generated clocks and their sources
- If a clock circuit contains both a clock generator and a gating element, a create_generated_clock constraint is needed. Without this constraint, the tool does not know the relationship between the gated signal and the source clock.

Defining Clocks Example

Using the above example, the first constraint applied is:

```
create_clock -name clk [get_ports clk] -period 10
```


Copyright © 2013 Synopsys, Inc.

In the above constraint, get_ports identifies clk as a port. With only the source clock defined, the tool is unable to determine which input of the AND gate is the clock and which input is the enable. Accordingly, no conversion is performed as shown in the following schematic diagram.

Continuing with the example, a second constraint is added:

```
create_clock -name clk [get_ports clk] -period 10

create_clock -name divclk [get_nets {n:divclk}] -period 20
```

In the second constraint, get_nets identifies divclk as a net. With independent clock constraints defining the source and generated clocks, the tool now knows which input of the AND gate is the clock, but still does not know the

relationship between the source and generated clocks. In this case, the clock gate and data register 'Z' are converted to an enable flip-flop (FDCE) which is connected to the generated clock as shown in the next schematic.

For the final example, the second clock constraint is replaced with a generated clock constraint.

```
create_clock -name clk [get_ports clk] -period 10

create_generated_clock -name divclk [get_nets {n:divclk}]
   -source [get_ports clk] -divide_by 2
```


Copyright © 2013 Synopsys, Inc.

Now the tool knows which input of the AND gate is the clock and the relationship between the source and generated clocks. For this case, the entire clock circuit is converted to an enable on the 'Z' register, and the 'Z' register clock pin is connected directly to the source clock as shown in the final schematic.

Internal Clocks

Applying internal clock constraints to output pins of inferred combinational logic (for example, an AND gate) is not supported. Apply clock constraints only to nets driven by logical output pins.

Only One Clock per Clock Pin Fan-in Cone

Every fan-in cone of the clock pin of a sequential device should have one defined clock. As long as one clock is defined, the clock can be traced forward through unate logic to the clock pins.

When no clock is defined:

- The clock source is derived by tracing the clock pin back to the first non-trivial (multiple input) gate
- Because the tool does not know which input of a multiple-input cell to trace, a clock is inferred at the output of that cell
- Since inferred clocks were not defined by the user explicitly or implicitly, no conversion occurs

When more than one clock is defined, the tool does not know which clock to convert, and no conversion occurs.

Synthesizing a Gated-Clock Design

The clock conversion feature described here is only available in the Synplify Pro and Synplify Premier tools for specific Altera, Lattice, and Xilinx technology families. To synthesize a gated-clock design:

1. Make sure that the gated clocks have the correct logic format and satisfy the prerequisites for conversion. See *Prerequisites for Gated Clock Conversion*, on page 468 for details.
2. If the gated clock drives a black box, specify the clock and the associated clock enable signal with using syn_force_seq_prim, syn_isclock, and syn_gatedclk_en directives. See *Using Gated Clocks for Black Boxes*, on page 489 for details.
3. Make sure that the clock net has a constraint specified in the constraint file for the current implementation. If you do not specify an explicit constraint on the clock net or if you set a global frequency constraint, enabling clock conversion as described in the next step will not have any effect.
4. Enable the Clock Conversion option.
   - Select Project->Implementation Options.

Copyright © 2013 Synopsys, Inc.

– On the GCC & Prototyping Tools tab (Synplify Premier) or GCC tab (Synplify Pro), click on the Clock Conversion check box.

5. Synthesize the design. For gated clocks, the option converts qualified flip-flops, counters, latches, synchronous memories, and instantiated technology primitives. The software logically separates the gating from the clock and routes the gating to the clock enables on the sequential devices, using the programmable routing resources of the FPGA. The ungated base clock is routed to the clock inputs of the sequential devices using the global clock resources. Because many gated clocks are normally derived from the same base clock, separating the gating from the clock allows a single global clock tree to be used for all gated clocks that reference the same base clock.

   See *Restrictions on Using Gated Clocks*, on page 494 for additional information.

6. Check the results in the START OF CLOCK OPTIMIZATION REPORT section of the log file. See *Analyzing the Clock Conversion Report*, on page 477 for an example of this report.

Accessing the Clock Conversion Report

The clock conversion report can be accessed from the log file or from the Project Status tab.

Log File Access

To access the clock conversion report directly in the log file:

1. Open the log file by clicking the View Log button.
2. In the browser, expand the Mapper Report entry by clicking the adjacent '+' sign and click on the Clock Conversion link to go directly to the clock

conversion report. Note that the browser is only available when the View log file in HTML option is enabled.

Project Status Tab Access

To access the clock conversion report directly from the Project Status tab, click the more link in the Optimizations Summary section at the bottom of the display.

Analyzing the Clock Conversion Report

When clock conversion is enabled, a typical report resembles the following:

Copyright © 2013 Synopsys, Inc.

```
#### START OF CLOCK OPTIMIZATION REPORT #####[

8 non-gated/non-generated clock tree(s) driving 434 clock pin(s) of sequential element(s)
2 gated/generated clock tree(s) driving 16 clock pin(s) of sequential element(s)
10 instances converted, 16 sequential instances remain driven by gated/generated clocks

========================================= Non-Gated/Non-Generated Clocks =========================================
Clock Tree ID     Driving Element     Drive Element Type     Fanout     Sample Instance
-----------------------------------------------------------------------------------------------------------------
ClockId0003       wb_clk_i            port                   215        camera_wishbone_if.dest_addr_cnt[31]
ClockId0004       bufg_inst2          BUFG                   163        control_synchronization.hsync_active_hi_sync
ClockId0005       ibufgds_inst        IBUFG
ClockId0006       bufg_inst3          BUFG                              ronization.cam_int_sync
ClockId0007       bufg_inst1          BUFG
ClockId0008       clk1                port                              reg_file_u1.ff_out[7]
ClockId0009       clk2                port                   2          clk_top_inst.clk_tree2_c1.ff_clk_u1.clk_out
ClockId0010       clk3                port                   1          clk_top_inst.reg_ff_u4.ff_out
=================================================================================================================
================================================================ Gated/Generated Clocks ================================
Clock Tree ID     Driving Element                                      Sample Instance
-----------------------------------------------------------------------------------------------------------------
ClockId0001       clk_top_inst.clk_tree2_c1.com                        clk_top_inst.reg_file_u2.ff_out[6]
ClockId0002       clk_top_inst.clk_tree3_c2.q                          clk_top_inst.reg_file_u3.ff_out[7]
=================================================================================================================

##### END OF CLOCK OPTIMIZATION REPORT ######]
```

3-line Summary

Non-gated/non-generated Clock Tree Table

Gated/generated Clock Tree Table

3-line Summary

The 3-line summary provides a quick snapshot of the design.

- The first line reports the number of *clean* clock trees (8 in the above report) and the number of driven clock pins (434).
- The next line reports the number of gated/generated clock trees (2 in the above report) and the number of driven clock pins (16).
- The last line reports the number of instances converted (10 in the above report) and the number of remaining instances driven by gated/generated clocks (16).

The following find command can be used to group the converted instances:

```
find -hier -inst * -filter @syn_gc_converted==1
```

Non-gated/non-generated Clocks Table

The non-gated/non-generated clocks section provides details of the *clean* clocks in the design and also includes any clock trees that never included gating logic. This section reports the following:

- Clock Tree ID – created for all instances in the clock tree that lead to the sample instance. Clicking on an ID opens a filtered technology view of the source clock.

- Driving Element – identifies the source of the driving clock.
- Drive Element Type – identifies the type of drive element such as a port or BUFG.
- Fanout – lists the clock fanout.
- Sample Instance – an instance within the clock tree.

The following find command can be used to identify all instances in the clock path:

- `find -hier -inst * -filter @syn_sample_clock_path==CKIDxxxx`

If a clock is defined within the cone-of-logic for the gated clock, it is reported in this table.

Gated/generated Clocks Table

The gated/generated clocks section lists a sample failure within each clock tree in the design with an explanation of why the conversion failed (see the following section *Interpreting Gated Clock Error Messages*, on page 479). The gated/generated clocks section reports the following:

- Clock Tree ID – created for all instances in the clock tree that lead to the failed sample instance. Clicking on an ID opens a filtered technology view of the source clock.
- Driving Element – identifies the source of the driving clock.
- Drive Element Type – identifies the type of drive element such as a LUT.
- Fanout – lists the clock fanout.
- Sample Instance – an instance within the clock tree.

In the table, failures with common gating-control signals are only reported once. The software traverses back from the clock pin of the sample instance until it reaches the driving element at the point of the failure.

Interpreting Gated Clock Error Messages

The following table describes the gated clock conversion error messages reported in the Gated/Generated Clocks section of the clock conversion report.

Copyright © 2013 Synopsys, Inc.

Error Message	Explanation
Asynchronous set/reset mismatch prevents generated clock conversion	Unshared asynchronous signals have been detected between a FF-derived clock circuit and its sequential load.
Can't determine input clock driver	
Clock conversion disabled	Gated clocks have been detected, but clock conversion is not enabled.
Clock propagation blocked by fixed hierarchy	Clock property has been blocked by a fixed hierarchy which prevents clock conversion from propagating upstream.
Clock propagation blocked by hard hierarchy	Clock property has been blocked by a hard hierarchy which prevents clock conversion from propagating upstream.
Clock propagation blocked by locked hierarchy	Clock property has been blocked by a locked hierarchy which prevents clock conversion from propagating upstream.
Clock propagation blocked by syn_keep	Clock property has been blocked by a syn_keep property which prevents clock conversion from propagating upstream.
Clock source is constant	
Combinational loop in clock network	
FF-derived clock conversion disabled	FF-derived clocks have been detected, but clock conversion is not enabled.
Gating structure creates improper gating logic	Unsupported gating structure has been detected. See _____ for supported gating types.
Illegal instance on clock path	
Inferred clock from port	
Input clock depends on output	
Latch gated by OR originally on clock tree	

Copyright © 2013 Synopsys, Inc.

Error Message	Explanation
Multiple clock inputs on sequential instance	
Multiple clocks on generating sequential element	
Multiple clocks on instance	Multiple clocks found feeding an instance of a clock tree.
Need declared clock or clock from port to derive clock from ff	
No clocks found on inputs	No clocks found feeding an instance of a clock tree.
No generated or derived clock directive on output of sequential instance	
No hierarchical driver	
Signal from port	
Unconverted clock gate	
Unable to determine clock driver on net	
Unable to determine clock input on sequential instance	
Unable to follow clock across hierarchy	
Unable to use latch as gated clock generator	

Checking the Log file

Many clock constraint problems can be identified by reviewing the Clock Summary table in the pre-mapping report section and the Performance Summary section of the log file shown below:

- Clock conversion is attempted on declared, generated, and derived clocks, and is never performed on inferred clocks or on the system clock.

Copyright © 2013 Synopsys, Inc.

- Declared, generated, and derived clocks can still have issues such as unsupported structures in their clock logic or incorrect constraints.
- Inferred clocks and the system clock are created when no clocks are defined in the fan-in logic of the clock pin of a sequential cell.

```
Performance Summary
*******************

Worst slack in design: 8.300

                                                          Clock      Clock
Starting Clock                                   Slack    Type       Group
-----------------------------------------------------------------------------------------
camera_open_source|CLKOUT0_derived_clock_CLKIN1  997.947  derived    Inferred_clkgroup_1
camera_open_source|CLKOUT1_derived_clock_CLKIN1  997.919  derived    Inferred_clkgroup_1
camera_open_source|camera_clock_pad_i            998.073  inferred   Inferred_clkgroup_1
camera_open_source|wb_clk_i                      996.467  inferred   Inferred_clkgroup_0
clk1                                             8.447    declared   default_clkgroup
clk2                                             8.300    declared   default_clkgroup
clk3                                             NA       declared   default_clkgroup
en3_0                                            NA       declared   default_clkgroup
System                                           NA       system     system_clkgroup
=========================================================================================

=========================================================================================
```

Following compilation, the software issues warnings about inferred clocks or the system clock, that include the clock name and one of the registers driven by that clock to help the user locate the issue in the HDL Analyst. The following are examples of these warnings:

- @W: MT529 : Found inferred clock *clkName* which controls *numClkPins* sequential elements including *instanceName*. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.
- @W: MT531 : Found signal identified as System clock which controls *numClkPins* sequential elements including *instanceName*. Using this clock, which has no specified timing constraint, can prevent conversion of gated or generated clocks and can adversely impact design performance.

Disabling Individual Gated Clock Conversions

By default, enabling gated-clock conversion applies globally to a design. For critical paths, individual clocked elements can be expressly excluded from the gated-clock conversion by adding a syn_keep directive directly to the input

clock net. The syn_keep attribute can be applied in the source code or in a constraint file. The following example shows a syn_keep attribute applied to the gclk net in a source file.

```
module and_gate (clk, en, a, z);
input clk, en, a;
output reg z;
wire gclk /* synthesis syn_keep = 1 */;
assign gclk = en & clk;

always @(posedge gclk) z <= a;
endmodule
```

The corresponding entry in a constraint file is:

```
define_attribute {n:gclk} {syn_keep} {1}
```

Instantiated Cells

The Synplify Premier synthesis tool supports conversion of gated- and generated-clock circuits, created from instantiated FPGA cells, to enable flops that connect directly to the clock (Synplify Pro does not support this feature). This feature is enabled via a switch in the GUI.

The equivalent set_option Tcl command for the project file is:

```
set_option -enhanced_optimization 1
```

If a MUX is described using instantiated LUT primitives, the tool is unable to recognize the structure as a MUX, and instead sees it as multiple clocks feeding a LUT (for example, a 3-input AND gate). As a result, no conversion occurs due to the multiplicity of declared clocks.

Copyright © 2013 Synopsys, Inc.

Integrated Clock Gating Cells

The Synopsys synthesis tools do not read in ASIC cell libraries. To support conversion of Integrated Clock Gating (IGC) cells, these cells must be modeled in RTL.

ASIC library ICG cell for posedge FFs

```
module icg (clk, en, gclk);
input clk, en;
output gclk;
wire lat;
assign lat = (!clk) ? en : lat;
assign gclk = lat & clk;
endmodule
```

RTL model

HDL Analyst view of compiled RTL

If there are ASIC-only pins on an ICG cell, such as a scan-enable input or an observation output, the pins must be removed to allow the clock conversion to occur.

The following example shows an ICG cell with ASIC-only pins and the corresponding Verilog source code.

```
module icg (clk, en, scan_en, obs, gclk);
input clk, en, scan_en;
output obs, gclk;
wire t0, t1;
assign t0 = (!clk) ? en : t0;
assign obs = t0;
assign t1 = t0 | scan_en;
assign gclk = t1 & clk;
endmodule
```

The presence of the scan control and observation point signals forces the tool to preserve intermediate signals in the clock-gating circuit that prevents it from being optimized. In this case, the data register is connected directly to the source clock, but the latch and OR gate for the ICG model remain as part of the enable logic as shown in the following schematic.

Copyright © 2013 Synopsys, Inc.

The following code simplifies the RTL model of the complex IGC cell and allows the desired gated-clock conversion to occur.

```
module icg (clk, en, scan_en, obs, gclk);
input clk, en, scan_en;
output obs, gclk;
wire t0, t1;
assign t0 = (!clk) ? en : t0;
assign obs = 1’b0 /* t0 */;
assign t1 = t0 | 1'b0 /* scan_en */;
assign gclk = t1 & clk;
endmodule
```

In the modified code, the scan_en input, which is generally not used in an FPGA implementation, is replaced with a constant ‘0’. The obs observation point output, which normally is not required in an FPGA implementation, is also tied to a constant ‘0’. These changes remove the intermediate points as shown in the following schematic. Note that if no other changes are made to the design, this model still “fits” because the ports are unchanged. Also, the ‘0’ tied to the obs output propagates to simplify downstream logic.

Copyright © 2013 Synopsys, Inc.

Incorrect Phase

When there is an error in the RTL model of an ICG cell such that an incorrect phase of the clock is connected to the latch:

- Only the AND gate is converted leaving the latch driving the enable pin of the data flip-flops.

- A warning is generated stating "Active phase of latch *latchName* prevents its use as a stability latch for clock gate *clockGateName*."

Using Gated Clocks for Black Boxes

To convert gated clocks that drive black boxes, you must identify the clock and clock enable signal inputs to the black box using the syn_force_seq_prim, syn_isclock, and syn_gatedclk_clock_en directives. Refer to the *Reference Manual* for information about these directives. You assume responsibility for the functionality of the black-box content.

The following are Verilog and VHDL examples of a black box with the required directives specified.

Verilog

```
module bbe (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim="clk" */
;
input clk
/* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL

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

entity bbe is
   port (clk : in std_logic;
         en : in std_logic;
         data_in : in std_logic;
         data_out : out std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of clk : signal is "en";
end bbe;

architecture behave of bbe is
attribute syn_black_box : boolean;
attribute syn_force_seq_prim : string;
attribute syn_black_box of behave : architecture is true;
attribute syn_force_seq_prim of behave : architecture is "clk";
begin
end behave;
```

OR Gates Driving Latches

To prevent problems in timing analysis when a latch is being driven by the OR of a clock and an enable, gated-clock conversion is implemented using the logic shown in the following diagram.

As illustrated in the above diagram, the enable (EN) and clock (CLK) feed separate latches, and XOR gates are used to condition the inputs and outputs. The conversion masks the combinational path from the timing engine and duplicates the behavior of the original circuit. Conversion only occurs if the original latch is not driving a clock network.

MUX or XOR Logic in Clock Structure

If you are running Synplify Premier and targeting a compatible Xilinx device, the Convert MUX/XOR Gates Clock Tree check box is enabled on the GCC & Prototyping Tools tab to allow selection of MUX/XOR clock conversion.

The equivalent set_option Tcl command for the project file is:

```
set_option -conv_mux_xor_gated_clocks 1
```

For other FPGA synthesis flows and non-compatible devices, clocks incorporating MUX or XOR logic in their structure are not converted. Note that unwanted MUX and XOR logic can be generated for the following RTL constructs:

- (A&B) | (!A&C) counts as a MUX, not simple AND/OR logic
- (A&!B) | (!A&B) counts as an XOR, not simple AND/OR logic

The conversion of MUX and XOR clock structures is area expensive, and should be used sparingly.

If the clock structure includes MUX or XOR logic, the following conditions must be met for automatic conversion to occur:

- All data inputs to a MUX must be clocks
- Exactly one input to a XOR must be a clock

Common MUX/XOR Conversion Issues

The following are common conversion issues with MUX/XOR logic:

- XOR conversion with more than one input defined as a clock is not attempted and no change is made to the clock circuit.
- MUX conversion when not all of the data inputs are defined as clocks is not attempted and no change is made to the clock circuit. The most

Copyright © 2013 Synopsys, Inc.

common case occurs with scan logic when the definition of the test clock is removed, but the MUX is left in place. The best solution for this case is to tie the select input of the MUX (test_enable in the following schematic) to a constant so that the functional clock is always selected. Constant propagation subsequently optimizes the MUX away.

In standard flow, a MUX in the clock logic cannot be converted

With test_en tied to 0, constant propagation optimizes the MUX to a wire

Obstructions to Optimization

The following optimization obstructions in the clock structure prevent conversion:

"Keep_buffers" in the Clock Structure

Keep buffers are components that are added to the RTL view netlist to preserve specific nets. If a net in the clock circuitry is preserved by a keep buffer, it cannot be optimized away and can prevent conversion of the associated clock circuit. Keep buffers are the result of syn_keep attributes and

constraints placed on nets such as false path, multi-cycle path, and other clock constraints. The following Tcl find command can be used to locate the keep buffers:

```
find -hier -view keepbuf
```

"Hard" Hierarchy Defined on a Clock Logic Block

A "hard" hierarchy attribute on a block containing a clock structure prevents conversion by forcing the interface of this block to remain unchanged. More specifically, if a hard hierarchy is between the clock logic and the data registers, the conversion cannot occur because the new signals for the enables cannot be added to the block interface. The following Tcl find command can be used to locate the hard hierarchies:

```
find -view * -filter (@syn_hier==hard)
```

Loads on Intermediate Nodes in the Clock Logic

Intermediate nodes in the clock logic should not have extra loads. For conversion to occur when extra loads are present, the RTL must be modified to separate the loads that drive clock pins of data registers from the other loads.

"Disable Sequential Optimizations" Project Option Prevents Latches from Being Removed

For conversion to occur, the Disable Sequential Optimizations option cannot be enabled.

Unsupported Constructs

The following are examples of constructs that prevent conversion from occurring.

- If the clock circuitry contains logic such that asynchronous set/reset signals can produce an active edge on the gated-clock line (without any change in the source clock), the circuit is not converted. Any active edges on the gated-clock output must be controlled by the source clock.
- By default, conversion only occurs if the registers in the generated-clock logic have the same asynchronous set/reset signals as the data registers that they control. If this condition is not met, conversion can be forced

Copyright © 2013 Synopsys, Inc.

by selecting the following option in the GCC & Prototyping Tools tab (available in Synplify Premier only).

The equivalent set_option Tcl command for the project file is:

```
set_option -force_async_genclk_conv 1
```

- If the wrong phase of the clock is connected to a latch in a gated-clock circuit, the following warning is reported:

```
Active phase of latch latchName prevents its use as a
   stability latch for clock gate clockGateName
```

Other Potential Workarounds to Solve Clock-Conversion Issues

If the previous sections do not help resolve your issue, there a few more things you can try:

- Use a syn_direct_enable attribute on one of the gating signals to force the conversion.
- Instantiate BUFGs (Xilinx) or CLKBUFGMUXs (Altera).
- Use netlist edit commands to modify the netlist.
- Recode RTL to fit one of the cases allowed for automatic conversion.

Restrictions on Using Gated Clocks

Currently, the Clock Conversion option has the following restrictions for gated clocks:

- If the syn_keep attribute is assigned to a net, the Clock Conversion option does not preserve this net during optimization.
- The Clock Conversion option cannot be implemented for inferred counters in Altera technologies.

Copyright © 2013 Synopsys, Inc.

- The Clock Conversion option cannot be implemented by the Synplify Premier tool if the gates associated with the gated clock are assigned to different design plan regions (see the following figure). The Clock Conversion option *can* be applied if all gates associated with the gated clock are assigned to the same design plan region. Also, the flip-flops can be assigned to any Synplify Premier region.

- A global buffer, when instantiated on a gated clock, blocks gated-clock conversion from being performed (the buffer is viewed as the clock source by the downstream circuitry). In the circuit below, the clk input to test_inst remains gated.

Similarly, defining a clock constraint at the output of a gated clock prevents conversion from continuing upstream.

Copyright © 2013 Synopsys, Inc.

Issues with Gated/Generated Clock Reporting

Several gated/generated clock reporting issues have been identified:

- When both inputs for XOR gates are defined as clocks, the software incorrectly reports the explanation as “No clocks found on input”. The correct explanation is “Improper clock gating structure.” For HAPs compatible devices, the correct explanation is “Multiple clocks found on inputs.”
- When both inputs for MUXs are defined as clocks, the software incorrectly reports “Multiple clocks on instances”. The correct explanation is “Improper clock gating structure”.
- Gated clocks might not be converted for logic from a hierarchy that uses syn_hier=hard/fixed. In the following example, the comb1 hierarchy has this attribute. Although the combinational logic within the comb2 hierarchy is outside the scope of syn_hier=hard, the enable logic of register, out1 does not get converted.

- For cascaded ICGs driving posedge flip-flops that use either compile points or the syn_hier attribute, the tool might produce a circuit with unused logic that does not get pruned away. See the shaded logic in the following figure. Depending on how the compile point or syn_hier boundaries are defined, the tool can report unused latches as unconverted, with the explanation “Unable to use latch as gated clock generator.”

RTL:

Post Mapping:

- The explanation "Unconverted clock gate" message indicates a gated/generated clock case that is not documented. Report this undocumented case on SolvNet.

Clock Conversion for Multiple Input Clocks to OR Gate Latch

If multiple clocks are defined as inputs to an OR gate driving a latch, then clock conversion does not occur and the following message is generated: Multiple clocks on instance. If this message occurs, make sure that only one clock is input to the OR gate driving the latch.

Copyright © 2013 Synopsys, Inc.

No Gated Clock Conversion for Altera Macros

Gated-clock conversion does not occur for RAM or DSP macros when Altera Models is off or when QUARTUS_ROOTDIR is not defined. The only exception is that gated-clock conversion does occur for instantiated altsyncram megafunctions.

No Gated Clock Conversion with altsyncram Megafunctions

Gated-clock conversion does not occur for instantiated altsyncram megafunctions when Altera Models is turned on (default). To use inferred RTL instead of the altsyncram megafunction, turn off Altera Models. Note, the Altera Models option will be turned off globally for the design.

Copyright © 2013 Synopsys, Inc.

Optimizing Generated Clocks

The Synplify Pro and Synplify Premier tools include an option for generated clocks that is available for specific Altera, Xilinx, and Lattice families. When the Clock Conversion option is enabled, the generated-clock logic is replaced during synthesis with logic that uses the initial clock with an enable. With generated-clock optimization, the original circuit functionality is preserved while performance is improved by reducing clock skew. For more information, see the following topics:

- Enabling Generated-Clock Optimization, on page 499
- Conditions for Generated-Clock Optimization, on page 499
- Generated-Clock Optimization Examples, on page 500

Enabling Generated-Clock Optimization

Generated-clock optimization is enabled by checking Clock Conversion on the GCC & Prototyping Tools tab (Synplify Premier) or GCC tab (Synplify Pro) of the Implementation Options dialog box.

The Force Generated Clock Conversion with Asynchronous Signals check box is only available with Synplify Premier and, when checked, enables the conversion of generated clock-driven datapath latches to a flip-flop and multiplexer when the generated-clock logic/datapath latch is set or reset by asynchronous signals (by default, generated-clock-driven datapath latches with asynchronous control signals are not converted).

Conditions for Generated-Clock Optimization

To perform generated-clock optimization, the following conditions must be met:

Copyright © 2013 Synopsys, Inc.

1. The combinational logic must be driven by flip-flops.
2. The input flip-flops cannot have an active set or reset (an active-low reset must be tied high, or an active-high set must be tied low). Similar rules apply to all the input flip-flops in the cone.
3. All input flip-flops must be driven by the same edge of the same clock.

With generated-clock optimization, you do not have to specify a primary clock.

Generated-Clock Optimization Examples

Example 1

The following code segment illustrates generated-clock optimization:

```
module gen_clk(clk1,a,b,c,q);
input clk1, a, b, c;
output q;
reg ao,bo,q;
wire en;

always @(posedge clk1)
   begin
      ao <= a;
      bo <= b;
   end
assign en = ao & bo;

always @(posedge en)
   begin
      q <= c;
   end
endmodule
```

With generated-clock optimization disabled (Clock Conversion unchecked), the circuit in the following figure shows a flip-flop (q) driven by a generated clock that originates from the combinational logic driven by flip-flops ao and bo which, in turn, are driven by the initial clock (clk1).

When generated-clock optimization is enabled (Clock Conversion checked), flip-flop q is replaced with an enable flip-flop. This flip-flop is clocked by the initial clock (clk1) and is enabled by combinational logic based on the a and b inputs as shown in the following figure.

Example 2

A design can have a register that is clocked by a generated clock, then followed by a gated clock. If the generated-clock constraint also includes the enable, the Clock Conversion option may optimize away the register and replace it with a constant value. When this occurs, the software converts the generated and gated-clock logic into a clock and an enable; the enable may never go active.

Copyright © 2013 Synopsys, Inc.

Example 3

Similar to example 2, a design can have a latch on the datapath that is clocked by a generated clock (a flip-flop derived clock).

In this case, the circuit is converted to a flip-flop and multiplexer as shown in the following figure.

The above conversion does not occur when the latch and generated-clock logic are either set or reset by asynchronous signals unless either:

- The Force Generated Clock Conversion with Asynchronous Signals check box on the GCC & Prototyping Tools tab of the Implementation Option dialog box is checked.
- The force_async_genclk_conv option of the set_option command is set to 1.

Copyright © 2013 Synopsys, Inc.

CHAPTER 10

Synthesizing and Analyzing the Results

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

- Synthesizing Your Design, on page 506
- Checking Log Results, on page 514
- Handling Messages, on page 521
- Continue on Error, on page 531
- Validating Results for Physical Synthesis, on page 536
- Congestion Analysis After Logic Synthesis, on page 538

Copyright © 2013 Synopsys, Inc.

Synthesizing Your Design

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

- Running Logic Synthesis, on page 506
- Running Physical Synthesis, on page 506
- Using Up-to-date Checking for Job Management

Running Logic Synthesis

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

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

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

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

3. To run logic synthesis as the initial phase of physical synthesis, see Running Physical Synthesis, on page 506.

   You can now run physical synthesis as described in *Running Physical Synthesis*, on page 506.

Running Physical Synthesis

When you run physical synthesis, the tool not only compiles the design and maps it to the technology target you selected, but also uses placement information to concurrently optimize and synthesize your design. Regardless of the flow you are using, run physical synthesis in two phases. First, run logic synthesis and fix any issues that come up. Then run physical synthesis.

Copyright © 2013 Synopsys, Inc.

1. Run logic synthesis as the initial phase of physical synthesis, by doing the following:
   - Set the options and attributes you want for physical synthesis, making sure to set up P&R to run automatically after synthesis.
   - Disable the Physical Plus switch either in the Project view or from the Implementation Options dialog box (Implementation Options->Options).

   - Click Run to run logic synthesis.

   The Synplify Premier tool goes through *compiling* and *mapping* phases. When logical synthesis completes, Done! (or Warnings!) displays in the Project view. Output results files are shown in the right pane of the Project view.

Note: The physical synthesis flow runs the Synplify global placer by default. To run the Xilinx global placer, use the syn_use_xilinx_placement attribute.

2. Make adjustments to your design as needed.
   - Check the output files and analyze the results.
   - Fix any errors.

   See *Validating Results for Physical Synthesis*, on page 536 for details.

3. Set options for the physical synthesis run.
   - Set any other physical constraints.
   - Enable the Physical Plus switch either in the Project view or from the Implementation Options dialog box (Implementation Options->Options).

Copyright © 2013 Synopsys, Inc.

– If you want a different directory for your physical synthesis results, click on the Implementation Options->Implementation Results tab and specify a new name for the implementation.

– Make sure the place-and-route implementation is enabled (Implementation Options->Place and Route tab).

– If you are using a Design Planner flow, click on the Design Planning tab and enable the desired design plan file (`sfp`) if needed.

You do not need a design plan file to run graph-based physical synthesis. However, if you are using a graph-based flow and want to use a design plan file, use the procedure described in *Creating and Using a Design Plan File for Physical Synthesis*, on page 618.

For older Altera technologies, you must create a design plan (`sfp`) to run physical synthesis. See Chapter 12, *Floorplanning with Design Planner* for more information.

   – Click OK in the Implementation Options dialog box.

4. Run physical synthesis by clicking Run.

   The tool performs optimizations using placement-aware synthesis. Synthesis and placement are integrated by performing concurrent placement and optimization based on timing constraints and device technology.

Using Up-to-date Checking for Job Management

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

Up-to-date checking is run for all synthesis design flows. However, for the Hierarchical Project Management flows, up-to-date checking is an essential feature. For example, if a project contains four sub-projects and only one project is modified, then the other three projects do not need to be rerun. This saves in overall runtime.

Up-to-date checking includes the following:

- The GUI launches mapper modules (pre-mapping and technology mapping) and saves the intermediate netlists and log files in the synwork and synlog folders, respectively.
- After each individual module run completes, the GUI optionally copies the contents of these intermediate log files from the synlog folder and adds them to the Project log file (rev_1/*projectName*.srr). To set this option, see *Copy Individual Job Logs to the SRR Log File*, on page 511.
- If you re-synthesize the design and there are no changes to the inputs (HDL, constraints, and Project options):
  – The GUI does not rerun pre-mapping and technology mapping and no new netlist files are created.
  – In the HTML log file, the GUI adds a link that points to the existing pre-mapping and mapping log files from the previous run. Double-click on this link (`@L:` indicates the link) to open the new text file window.

Copyright © 2013 Synopsys, Inc.

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

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

```
##########################################################]

Synopsys Pre-mapping Report
@N: :  | premapping output is up to date. No run necessary.
To force a re-synthesis, select [Resynthesize All] in menu [Run].

Log file from previous run:
@L:top_premapping.srr
##########################################################]
```

```
ersion map610dev, Build
ll Rights Reserved

d\gated_clocks\test.sdc

le <C:\builds\syn201103_
table from file <C:\buil
@W:"c:\syn_hier\fixed\gated_clocks\myreg.v":18:14:18:22|Net U_reg
@W:"c:\syn_hier\fixed\gated_clocks\top.v":17:14:17:24|Net U_gt_re

Finished Pre Mapping Phase. (Time elapsed 0h:00m:00s; Memory used

@N: BN225 |Writing default property annotation file C:\syn_hier\f
Pre Mapping successful!
Process took 0h:00m:01s realtime, 0h:00m:01s cputime
# Mon Jan 24 18:39:06 2011

##########################################################]
```

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

Job Status for Synthesis Run

Job Status for Re-synthesis Run

See also:

- Copy Individual Job Logs to the SRR Log File
- Limitations and Risks

Copy Individual Job Logs to the SRR Log File

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

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

Copyright © 2013 Synopsys, Inc.

Limitations and Risks

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

- Compiler up-to-date checks are done internally by the compiler and with no changes to the compiler reporting structure.
- GUI up-to-date checks use timestamp information of its input files to decide when mapping is re-run. Be aware that:
  - The GUI uses netlist files (`srs` and `srd`) from the synwork folder for timestamp checks. If you delete an `srs` file from the implementation folder, this does not trigger compiler or mapper re-runs. You must delete netlist files from the synwork folder instead.
  - The copy command behaves differently on Windows and Linux. On Windows, the timestamp does not change if you copy a file from one

directory to another. But on Linux (and MKS shell), the timestamp information gets changed.

  - If your Project file includes `ngc` or `ngo` cores, up-to-date checking does not work. This condition always triggers a rerun, even when nothing is changed in your design. The Xilinx ngc2edif utility converts the `ngc/ngo` to `edif` and the timestamp is always the latest.

- When running a design, the up-to-date checking feature automatically determines if the design needs to be re-synthesized. However, when you modify constraints in a Tcl file sourced within the constraints file, the software is not aware of these changes and does not force the design to be re-synthesized.

Copyright © 2013 Synopsys, Inc.

Checking Log Results

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

- Viewing the Log File, on page 514
- Analyzing Results Using the Log File Reports, on page 517
- Using the Watch Window, on page 518

Viewing the Log File

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

For Synplify Pro or Synplify Premier users who only want to check a few critical performance criteria, it is easier to use the Watch Window (see *Using the Watch Window*, on page 518) instead of the log file. For details, read through the log file.

1. To view the log file, do one of the following:
   - To view the log file in the default HTML format, select View->Log File or click the View Log button in the Project window. You see the log file in HTML format. Alternatively you can double-click the *designName*_srr.htm file in the Implementation Results view to open the HTML log file.
   - To see a text version of the log file, double-click the *designName*.srr file in the Implementation Results view. A Text Editor window opens with the log file.

     Alternatively, you can set the default to show the text file version instead of the HTML version. Select Options->Project View Options, and toggle off the View log file in HTML option.

   The log file lists the compiled files, details of the synthesis run, color-coded errors, warnings and notes, and a number of reports. For information about the reports, see *Analyzing Results Using the Log File Reports*, on page 517.

2. To navigate in the log file, use the following techniques:
   - Use the scroll bars.
   - Use the Find command as described in the next step.
   - In the HTML file, click the appropriate header to jump to that point in the log file. For example, you can jump to the Starting Points with Worst Slack section.
   - In the HTML file, you can use the search field to find a header in the table of contents. Enter all or part of the header name in the search field, then click Find. The log file will display the resulting section.

Copyright © 2013 Synopsys, Inc.

3. To find information in the log file, you can also select Edit->Find or press Ctrl-f. Fill out the criteria in the form and click OK.

   For general information about working in an Editing window, including adding bookmarks, see *Editing HDL Source Files with the Built-in Text Editor*, on page 111.

   The areas of the log file that are most important are the warning messages and the timing report. The log file includes a timing report that lists the most critical paths. The Synplify Pro and Synplify Premier products also let you generate a report for a path between any two designated points, see *Generating Custom Timing Reports with STA*, on page 729. The following table lists places in the log file you can use when searching for information.

To find...	Search for...
Notes	@N or look for blue text
Warnings and errors	@W and @E, or look for purple and red text respectively
Performance summary	Performance Summary
The beginning of the timing report	START TIMING REPORT
Detailed information about slack times, constraints, arrival times, etc.	Interface Information
Resource usage	Resource Usage Report
Gated clock conversions	Gated clock report

4. Resolve any errors and check all warnings.

   You must fix errors, because you cannot synthesize a design with errors. Check the warnings and make sure you understand them. See *Checking Results in the Message Viewer*, on page 521 for information. Notes are informational and usually can be ignored. For details about crossprobing and fixing errors, see *Handling Warnings*, on page 530, *Editing HDL Source Files with the Built-in Text Editor*, on page 111, and *Crossprobing from the Text Editor Window*, on page 584.

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

5. If you are trying to find and resolve warnings, you can bookmark them as shown in this procedure:
   - Select Edit->Find or press Ctrl-f.
   - Type @W as the criteria on the Find form and click Mark All. The software inserts bookmarks at every line with a warning. You can now page through the file from bookmark to bookmark using the commands in the Edit menu or the icons in the Edit toolbar. For more information on using bookmarks, see *Editing HDL Source Files with the Built-in Text Editor*, on page 111.

6. To crossprobe from the log file to the source code, click on the file name in the HTML log file or double-click on the warning text (not the ID code) in the ASCII text log file.

Analyzing Results Using the Log File Reports

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

1. To analyze timing results, do the following:
   - View the Timing Report by going to the Performance Summary section of the log file.
   - Check the slack times. See *Handling Negative Slack*, on page 728 for details.
   - Check the detailed information for the critical paths, including the setup requirements at the end of the detailed critical path

Copyright © 2013 Synopsys, Inc.

description. You can crossprobe and view the information graphically and determine how to improve the timing.

– In the HTML log file, click the link to open up the HDL Analyst view for the path with the worst slack.

To generate Synplify Premier or Synplify Pro timing information about a path between any two designated points, see *Generating Custom Timing Reports with STA*, on page 729.

2. To check buffers,
   – Check the report by going to the Net Buffering Report section of the log file.
   – Check the number of buffers or registers added or replicated and determine whether this fits into your design optimization strategy.
3. To check logic resources,
   – Go to the Resource Usage Report section at the end of the log file.
   – Check the number and types of components used to determine if you have used too much of your resources.

Using the Watch Window

The Synplify Pro and Synplify Premier Watch window provides a more convenient viewing mechanism than the log file for quickly checking key performance criteria or comparing results from different runs. Its limitation is that it only displays certain criteria. If you need details, use the log file, as described in *Viewing the Log File*, on page 514.

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

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

2. If you need a larger window, either resize the window or move the Watch Window as described below.
   – Hold down Ctrl or Shift, click on the window, and move it to a position you want. This makes the Watch window an independent window, separate from the Project view.

– To move the window to another position within the Project view, right-click in the window border and select Float in Main Window. Then move the window to the position you want, as described above.

See *Watch Window*, on page 81 in the *Reference Manual* for information about the popup menu commands.

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

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

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

The software shows the values for the selected implementations side by side. For more information about multiple implementations, see *Tips for Optimization*, on page 422.

Copyright © 2013 Synopsys, Inc.

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

Copyright © 2013 Synopsys, Inc.

Handling Messages

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

- Checking Results in the Message Viewer, on page 521
- Filtering Messages in the Message Viewer, on page 523
- Filtering Messages from the Command Line, on page 525
- Automating Message Filtering with a Tcl Script, on page 526
- Log File Message Controls, on page 528
- Handling Warnings, on page 530

Checking Results in the Message Viewer

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

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

2. Click the Messages tab to open the message viewer.

   The window lists the errors, warnings, and notes in a spreadsheet format. See *Message Viewer*, on page 85 in the *Reference Manual* for a full description of the window.

Copyright © 2013 Synopsys, Inc.

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

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

     There are two types of message groups:

     - The same warning or note ID appears in multiple source files indicated by a dash in the source files column.

     - Multiple warnings or notes in the same line of source code indicated by a bracketed number.
   - Sort the messages. To sort by a column header, click that column heading. For example, click Type to sort the messages by type. For example, you can use this to organize the messages and work through the warnings before you look at the notes.
   - To find a particular message, type text in the Find field. The tool finds the next occurrence. You can also click the F3 key to search forward, and the Shift-F3 key combination to search backwards.

4. To filter the messages, use the procedure described in *Filtering Messages in the Message Viewer*, on page 523. Crossprobe errors from the message window:
   - If you need more information about how to handle a particular message, click the message ID in the ID column. This opens the documentation for that message.
   - To open the corresponding source code file, click the link in the Source Location column. Correct any errors and rerun synthesis. For warnings, see *Handling Warnings*, on page 530.
   - To view the message in the context of the log file, click the link in the Log Location column.

Filtering Messages in the Message Viewer

The Message viewer lists all the notes, warnings, and errors. It is not available with the Synplify tool. The following procedure shows you how to filter out the unwanted messages from the display, instead of just sorting it as described in *Checking Results in the Message Viewer*, on page 521. For the command line equivalent of this procedure, see *Filtering Messages from the Command Line*, on page 525.

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

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

3. Set your display preferences.

Copyright © 2013 Synopsys, Inc.

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

4. Set the filtering criteria.
   - Set the columns to reflect the criteria you want to filter. You can either select from the pull-down menus or type your criteria. If you have multiple synthesis runs, the pull-down menu might contain selections that are not relevant to your design.

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

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

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

5. To save your message filters and reuse them, do the following:
   - Save the project. The synthesis tool generates a Tcl file called *projectName*.pfl (Project Filter Log) in the same location as the main project file. The following is an example of the information in this file:

```
log_filter -hide_matches
log_filter -field type==Warning
   -field message==*Una*
   -field source_loc==sendpacket.v
   -field log_loc==usbHostSlave.srr
   -field report=="Compiler Report"
log_filter -field type==Note
log_filter -field id==BN132
log_filter -field id==CL169
log_filter -field message=="Input *"
log_filter -field report=="Compiler Report"
```

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

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

Filtering Messages from the Command Line

The following procedure shows you how to use Tcl commands to filter out unwanted messages. If you want to use the GUI, see *Filtering Messages in the Message Viewer*, on page 523. Message filtering is not available with the Synplify tool.

1. Type your filter expressions in the Tcl window using the log_filter command. For details of the syntax, see *log_filter Tcl Command*, on page 1721 in the *Reference Manual*.

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

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

2. To save and reuse the filter commands, do the following:
   – Type the log_filter commands in a Tcl file.
   – Source the file when you want to reuse the filters you set up.

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

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


Copyright © 2013 Synopsys, Inc.

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

```
@N MF138 Rom slaveControlSel_1 mapped in logic. Mapper Report
   wishbonebi.v (156) usbHostSlave.srr (819) 05:22:06 Mon Oct 18
@N(2) MO106 Found ROM, 'slaveControlSel_1', 15 words by 1 bits
   Mapper Report wishbonebi.v (156) usbHostSlave.srr (820)
   05:22:06 Mon Oct 18
@N MO106 Found ROM, 'slaveControlSel_1', 15 words by 1 bits Mapper
   Report wishbonebi.v (156) usbHostSlave.srr (820) 05:22:06 Mon
   Oct 18
@N MF138 Rom hostControlSel_1 mapped in logic. Mapper Report
   wishbonebi.v (156) usbHostSlave.srr (821) 05:22:06 Mon Oct 18
@N MO106 Found ROM, 'hostControlSel_1', 15 words by 1 bits Mapper
   Report wishbonebi.v (156) usbHostSlave.srr (822) 05:22:06 Mon
   Oct 18
@N Synthesizing module writeUSBWireData Compiler Report
   writeusbwiredata.v (59) usbHostSlave.srr (704) 05:22:06 Mon Oct 18
```

Automating Message Filtering with a Tcl Script

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

1. Create a message filter file like the following. (See *Filtering Messages in the Message Viewer*, on page 523 or *Filtering Messages from the Command Line*, on page 525 for details about creating this file.)

   ```
   log_filter -clear
   log_filter -hide_matches
   log_filter -field report=="VIRTEX2P MAPPER"
   log_filter -field type==NOTE
   log_filter -field message=="Input *"
   log_filter -field message=="Pruning *"
   puts "DONE!"
   ```

2. Copy the synhooks.tcl file and set the environment variable as described in *Automating Flows with synhooks.tcl*, on page 1146.

3. Edit the synhooks.tcl file so that it reads like the following example. For syntax details, see *synhooks File Syntax*, on page 1719 in the *Reference Manual*.

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

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

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

```
proc syn_on_end_run {runName run_dir implName} {
set warningFileName "messageFilename"

if {$runName == "synthesis"} {
   puts "Mapper Done!"
   log_report -print $warningFileName
set f [open [lindex $warningFileName] r]
set msg ""
while {[gets $f warningLine]>=0} {
   puts $warningLine
   append msg $warningLine\n
   }
close $f
```

– Continue by specifying that the messages be sent in email. You can obtain the smtp email packages off the web.

```
source "d:/tcl/smtp_setup.tcl"
proc send_simple_message {recipient email_server subject body}{
   set token [mime::initialize -canonical text/plain -string
      $body]
   mime::setheader $token Subject $subject
   smtp::sendmessage $token -recipients $recipient -servers
      $email_server
   mime::finalize $token
}
puts "Sending email..."

send_simple_message {address1,address2}
   yourEmailServer subjectText> emailText
   }
}
```

When the script runs, an email with all the warnings from the synthesis run is automatically sent to the specified email addresses.

Copyright © 2013 Synopsys, Inc.

Log File Message Controls

The log file message control feature allows messages in the current session to be elevated in severity (for example, promoted to an error from a warning), lowered in severity (for example, demoting a warning to a note), or suppressed from the log file after the next run through the Log File Filter dialog box. This dialog box is displayed by opening the log file in HTML mode and selecting Log File Message Filter from the popup menu with the right mouse button.

Log File Filter Dialog Box

The Log File Filter dialog box is the primary control for changing a message priority or suppressing a message. When you initially open the dialog box, all of the messages from the log (`srr`) file for the active implementation are displayed in the upper section and the lower section is empty. To use the dialog box:

1. Select (highlight) the message to be promoted, demoted, or suppressed from the messages displayed in the upper section.

2. Select the Suppress Message, Make Error, Make Warning, or Make Note button to move the selected message from the upper section to the lower section. The selected message is repopulated in the lower section with the Override column reflecting the disposition of the message according to the button selected.

Allowed Severity Changes

Allowed severity levels and preference settings for warning, note, and advisory messages are:

- Promote – warning to error, note to warning, note to error
- Demote – warning to note
- Suppress – suppress warning, suppress note, suppress advisory

---

Note: Normal error messages (messages generated by default) cannot be suppressed or changed to a lesser severity level.

When using the dialog box:

- Use the control and shift keys to select multiple messages.
- If an `srr` file is not present (for example, if you are starting a new project) the table will be empty. Run the design at least once to generate an `srr` file.
- Clicking the OK button saves the message status changes to the *projectName*.pfl file in the project directory.

Message Reporting

The compiler and mapper must be rerun before the impact of the message status changes can be seen in the updated log file.

When a *projectName*.pfl input file is present at the start of the run, the message-status changes in the file are forwarded to the mapper and compiler which generate an updated log file. Depending on the changes specified:

- If an ID is promoted to an error, the mapper/compiler stops execution at the first occurrence of the message and prints the message in the @E:*msgID* :*messageText* format

Copyright © 2013 Synopsys, Inc.

- If an ID is promoted to a warning, the mapper/compiler prints the message in the @W:*msgID* :*messageText* format.
- If an ID is demoted to a note, the mapper/compiler prints the message in the @N:*msgID* :*messageText* format.
- If an ID is suppressed, the mapper/compiler excludes the message from the `srr` file.

Note: The online, error-message help documentation is unchanged by any message modification performed by the filtering mechanism. If a message is initially categorized as a warning in the synthesis tool, it continues to be reported as a warning in error-message help irrespective its promotion/demotion status.

Updating the *projectName*.pfl file

The *projectName*.pfl file in the top-level project directory stores the user message filter settings from the Log File Filter dialog box for that project. This file can be edited with a text editor. The file entry syntax is:

message_override -suppress *ID* [*ID* ...] | **-error** *ID* [*ID* ...] | **-warning** *ID* [*ID* ...] | **-note** *ID* [*ID* ...]

For example, to override the default message definition for note FX702 as a warning, enter:

```
message_override -warning FX702
```

Note: After editing the `pfl` file, close and reopen the project to update the overrides.

Handling Warnings

If you get warnings (@W prefix) after a synthesis run, do the following:

- Read the warning message and decide if it is something you need to act on, or whether you can ignore it.

- If the message is not self-explanatory or if you are unsure about how to handle the error, click the message ID in either the message window or HTML log file or double click the message ID in the ASCII text log file. These actions take you to online information about the condition that generated the warning.

Continue on Error

By default, the compiler stops the compilation process as soon as an error is encountered in the design. As a result, a design having multiple problems can require multiple iterations to identify and correct all the design issues. This behavior often leads to long turn-around times, especially with complex designs that require long compilation times.

A “continue-on-error” feature is available for users of Synplify Premier to allow the compilation process to continue for certain, non-syntax-related compiler errors.

Compilation Process

The continue-on-error feature is based on the four stages of the compiler’s compilation process:

- Syntax Check/Parse
- Elaboration/Sizing
- Hardware Generation
- Optimization

Stage 1 – Syntax Check/Parse

During this stage, the compiler parses all files in the project and reports the syntax errors found in each file. If parse errors are found, the compilation process terminates at this point and does not continue to the next (elaboration/sizing) stage. Any errors reported during this stage must be corrected before compilation can advance to the next stage.

Copyright © 2013 Synopsys, Inc.

Stage 2 – Elaboration/Sizing

During the elaboration/sizing stage, if a compilation error is encountered, the design unit containing the errors is converted to a black box as shown in the following figure; the compiler continues the processing of the remaining design units and reports any errors found in this stage. The compiler automatically continues to the hardware-generation stage.

Stage 3 – Hardware Generation

During the hardware-generation stage, if compilation errors are encountered, the design unit containing the errors is converted to a black box; the compiler continues the processing of the remaining design units and reports any errors found in this stage. The compiler automatically continues to the optimization stage.

Stage 4 – Optimization

During the optimization stage, if compilation errors are encountered, the design unit containing the errors is converted to a black box; the compiler continues the processing of the remaining design units and reports any errors found in this stage. If errors are reported in stages 2, 3, or 4, compilation terminates with an error status; these errors must be corrected before any subsequent synthesis operations can be performed.

Enabling the Continue-on-Error Feature

The continue-on-error feature for the compiler in Synplify Premier is enabled in the UI by either:

- Enabling the Continue On Error checkbox on the Options tab of the Implementation Options dialog box

- Selecting Options->Configure Compile Point Process and enabling the Continue on Error checkbox in the dialog box

The continue-on-error feature can also be enabled from the Tcl Script window using the following set_option Tcl command:

```
set_option -continue_on_error 1
```

Error Reporting

With continue-on-error, the compiler reports multiple errors within a single design unit as well as multiple errors in a complete hierarchy.

As stated earlier, design units containing one or more errors are automatically black boxed and the compile operation is allowed to continue until all design units have been verified. If any errors are encountered, they must be corrected before any synthesis operations can be performed. To view the individual errors reported in the log file, either view the error messages in the Messages window or click Detailed report in the Compile Input section of the Run Status area.

Run Status								
Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	out-of-date	6	2	7	-	0m:01s	-	1/31/2013 11:02:21 AM
Premap Detailed report	out-of-date				0m:00s			
Map & Optimize Detailed report	out-of-date				0m:00s			

The following Verilog example contains seven compilation errors within the two sub-modules shown in the example hierarchy below:

Copyright © 2013 Synopsys, Inc.

Example for Hierarchical Error Reporting

In the following example, the locations of the errors are highlighted in red.

```
module sub2 ( input d,rst,set,clk,
   output reg q,
   input [7:0]in, input [2:0] raddr, waddr,
   output [7:0] out3,
   input [3:0] a, b,
   output reg [4:0] q1 );
   nomodule i0(in, clk);

always@(posedge clk)
   begin
   if(clk)
      q = d << -3;
   end
   reg [7:0] mem [7:0];
   initial
   begin
      $readmemb("data.dat",in);
   end

always @(posedge clk)
mem[waddr] <= in;
assign out3 = mem[raddr];
wire [2:0] temp;

always@(posedge clk)
   begin
      q1 <= temp | a;
   end
assign temp = a ** b;
endmodule

module sub1 ( input d,rst,set,clk,
   output reg q,
   input [7:0]in, input [2:0] raddr, waddr,
   output [7:0] out3,
   input [3:0] a, b,
   output reg [4:0] q1 );

always @(posedge clk)
   begin
      q <= data1 | test; // 2 errors reported
   end
reg q;
```

```
always @(posedge clk)
begin
   if (rst)
      q <= 1'b0;
   else
      q = d;
   end
sub2 u1 (.*);
endmodule

module top ( input d,rst,set,clk,
   output reg q,
   input [7:0]in, input [2:0] raddr, waddr,
   output [7:0] out3,
   input [3:0] a, b,
   output reg [4:0] q1 );
sub1 u2 (.*);
endmodule
```

The following shows the resulting seven error messages in the Messages window after compiling the above example.

7 errors, 2 warnings, 6 notes

Type	ID	Message
	CG389	Reference to undefined module nomodule
	CG203	Negative shift distance
	CG366	Second argument for $readmemb must be a memory.
	CG198	Exponents with more than one variable not allowed
	CG478	Expression size is either 0 or could not be determined.
	CG853	Attempting to evaluate unsized expression
	CS153	Can't mix blocking and non-blocking assignments to a variable
	CG141	Creating black box for nomodule

Continue-on-Error for Compile Points

The continue-on-error feature is also available for use with compile points and, when enabled, allows both the Synplify Pro and Synplify Premier mappers to black box the affected compile point and continue synthesis with the other compile points (for more information on using the continue-on-error feature with compile points, see *Continue on Error Mode*, on page 1106).

Copyright © 2013 Synopsys, Inc.

Validating Results for Physical Synthesis

Use the following checklist to validate the results of logic synthesis in your physical design flows. These points apply to physical design flows for Altera technologies.

1. Check that the logic synthesis run was successful. Check the following:
   - The Physical Synthesis switch was disabled.
   - Logic synthesis completed successfully.
   - Check the log file, as described in *Checking Log Results*, on page 514.
2. Check that you used the correct version of the place-and-route tool. See the Release Notes, Help->Online Documents->release_notes.pdf->*Third Party Tool Versions* for information.
3. Check for black boxes. Search the synthesis `srr` log file for black box.

   A design that contains black boxes errors out in the tool and should be eliminated from the design.
4. Check for combinational feedback loops. Search the synthesis `srr` log file for Found combinational loop.

   Combinational loops cause random timing analysis results that invalidate any comparison and should be eliminated from the design.
5. Make sure the clock constraints are correct. Check the Clock Relationships table in the `srr` log file.
6. Check that the forward annotated timing constraints are consistent with the post place-and-route timing constraints.

Altera forward annotation file	`.tcl`

7. Are the false and multi-cycle paths constraints correctly defined in the `sdc` file? Ensure that the back-annotation timing report (`srr` log file in the PAR directory) matches the report file.

   The file varies, depending on the vendor:

Altera report file	`.tan.rpt`

Copyright © 2013 Synopsys, Inc.

For Altera designs, there are a couple of additional points to check:

1. Check that the clocks are routed on global resources.
   - Check the Clock Path Skew numbers in the report file.
     Clocks routed on general routing resources usually result in large skews. Because the tool does not take clock skew into account, large skews can degrade the quality of results (QoR) and result in poor timing correlation. The name of the report file varies, depending on the vendor:

Altera report file	`.tan.rpt`

Copyright © 2013 Synopsys, Inc.

Congestion Analysis After Logic Synthesis

Synplify Premier tools for Xilinx Virtex-6 and later devices

Congestion analysis after logic synthesis lets you generate a congestion report to help you handle congested designs early in the design process and improve your turnaround time. This report provides congestion analysis and possible fixes for congestion problems, after running logic synthesis and Xilinx place and route when the design is fully placed. This feature can provide better routability for the design.

The Congestion Analysis Report can be applied in the following scenarios:

- After you have run logic synthesis. If place and route is run from:
  - Within the synthesis tool, set the following options on the place-and-route dialog box: Run Place & Route after Synthesis and Run Timing & Placement Backannotation + Generate Congestion Analysis Report.

The Congestion Analysis Report automatically gets generated after place and route is run.

– Outside the synthesis tool, you can specify the externally generated NCD file.

- After you run physical synthesis or Physical Plus, the Congestion Analysis Report is automatically generated.

To generate the Congestion Analysis Report when you run place and route from outside the synthesis tool, use the following procedure:

1. To run the Congestion Analysis report for the active implementation, select the Analysis->Generate Congestion Analysis menu option.

   – Specify the input P&R NCD file.
   – Specify the output file location for the congestion report.
   – Click the Generate button.

2. Click the View button to look at the results.

   Congestion analysis produces the following output:

   - Log file – Output <*design*>_congestion_est.rpt

   - Congestion displayed in the Physical Analyst Implementation Maps

   For details, see *Using the Congestion Analysis Report*, on page 540 and *Displaying Congestion in the Physical Analyst*, on page 542.

Copyright © 2013 Synopsys, Inc.

Using the Congestion Analysis Report

Open the Congestion Analysis Report from the:

- Synthesis tool GUI after running the Congestion Analysis Report.
- Log file (<*design*>_congestion_est.rpt) located in the current Implementation Results directory.
- Click on the Congestion Estimate Report link from the contents panel of the HTML log file.

The Congestion Analysis Report contains the following information:

- Path location of the NCD file used to generate the congestion report.
- Slices occupied—number of slices and its utilization
- LUT utilization
- Number of unique control sets
- Percentage of matched backannotated instances after they have been placed from the NCD file. The predictions for congestion are more accurate if this percentage is greater than 90%.
- Prediction of congestion:
  - Estimated global congestion [High | Medium | Low]
  - Estimated local congestion [High | Medium | Low]
- Top ten high fanout nets contributing to congestion—nets displayed with net name, driver, and fanout number

Copyright © 2013 Synopsys, Inc.

The following example shows the Estimated Routing Congestion Report with HIGH routing congestion and recommends ways to alleviate this congestion.

```
#######################################################################
######          Estimated Routing Congestion Report               ######
#######################################################################
Congestion report for Xilinx NCD file:
/slowfs/sbg_builds3/FastSynthesis/TesedaDataV6/external_par/external.ncd
Congestion report for Physical Plus

Matched Instances: 65510 out of 65510 (100.00%)
Occupied Slices: 10558 out of 37680 (28.02%)
Lut Usage: 19932 out of 150720 (13.22%)

Number of unique control sets:              140

Fanout                  Nets                      Avg-WireUsage
0 - 3                     59769                        8.07
4 - 9                      4257                       20.86
10 - 24                     671                       34.66
25 - 49                     254                      110.28
   > 49                     210                      128.65


Synplify Premier estimates HIGH routing congestion based on the estimated overall wire
usage. To alleviate routing congestion:
- Consider inserting up to two BUFGs on the top two high-fanout signals using
syn_insert_buffer attribute
- Consider forcing replication on high-fanout signals using syn_maxfan attribute
- Consider running Synplify Premier Physical Plus in decongest mode with
syn_placer_effort_level=DECONGEST attribute if the design fails to route

#=====================================================================#
#                 Top 10 high-fanout nets                             #
#=====================================================================#
Net: reset_n_i, Driven by Instance: reset_n_i, Fanout = 3316, WireUsage= 308
Net: SYS_MasterSync[1], Driven by Instance: SYS_MasterSync[1], Fanout = 1978, WireUsage= 131
Net: cpu_dr.ff1_i[15], Driven by Instance: ff1_0_RNILTB5[15], Fanout = 1805, WireUsage= 219
Net: SYS2x_load, Driven by Instance: SYS2x_load, Fanout = 1539, WireUsage= 149
Net: SYS_WaveForm_time[4], Driven by Instance:SYS_WaveForm_time[4],Fanout = 1538, WireUsage= 95
Net: BS1.reset_i_i, Driven by Instance: ff1_RNI6CE7[15], Fanout = 1386, WireUsage= 263
Net: SYS_smode, Driven by Instance: SYS_smode, Fanout = 1281, WireUsage= 150
Net: idelay_rst_idle_r17,Driven by Instance:tap_counter_RNI18PF_o6[1], Fanout = 1134,
WireUsage= 293
Net: sys2x_dr.ff1[15], Driven by Instance: ff1_0[15], Fanout = 1046, WireUsage= 149
Net: SYS_sflag[3], Driven by Instance: SYS_sflag[3], Fanout = 1025, WireUsage= 120
```


Copyright © 2013 Synopsys, Inc.

Displaying Congestion in the Physical Analyst

The output of the Congestion Analysis results updates the existing SRM file with backannotated placement information. The Physical Analyst tool can be opened using this SRM file to display the congestion Implementation Map.

The following Estimated Routing Congestion Report shows HIGH local congestion hot spots in the design but only MEDIUM routing congestion. The report recommends that you use the Routing Implementation Map in the Physical Analyst tool to help you eliminate congestion for the design.

```
#######################################################################
######          Estimated Routing Congestion Report               ######
#######################################################################
Congestion report for Xilinx NCD file: \\us01t3filer09\sbg_builds3\congestion_map\external.ncd
Congestion report for Physical Plus

Matched Instances: 332430 out of 336389 (98.82%)
Occupied Slices: 50061 out of 51840 (96.57%)
Lut Usage: 132187 out of 207360 (63.75%)

Number of unique control sets:              142

Fanout      Nets        Avg-WireUsage
 0 -  3     267391           4.20
 4 -  9     32575            4.72
10 - 24     4269            16.04
25 - 49     1530            39.81
   > 49     1503            36.88

Synplify Premier estimates HIGH localized congestion hotspots in the design.
Please use Routing Congestion Map feature in the Physical Analyst to analyze them.

Synplify Premier estimates MEDIUM routing congestion based on the estimated overall wire usage.
#######################################################################
```

The following example shows the Routing Implementation Map for the design.

For more information about Physical Analyst Implementation Maps, see *Using Implementation Maps in Physical Analyst*, on page 710.

Copyright © 2013 Synopsys, Inc.

CHAPTER 11

Analyzing with HDL Analyst and FSM Viewer

This chapter describes how to analyze logic in the HDL Analyst and FSM Viewer. These tools are only available in the Synplify Pro and Synplify Premier products, though you can purchase HDL Analyst as an option to the base Synplify product.

See the following for detailed procedures:

- Working in the Schematic Views, on page 546
- Exploring Design Hierarchy, on page 560
- Finding Objects, on page 568
- Crossprobing, on page 581
- Analyzing With the HDL Analyst Tool, on page 589
- Using the FSM Viewer, on page 606

For information about analyzing timing, see Chapter 14, *Analyzing Timing*. For information about using Synplify Premier Physical Analyst tool, see Chapter 13, *Analyzing Designs in Physical Analyst*.

Copyright © 2013 Synopsys, Inc.

Working in the Schematic Views

The HDL Analyst includes the RTL and Technology views, which are schematic views used to graphically analyze your design. In the Synplify product, these views are part of the optional HDL Analyst package. The RTL view is available after a design is compiled; the Technology view is available after a designed has been synthesized and contains technology-specific primitives. In the Synplify Premier product, a RTL Floorplan view is available after a floorplan has been created with physical constraint regions and synthesized for the device.

For detailed descriptions of these views, see Chapter 2 of the *Reference Manual*. This section describes basic procedures you use in the RTL and Technology views. The information is organized into these topics:

- Differentiating Between the Views, on page 547
- Opening the Views, on page 547
- Viewing Object Properties, on page 549
- Selecting Objects in the RTL/Technology Views, on page 553
- Working with Multisheet Schematics, on page 555
- Moving Between Views in a Schematic Window, on page 556
- Setting Schematic View Preferences, on page 557
- Managing Windows, on page 559

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

- Exploring Object Hierarchy by Pushing/Popping, on page 561
- Exploring Object Hierarchy of Transparent Instances, on page 567
- Browsing to Find Objects in HDL Analyst Views, on page 568
- Crossprobing, on page 581
- Analyzing With the HDL Analyst Tool, on page 589

Copyright © 2013 Synopsys, Inc.

Differentiating Between the Views

- The difference between the RTL and Technology views is that the RTL view is the view generated after compilation, while the Technology view is the view generated after mapping. The RTL view displays your design as a high-level, technology-independent schematic. At this high level of abstraction, the design is represented with technology-independent components like variable-width adders, registers, large muxes, state machines, and so on. This view corresponds to the `srs` netlist file generated by the software in the Synopsys proprietary format. For a detailed description, see Chapter 2 of the *Reference Manual.*
- The Technology view contains technology-specific primitives. It shows low-level, vendor-specific components such as look-up tables, cascade and carry chains, muxes, and flip-flops, which can vary with the vendor and the technology. This view corresponds to the `srm` netlist file, generated by the software in the Synopsys proprietary format. For a detailed description, see Chapter 2 of the *Reference Manual.*
- The Synplify Premier RTL Floorplan view displays a floorplan schematic that includes all the logic assigned to any physical constraint regions created on the device, as well as, all other logic of the design. This view uses the same high-level abstraction and technology-independent components of the RTL view.

Opening the Views

The procedure for opening an RTL or Technology view is similar; the main difference is the design stage at which these views are available.

Copyright © 2013 Synopsys, Inc.

To open an RTL view...	Start with a compiled design. To open a hierarchical RTL view, do one of the following: • Select HDL Analyst->RTL->Hierarchical View. • Click the RTL View icon ( ) (a plus sign inside a circle). • Double-click the `srs` file in the Implementation Results view. To open a flattened RTL view, select HDL Analyst->RTL->Flattened View.
To open a Technology view...	Start with a mapped (synthesized) design. To open a hierarchical Technology view, do one of the following: • Select HDL Analyst ->Technology->Hierarchical View. • Click the Technology View icon (NAND gate icon). • Double-click the `srm` file in the Implementation Results view. To open a flattened Technology view, select HDL Analyst-> Technology->Flattened View.
To open a Floorplan view	Start with a synthesized design that has been floorplanned with physical constraint regions. To open a RTL Floorplan view: • Select HDL Analyst->RTL->Floorplanned View. • Double-click the partitioned netlist (`srp`) file from the Implementation Results view.

All RTL and Technology views have the schematic on the right and a pane on the left that contains a hierarchical list of the objects in the design. This pane is called the Hierarchy Browser. The bar at the top of the window contains the name of the view, the kind of view, hierarchical level, and the number of sheets in the schematic. See *Hierarchy Browser*, on page 93 in the *Reference Manual* for a description of the Hierarchy Browser.

RTL View

Technology View

Viewing Object Properties

There are a few ways in which you can view the properties of objects.

1. To temporarily display the properties of a particular object, hold the cursor over the object. A tooltip temporarily displays the information. at the cursor and in the status bar at the bottom of the tool window.

2. Select the object, right-click, and select Properties. The properties and their values are displayed in a table.

   If you select an instance, you can view the properties of the associated pins by selecting the pin from the list. Similarly, if you select a port, you can view the properties on individual bits.

Copyright © 2013 Synopsys, Inc.

3. To flag objects by property, do the following with an open RTL/Technology view:
   - Set the properties you want to see by selecting Options->HDL Analyst Options->Visual Properties, and selecting the properties from the pull-down list. Some properties are only available in certain views.

   - Close the HDL Analyst Options dialog box.

– Enable View->Visual Properties. If you do not enable this, the software does not display the property flags in the schematics. The HDL Analyst annotates all objects in the current view that have the specified property with a rectangular flag that contains the property name and value. The software uses different colors for different properties, so you can enable and view many properties at the same time.

Example: Slow and New Properties

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

When you are working with filtered views, you can use the New property to quickly identify objects that have been added to the current schematic with commands like Expand. You can step through successive filtered views to determine what was added at each step. This can be useful when you are debugging your design.

The following figure expands one of the pins from the previous filtered view. The new instance added to the view has two flags: new and slow.

Copyright © 2013 Synopsys, Inc.

Using the orig_inst_of Property for Parameterized Modules

The compiler automatically uniquifies parameterized modules or instances. Properties are available to identify the RTL names of both uniquified and original modules or instances.

- inst_of property – identifies module or instance by uniquified name
- orig_inst_of property – identifies module or instance by its original name before it was uniquified

In the following example, top-level module (top) instantiates the module sub multiple times using different parameter values. The compiler uniquifies the module sub as: sub_3s, sub_1s, and sub_4s.

Top.v

```
module top (input clk, [7:0] din, output [7:0] dout);
   sub #(.W(3)) UUT1 (.clk, .din(din[2:0]), .dout(dout[2:0]));
   sub #(.W(1)) UUT2 (.clk, .din(din[3]), .dout(dout[3]));
   sub #(.W(4)) UUT3 (.clk, .din(din[7:4]), .dout(dout[7:4]));
endmodule

module sub #(parameter W = 0) (
   input clk,
   input [W-1:0] din,
   output logic [W-1:0] dout );
```

```
always@(posedge clk)
   begin
      dout <= din;
   end
endmodule
```

RTL View

TCL Command Example

Use the get_prop command with the orig_inst_of property to identify the original RTL name for the module:

```
% get_prop -prop orig_inst_of {v:sub_3s}
sub

% get_prop -prop orig_inst_of {i:UUT3}
sub
```

Selecting Objects in the RTL/Technology Views

For mouse selection, standard object selection rules apply: In selection mode, the pointer is shaped like a crosshair.

Copyright © 2013 Synopsys, Inc.

To select...	Do this...
Single objects	Click on the object in the RTL or Technology schematic, or click the object name in the Hierarchy Browser.
Multiple objects	Use one of these methods: • Draw a rectangle around the objects. • Select an object, press Ctrl, and click other objects you want to select. • Select multiple objects in the Hierarchy Browser. See *Browsing With the Hierarchy Browser*, on page 568. • Use Find to select the objects you want. See *Using Find for Hierarchical and Restricted Searches*, on page 570.
Objects by type (instances, ports, nets)	Use Edit->Find to select the objects (see *Browsing With the Find Command*, on page 569), or use the Hierarchy Browser, which lists objects by type.
All objects of a certain type (instances, ports, nets)	To select all objects of a certain type, do either of the following: • Right-click and choose the appropriate command from the Select All Schematic/Current Sheet popup menus. • Select the objects in the Hierarchy Browser.
No objects (deselect all currently selected objects)	Click the left mouse button in a blank area of the schematic or click the right mouse button to bring up the pop-up menu and choose Unselect All. Deselected objects are no longer highlighted.

The HDL Analyst view highlights selected objects in red. If the object you select is on another sheet of the schematic, the schematic tracks to the appropriate sheet. If you have other windows open, the selected object is highlighted in the other windows as well (crossprobing), but the other windows do not track to the correct sheet. Selected nets that span different hierarchical levels are highlighted on all the levels. See *Crossprobing*, on page 581 for more information about crossprobing.

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

Working with Multisheet Schematics

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

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

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

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

To view...	Use one of these methods...
Next sheet or previous sheet	Select View->Next/Previous Sheet. Press the right mouse button and draw a horizontal mouse stroke (left to right for next sheet, right to left for previous sheet). Click the icons: Next Sheet ( ) or Previous Sheet ( ) Press Shift-right arrow (Next Sheet) or Shift-left arrow (Previous sheet). Navigate with View->Back and View ->Forward if the next/previous sheets are part of the display history.
A specific sheet number	Select View->View Sheets and select the sheet. Click the right mouse button, select View Sheets from the popup menu, and then select the sheet you want. Press Ctrl-g and select the sheet you want.
Lower-level logic of a transparent instance on separate sheets	Check the sheet numbers indicated inside the empty transparent instance. Use the sheet navigation commands like Next Sheet or View Sheets to move to the sheet you need.

Copyright © 2013 Synopsys, Inc.

To view...	Use one of these methods...
All objects of a certain type	To highlight all the objects of the same type in the schematic, right-click and select the appropriate command from the Select All Schematic popup menu. To highlight all the objects of the same type on the current sheet, right-click and select the appropriate command from the Select All Sheet popup menu.
Selected items only	Filter the schematic as described in *Filtering Schematics*, on page 593.
A net across sheets	If there are no sheet numbers displayed in a hexagon at the end of the sheet connector, select Options ->HDL Analyst Options and enable Show Sheet Connector Index. Right-click the sheet connector and select the sheet number from the popup as shown in the following figure.

Sheet Connector Symbol

Connected sheet numbers

Sheet connector with multisheet popup menu

Moving Between Views in a Schematic Window

When you filter or expand your design, you move through a number of different design views in the same schematic window. For example, you might start with a view of the entire design, zoom in on an area, then filter an object, and finally expand a connection in the filtered view, for a total of four views.

1. To move back to the previous view, click the Back icon or draw the appropriate mouse stroke.

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

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

   The software displays the next view in the display history.

Setting Schematic View Preferences

You can set various preferences for the RTL and Technology views from the user interface.

1. Select Options->HDL Analyst Options. For a description of all the options on this form, see *HDL Analyst Options Command*, on page 345 in the *Reference Manual*.

2. The following table details some common operations:

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

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

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

Copyright © 2013 Synopsys, Inc.

4. To control the display of labels, first enable the Text->Show Text option, and then enable the Label Options you want. The following figure illustrates the label that each option controls.

For a more detailed information about some of these options, see *Schematic Display*, on page 423 in the *Reference Manual*.

5. Click OK on the HDL Analyst Options form.

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

Managing Windows

As you work on a project, you open different windows. For example, you might have two Technology views, an RTL view, and a source code window open. The following guidelines help you manage the different windows you have open. For information about cycling through the display history in a single schematic, see *Moving Between Views in a Schematic Window*, on page 556.

1. Toggle on View->Workbook Mode.

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

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

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

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

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

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

Copyright © 2013 Synopsys, Inc.

Exploring Design Hierarchy

Schematics generally have a certain amount of design hierarchy. You can move between hierarchical levels using the Hierarchy Browser or Push/Pop mode. For additional information, see *Analyzing With the HDL Analyst Tool*, on page 589. The topics include:

- Traversing Design Hierarchy with the Hierarchy Browser, on page 560
- Exploring Object Hierarchy by Pushing/Popping, on page 561
- Exploring Object Hierarchy of Transparent Instances, on page 567

Traversing Design Hierarchy with the Hierarchy Browser

The Hierarchy Browser is the list of objects on the left side of the RTL and Technology views. It is best used to get an overview, or when you need to browse and find an object. If you want to move between design levels of a particular object, Push/Pop mode is more direct. Refer to *Exploring Object Hierarchy by Pushing/Popping*, on page 561 for details.

The hierarchy browser allows you to traverse and select the following:

- Instances and submodules
- Ports
- Internal nets
- Clock trees (in an RTL view)

The browser lists the objects by type. A plus sign in a square icon indicates that there is hierarchy under that object and a minus sign indicates that the design hierarchy has been expanded. To see lower-level hierarchy, click on the plus sign for the object. To ascend the hierarchy, click on the minus sign.

Copyright © 2013 Synopsys, Inc.

Refer to *Hierarchy Browser Symbols*, on page 94 in the *Reference Manual* for an explanation of the symbols.

Exploring Object Hierarchy by Pushing/Popping

To view the internal hierarchy of a specific object, it is best to use Push/Pop mode or examine transparent instances, instead of using the Hierarchy Browser described in *Traversing Design Hierarchy with the Hierarchy Browser*, on page 560. You can access Push/Pop mode with the Push/Pop Hierarchy icon, the Push/Pop Hierarchy command, or mouse strokes.

When combined with other commands like filtering and expansion commands, Push/Pop mode can be a very powerful tool for isolating and analyzing logic. See *Filtering Schematics*, on page 593, *Expanding Pin and Net Logic*, on page 595, and *Expanding and Viewing Connections*, on page 599 for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

- Pushing into Objects, on page 562, next
- Popping up a Hierarchical Level, on page 565

Copyright © 2013 Synopsys, Inc.

Pushing into Objects

In the schematic views, you can push into objects and view the lower-level hierarchy. You can use a mouse stroke, the command, or the icon to push into objects:

1. To move down a level (push into an object) with a mouse stroke, put your cursor near the top of the object, hold down the right mouse button, and draw a vertical stroke from top to bottom. You can push into the following objects; see step 3 for examples of pushing into different types of objects.
   - Hierarchical instances. They can be displayed as pale yellow boxes (opaque instances) or hollow boxes with internal logic displayed (transparent instances). You cannot push into a hierarchical instance that is hidden with the Hide Instance command (internal logic is hidden).

   - Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the Technology view, under Instances - Primitives.
   - Inferred ROMs and state machines.

   The remaining steps show you how to use the icon or command to push into an object.

2. Enable Push/Pop mode by doing one of the following:
   - Select View->Push/Pop Hierarchy.
   - Right-click in the Technology view and select Push/Pop Hierarchy from the popup menu.
   - Click the Push/Pop Hierarchy icon ( ) in the toolbar (two arrows pointing up and down).
   - Press F2.

   The cursor changes to an arrow. The direction of the arrow indicates the underlying hierarchy, as shown in the following figure. The status bar at the bottom of the window reports information about the objects over which you move your cursor.

3. To push (descend) into an object, click on the hierarchical object. For a transparent instance, you must click on the pale yellow border. The following figure shows the result of pushing into a ROM.

   When you descend into a ROM, you can push into it one more time to see the ROM data table. The information is in a view-only text file called rom.info.

Copyright © 2013 Synopsys, Inc.

Similarly, you can push into a state machine. When you push into an FSM from the RTL view, you open the FSM viewer where you can graphically view the transitions. For more information, see *Using the FSM Viewer*, on page 606. If you push into a state machine from the Technology view, you see the underlying logic.

Popping up a Hierarchical Level

1. To move up a level (pop up a level), put your cursor anywhere in the design, hold down the right mouse button, and draw a vertical mouse stroke, moving from the bottom upwards.

Copyright © 2013 Synopsys, Inc.

The software moves up a level, and displays the next level of hierarchy.

2. To pop (ascend) a level using the commands or icon, do the following:
   - Select the command or icon if you are not already in Push/Pop mode. See *Pushing into Objects*, on page 562for details.
   - Move your cursor to a blank area and click.
3. To exit Push/Pop mode, do one of the following:
   - Click the right mouse button in a blank area of the view.
   - Deselect View->Push/Pop Hierarchy.
   - Deselect the Push/Pop Hierarchy icon.
   - Press F2.

Exploring Object Hierarchy of Transparent Instances

Examining a transparent instance is one way of exploring the design hierarchy of an object. The following table compares this method with pushing (described in *Exploring Object Hierarchy by Pushing/Popping*, on page 561).

	Pushing	Transparent Instance
User control	You initiate the operation through the command or icon.	You have no direct control; the transparent instance is automatically generated by some commands that result in a filtered view.
Design context	Context lost; the lower-level logic is shown in a separate view	Context maintained; lower-level logic is displayed inside a hollow yellow box at the hierarchical level of the parent.

Copyright © 2013 Synopsys, Inc.

Finding Objects

In the schematic views, you can use the Hierarchy Browser or the Find command to find objects, as explained in these sections:

- Browsing to Find Objects in HDL Analyst Views, on page 568
- Using Find for Hierarchical and Restricted Searches, on page 570
- Using Wildcards with the Find Command, on page 573
- Using Find to Search the Output Netlist, on page 578

For information about the Tcl Find command, which you use to locate objects, and create collections, see *Tcl find Command*, on page 1774 in the *Reference Manual*.

Browsing to Find Objects in HDL Analyst Views

You can always zoom in to find an object in the RTL and Technology schematics. The following procedure shows you how to browse through design objects and find an object at any level of the design hierarchy. You can use the Hierarchy Browser or the Find command to do this. If you are familiar with the design hierarchy, the Hierarchy Browser can be the quickest method to locate an object. The Find command is best used to graphically browse and locate the object you want.

Browsing With the Hierarchy Browser

1. In the Hierarchy Browser, click the name of the net, port, or instance you want to select.

   The object is highlighted in the schematic.

2. To select a range of objects, select the first object in the range. Then, scroll to display the last object in the range. Press and hold the Shift key while clicking the last object in the range.

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

3. If the object is on a lower hierarchical level, do either of the following:
   - Expand the appropriate higher-level object by clicking the plus symbol next to it, and then select the object you want.

– Push down into the higher-level object, and then select the object from the Hierarchy Browser.

The selected object is highlighted in the schematic. The following example shows how moving down the object hierarchy and selecting an object causes the schematic to move to the sheet and level that contains the selected object.

4. To select all objects of the same type, select them from the Hierarchy Browser. For example, you can find all the nets in your design.

Browsing With the Find Command

1. In a schematic view, select HDL Analyst->Find or press Ctrl-f to open the Object Query dialog box.

2. Do the following in the dialog box:

   – Select objects in the selection box on the left. You can select all the objects or a smaller set of objects to browse. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the dialog box.

Copyright © 2013 Synopsys, Inc.

– Click the arrow to move the selected objects over to the box on the right.

The software highlights the selected objects.

3. In the Object Query dialog box, click on an object in the box on the right.

   The software tracks to the schematic page with that object.

Using Find for Hierarchical and Restricted Searches

You can always zoom in to find an object in the RTL and Technology schematics or use the Hierarchy Browser (see *Browsing to Find Objects in HDL Analyst Views*, on page 568). This procedure shows you how to use the Find command to do hierarchical object searches or restrict the search to the current level or the current level and its underlying hierarchy.

Note that Find only adds to the current selection; it does not deselect anything that is already selected. you can use successive searches to build up exactly the selection you need, before filtering.

1. If needed, restrict the range of the search by filtering the view.

   See *Viewing Design Hierarchy and Context*, on page 590 and *Filtering Schematics*, on page 593 for details. With a filtered view, the software only searches the filtered instances, unless you set the scope of the search to Entire Design, as described below, in which case Find searches the entire design.

   You can use the filtering technique to restrict your search to just one schematic sheet. Select all the objects on one sheet and filter the view. Continue with the procedure.

2. To further restrict the range of the search, hide instances you do not need.

   You can do this in addition to filtering the view, or instead of filtering the view. Hidden instances and their hierarchy are excluded from the search. When you have finished the search, use the Unhide Instances command to make the hierarchy visible again.

3. Open the Object Query dialog box.

– Do one of the following: right click in the RTL or Technology view and select Find from the popup menu, press Ctrl-f, or click the Find icon ( ).
– Reposition the dialog box so you can see both your schematic and the dialog box.

4. Select the tab for the type of object. The Unhighlighted box on the left lists all objects of that type (instances, symbols, nets, or ports).

   For fastest results, search by Instances rather than Nets. When you select Nets, the software loads the whole design, which could take some time.

5. Click one of these buttons to set the hierarchical range for the search: Entire Design, Current Level & Below, or Current Level Only, depending on the hierarchical level of the design to which you want to restrict your search.

   The range setting is especially important when you use wildcards. See *Effect of Hierarchy and Range on Wildcard Searches*, on page 573 for details. Current Level Only or Current Level & Below are useful for searching filtered schematics or critical path schematics.

   The lower-level details of a transparent instance appear at the current level and are included in the search when you set it to Current Level Only. To exclude them, temporarily hide the transparent instances, as described in step 2.

Copyright © 2013 Synopsys, Inc.

Use Entire Design to hierarchically search the whole design. For large hierarchical designs, reduce the scope of the search by using the techniques described in the first step.

The Unhighlighted box shows available objects within the scope you set. Objects are listed in alphabetical order, not hierarchical order.

6. To search for objects in the mapped database or the output netlist, set the Name Space option.

   The name of an object might be changed because of synthesis optimizations or to match the place-and-route tool conventions, so that the object name may no longer match the name in the original netlist. Setting the Name Space option ensures that the Find command searches the correct database for the object. For example, if you set this option to Tech View, the tool searches the mapped database (`srm`) for the object name you specify. For information about using this feature to find objects from an output netlist, see *Using Find to Search the Output Netlist*, on page 578.

7. Do the following to select objects from the list. To use wildcards in the selection, see the next step.
   - Click on the objects you want from the list. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the dialog box.
   - Click Find 200 or Find All. The former finds the first 200 matches, and then you can click the button again to find the next 200.
   - Click the right arrow to move the objects into the box on the right, or double-click individual names.

   The schematic displays highlighted objects in red.

8. Do the following to select objects using patterns or wildcards.
   - Type a pattern in the Highlight Wildcard field. See *Using Wildcards with the Find Command*, on page 573 for a detailed discussion of wildcards.

     The Unhighlighted list shows the objects that match the wildcard criteria. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the form.

– Click the right arrow to move the selections to the box on the right, or double-click individual names. The schematic displays highlighted objects in red.

You can use wildcards to avoid typing long pathnames. Start with a general pattern, and then make it more specific. The following example browses and uses wildcards successively to narrow the search.

Find all instances three levels down	`*.*.*`
Narrow search to find instances that begin with i_	`i_*.*.*`
Narrow search to find instances that begin with un2 after the second hierarchy separator	`i_*.*.un2*`

Note that there are some differences when you specify the find command in the RTL view, Technology view, or the constraint file.

9. You can leave the dialog box open to do successive Find operations. Click OK or Cancel to close the dialog box when you are done.

For detailed information about the Find command and the Object Query dialog box, see *Find Command (HDL Analyst)*, on page 152 of the *Reference Manual*.

Using Wildcards with the Find Command

Use the following wildcards when you search the schematics:

*	The asterisk matches any sequence of characters.
?	The question mark matches any single character.
.	The dot explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not a hierarchy separator, type a backslash before the dot: \.

Effect of Hierarchy and Range on Wildcard Searches

The asterisk and question mark wildcards do not cross hierarchical boundaries, but search each level of hierarchy individually with the search pattern. This default is affected by the following:

- Hierarchical separators

Copyright © 2013 Synopsys, Inc.

Dots match hierarchy separators, unless you use the backslash escape character in front of the dot (\.). Hierarchical search patterns with a dot (l*.*) are repeated at each level included in the scope. If you use the `*.*` pattern with Current Level, the software matches non-hierarchical names at the current level that include a dot.

- Search range

  The scope of the search determines the starting point for the searches. Some times the starting point might make it appear as if the wildcards cross hierarchical boundaries. If you are at 2A in the following figure and the scope of the search is set to Current Level and Below, separate searches start at 2A, 3A1, and 3A2. Each search does not cross hierarchical boundaries. If the scope of the search is Entire Design, the wildcard searches run from each hierarchical point (1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3). The result of an asterisk search (*) with Entire Design is a list of all matches in the design, regardless of the current level.

See *Wildcard Search Examples*, on page 575 for examples.

How a Wildcard Search Works

1. The starting point of a wildcard search depends on the range set for the search.

Entire Design	Starts at top level and uses the pattern to search from that level. It then moves to any child levels below the top level and searches them. The software repeats the search pattern at each hierarchical point in the design until it searches the entire design.
Current Level	Starts at the current hierarchical level and searches that level only. A search started at 2A only covers 2A.
Current Level and Below	Starts at the current hierarchical level and searches that level. It then moves to any child levels below the starting point and conducts separate searches from each of these starting points.

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

   Dots match hierarchy separators, unless you use the backslash escape character in front of the dot (\.). Hierarchical search patterns with a dot (l*.*) are repeated at each level included in the scope. See *Effect of Hierarchy and Range on Wildcard Searches*, on page 573 and *Wildcard Search Examples*, on page 575 for details and examples, respectively. If you use the `*.*` pattern with Current Level, the software matches non-hierarchical names at the current level that include a dot.

Wildcard Search Examples

The figure shows a design with three hierarchical levels, and the table shows the results of some searches on this design.

Copyright © 2013 Synopsys, Inc.

Scope	Pattern	Starting Point	Finds Matches in...
Entire Design	*	3A1	1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3 (* at all levels)
	.	2B	2A and 2B (*.* from 1) 3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B) No matches in 1 (because of the hierarchical dot), unless a name includes a non-hierarchical dot.
Current Level	*	1	1 only (no hierarchical boundary crossing)
	.	2B	2B only. No search of lower levels even though the dot is specified, because the scope is Current Level. No matches, unless a 2B name includes a non-hierarchical dot.
Current Level and Below	*	2A	2A only (no hierarchical boundary crossing)
	.	1	2A and 2B (*.* from 1) 3A1, 3A2, 3B1, 3B2, and 3B3 (*.* from 2A and 2B) No matches from 1, because the dot is specified.
	.	2B	3B1, 3B2, and 3B3 (*.* from 2B)
	.	3A2	No matches (no hierarchy below 3A2)
	..*	1	3A1, 3A2, 3B1, 3B2, and 3B3 (*.*.* from 1) Search ends because there is no hierarchy two levels below 2A and 2B.

Difference from Tcl Search

In a simple Tcl search like the one used by ISE in UCF files, no character (except the backslash, \) has special meaning. This means that the asterisk matches everything in a string. The FPGA synthesis tools and Synopsys TimeQuest and Design Compiler products confine the simple search to within one level of hierarchy. The following command searches each level of hierarchy individually for the specified pattern:

```
find -hier *abc*addr_reg[*]
```

If you want to go through the hierarchy, you must add the hierarchy separators to the search pattern:

```
find {*.*.abc.*.*.addr_reg[*]}
```

The following shows the equivalent SDC commands to the simple Tcl searches in UCF:

UCF	SDC Equivalent
INST “*ctrlfifo*" TNM = “FIFO_GRP";_	set FIFOS [find –hier -inst {i:*ctrlfifo*}]
INST “*ctrlfifo" TNM = “FIFO_GRP";_	set FIFOS [find –hier -inst {i:*ctrlfifo}]
INST “ctrlfifo*" TNM = “FIFO_GRP";_	set FIFOS [find –hier -inst * -filter @hier_rtl_name == ctrlfifo*]
INST “ctrlfifo*hier_inst" TNM = “FIFO_GRP";_	set FIFOS [find –hier -inst * -filter @hier_rtl_name == ctrlfifo*hier_inst]

Find Command Differences in HDL Analyst Views and Constraint File

There are some slight differences when you use the Find command in the RTL view, Technology view, and the constraint files:

- You cannot use find to search for bit registers of a bit array in the RTL or Technology views, but you can specify it in a constraint file, where the following command will work:

  ```
  find –seq {i:modulex_inst.qb[7]}
  ```

  In a HDL Analyst view, you cannot find {i:modulex_inst.qb[7]}, but you can specify and find {i:modulex_inst.qb[7:0]}.

- By default, the following Tcl command does not find objects in the RTL view, although it does find objects in the Technology view:

  ```
  –hier –seq * -filter @clock == clk75
  ```

  To make this work in an RTL view, you must turn on Annotated Properties for Analyst in the Device tab of the Implementation Options dialog box, recompile the design, and then open a new RTL view.

Copyright © 2013 Synopsys, Inc.

Combining Find with Filtering to Refine Searches

You can combine the Find command with the filtering commands to better effect. Depending on what you want to do, use the Find command first, or a filtering command.

1. To limit the range of a search, do the following:
   - Filter the design.
   - Use the Find command on the filtered view, but set the search range to Current Level Only.
2. Select objects for a filtered view.
   - Use the Find command to browse and select objects.
   - Filter the objects to display them.

Using Find to Search the Output Netlist

When the synthesis tool creates an output netlist like a `vqm` or `edf` file, some names are optimized for use in the P&R tool. When you debug your design for place and route looking for a particular object, use the Name Space option in the Object Query dialog box to locate the optimized names in the output netlist. The following procedure shows you how to locate an object, highlight and filter it in the Technology view, and crossprobe to the source code for editing.

1. Select the output netlist file option in the Implementations Results tab of the Implementation Options dialog box.
2. After you synthesize your design, open your output netlist file and select the name of the object you want to find.

Copyright © 2013 Synopsys, Inc.

3. Copy the name and open a Technology view.
4. In the Technology view, press Ctrl-f or select Edit->Find to open the Object Query dialog box and do the following:
   - Paste the object name you copied into the Highlight Search field.
   - Set the Name Space option to Netlist and click Find All.

Search by Tech View

Search by Netlist

Copyright © 2013 Synopsys, Inc.

If you leave the Name Space option set to the default of Tech View, the tool does not find the name because it is searching the mapped database instead of the output netlist.

– Double click the name to move it into the Highlighted field and close the dialog box.

In the Technology view, the name is highlighted in the schematic.

5. Select HDL Analyst->Filter Schematic to view only the highlighted portion of the schematic.

Filtered View

The tooltip shows the equivalent name in the Technology view.

6. Double click on the filtered schematic to crossprobe to the corresponding code in the HDL file.

Crossprobing

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. Highlighting a line of text, for example, highlights the corresponding logic in the schematic views. Crossprobing helps you visualize where coding changes or timing constraints might help to reduce area or improve performance.

You can crossprobe between the RTL view, Technology view, the FSM Viewer (not available in the Synplify product), the log file, the source files, and some external text files from place-and-route tools. However, not all objects or source code crossprobe to other views, because some source code and RTL view logic is optimized away during the compilation or mapping processes.

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

- Crossprobing within an RTL/Technology View, on page 581
- Crossprobing from the RTL/Technology View, on page 582
- Crossprobing from the Text Editor Window, on page 584
- Crossprobing from the Tcl Script Window, on page 587
- Crossprobing from the FSM Viewer, on page 587

Crossprobing within an RTL/Technology View

Selecting an object name in the Hierarchy Browser highlights the object in the schematic, and vice versa.

Selected Object	Highlighted Object
Instance in schematic (single-click)	Module icon in Hierarchy Browser
Net in schematic	Net icon in Hierarchy Browser
Port in schematic	Port icon in Hierarchy Browser
Logic icon in Hierarchy Browser	Instance in schematic
Net icon in Hierarchy Browser	Net in schematic
Port icon in Hierarchy Browser	Port in schematic

Copyright © 2013 Synopsys, Inc.

In this example, when you select the DECODE module in the Hierarchy Browser, the DECODE module is automatically selected in the RTL view.

Crossprobing from the RTL/Technology View

1. To crossprobe from an RTL or Technology views to other open views, select the object by clicking on it.

   The software automatically highlights the object in all open views. If the open view is a schematic, the software highlights the object in the Hierarchy Browser on the left as well as in the schematic. If the highlighted object is on another sheet of a multi-sheet schematic, the view does not automatically track to the page. If the crossprobed object is inside a hidden instance, the hidden instance is highlighted in the schematic.

   If the open view is a source file, the software tracks to the appropriate code and highlights it. The following figure shows crossprobing between the RTL, Technology, and Text Editor (source code) views.

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

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

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

Copyright © 2013 Synopsys, Inc.

From	To	Procedure
RTL	Source code	Double-click an object. If the source code file is not open, the software opens the Text Editor window to the appropriate section of code. If the source file is already open, the software scrolls to the correct section of the code and highlights it.
RTL	Technology	The Technology view must be open. Click the object to highlight and crossprobe.
RTL	FSM Viewer (not in Synplify)	The FSM view must be open. The state machine must be coded with a onehot encoding style. Click the FSM to highlight and crossprobe.
Technology	Source code	If the source code file is already, open, the software scrolls to the correct section of the code and highlights it. If the source code file is not open, double-click an object in the Technology view to open the source code file.
Technology	RTL	The RTL view must be open. Click the object to highlight and crossprobe.

Crossprobing from the Text Editor Window

To crossprobe from a source code window or from the log file to an RTL, Technology, or FSM view, use this procedure. You can use this method to crossprobe from any text file with objects that have the same instance names as in the synthesis software. For example, you can crossprobe from place-and-route files. See *Example of Crossprobing a Path from a Text File*, on page 585 for a practical example of how to use crossprobing.

1. Open the RTL, FSM, or Technology view to which you want to crossprobe. The FSM view is not a Synplify feature.

2. To crossprobe from an error, warning, or note in the html log file, click on the file name to open the corresponding source code in another Text Editor window; to crossprobe from a text log file, double-click on the text of the error, warning, or note.

3. To crossprobe from a third-party text file (not source code or a log file), select Options->HDL Analyst Options->General, and enable Enhanced text crossprobing.

4. Select the appropriate portion of text in the Text Editor window. In some cases, it may be necessary to select an entire block of text to crossprobe.

   The software highlights the objects corresponding to the selected code in all the open windows. For example, if you select a state name in the code, it highlights the state in the FSM viewer. If an object is on another schematic sheet or on another hierarchical level, the highlighting might not be obvious. If you filter the RTL or schematic view (right-click in the source code window with the selected text and select Filter Schematic from the popup menu), you can isolate the highlighted objects for easy viewing.

Example of Crossprobing a Path from a Text File

This example selects a path in a log file and crossprobes it in the Technology view. You can use the same technique to crossprobe from other text files like place-and-route files, as long as the instance names in the text file match the instance names in the synthesis tool.

1. Open the log file, the RTL, and Technology views.
2. Select the path objects in the log file.
   - Select the column by pressing Alt and dragging the cursor to the end of the column. On the Linux platform, use the key to which the Alt function is mapped; this is usually the Ctrl-Alt key combination.
   - To select all the objects in the path, right-click and choose Select in Analyst from the popup menu. Alternatively, you can select certain objects only, as described next.

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

Copyright © 2013 Synopsys, Inc.

Text Editor

Technology View

– To further filter the objects in the path, right-click and choose Select From from the popup menu. On the form, check the objects you want, and click OK. Only the corresponding objects are highlighted.

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

   You see just the selected objects.

Crossprobing from the Tcl Script Window

Crossprobing from the Tcl script window is useful for debugging error messages. You cannot do this with the Synplify product, because it does not have the Tcl window feature.

To crossprobe from the Tcl Script window to the source code, double-click a line in the Tcl window. To crossprobe a warning or error, first click the Messages tab and then double-click the warning or error. The software opens the relevant source code file and highlights the corresponding code.

Crossprobing from the FSM Viewer

You can crossprobe to the FSM Viewer if you have the FSM view open. You can crossprobe from an RTL, Technology, or source code window. The Synplify tool does not support the FSM viewer.

Copyright © 2013 Synopsys, Inc.

To crossprobe from the FSM Viewer, do the following:

1. Open the view to which you want to crossprobe: RTL/Technology view, or the source code file.
2. Do the following in the open FSM view:
   - For FSMs with a onehot encoding style, click the state bubbles in the bubble diagram or the states in the FSM transition table.
   - For all other FSMs, click the states in the bubble diagram. You cannot use the transition table because with these encoding styles, the number of registers in the RTL or Technology views do not match the number of registers in the FSM Viewer.

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

Copyright © 2013 Synopsys, Inc.

Analyzing With the HDL Analyst Tool

The HDL Analyst tool is a graphical productivity tool that helps you visualize your synthesis results. It consists of RTL-level and technology-primitive level schematics that let you graphically view and analyze your design.

- RTL View
  Using BEST® (Behavior Extracting Synthesis Technology) in the RTL view, the software keeps a high-level of abstraction and makes the RTL view easy to view and debug. High-level structures like RAMs, ROMs, operators, and FSMs are kept as abstractions in this view instead of being converted to gates. You can examine the high-level structure, or push into a component and view the gate-level structure.
- Technology View
  The software uses module generators to implement the high-level structures from the RTL view, and maps them to technology-specific resources.

To analyze information, compare the current view with the information in the RTL/Technology view, the log file, the FSM view, and the source code, you can use techniques like crossprobing, flattening, and filtering. Note that Synplify users do not have access to the FSM viewer. See the following for more information about analysis techniques.

- Viewing Design Hierarchy and Context, on page 590
- Filtering Schematics, on page 593
- Expanding Pin and Net Logic, on page 595
- Expanding and Viewing Connections, on page 599
- Flattening Schematic Hierarchy, on page 600
- Minimizing Memory Usage While Analyzing Designs, on page 605

For additional information about navigating the HDL Analyst views or using other techniques like crossprobing, see the following:

- Working in the Schematic Views, on page 546
- Exploring Design Hierarchy, on page 560
- Finding Objects, on page 568
- Crossprobing, on page 581

Copyright © 2013 Synopsys, Inc.

Viewing Design Hierarchy and Context

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

This section describes how to use interactive hierarchical viewing operations to better analyze your design. Automatic hierarchy viewing operations that are built into other commands are described in the context in which they appear. For example, *Viewing Critical Paths*, on page 725 describes how the software automatically traces a critical path through different hierarchical levels using hollow boxes with nested internal logic (transparent instances) to indicate levels in hierarchical instances.

1. To view the internal logic of primitives in your design, do either of the following:
   - To view the logic of an individual primitive, push into it. This generates a new schematic view with the internal details. Click the Back icon to return to the previous view.
   - To view the logic of all primitives in the design, select Options->HDL Analyst Options->General, and enable Show Cell Interior. This command lets you see internal logic in context, by adding the internal details to the current schematic view and all subsequent views. If the view is too cluttered with this option on, filter the view (see *Filtering Schematics*, on page 593) or push into the primitive. Click the Back icon to return to the previous view after filtering or pushing into the object.

The following figure compares these two methods:

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

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

2. To hide selected hierarchy, select the instance whose hierarchy you want to exclude, and then select Hide Instances from the HDL Analyst menu or the right-click popup menu in the schematic view.

   You can hide opaque (solid yellow) or transparent (hollow) instances. The software marks hidden instances with an H in the lower left. Hidden instances are like black boxes; their hierarchy is excluded from filtering, expanding, dissolving, or searching in the current window, although they can be crossprobed. An instance is only hidden in the current view window; other view windows are not affected. Temporarily hiding unnecessary hierarchy focuses analysis and saves time in large designs.

   Before you save a design with hidden instances, select Unhide Instances from the HDL Analyst menu or the right-click popup menu and make the hidden internal hierarchy accessible again. Otherwise, the hidden instances are saved as black boxes, without their internal logic. Conversely, you can use this feature to reduce the scope of analysis in a large design by hiding instances you do not need, saving the reduced design to a new name, and then analyzing it.

3. To view the internal logic of a hierarchical instance, you can push into the instance, dissolve the selected instance with the Dissolve Instances command, or flatten the design. You cannot use these methods to view the internal logic of a hidden instance.

Copyright © 2013 Synopsys, Inc.

Pushing into an instance	Generates a view that shows only the internal logic. You do not see the internal hierarchy in context. To return to the previous view, click Back. See *Exploring Object Hierarchy by Pushing/Popping*, on page 561 for details.
Flattening the entire design	Opens a new view where the entire design is flattened, except for hidden hierarchy. Large flattened designs can be overwhelming. See *Flattening Schematic Hierarchy*, on page 600 for details about flattening designs. Because this is a new view, you cannot use Back to return to the previous view. To return to the top-level unflattened schematic, right-click in the view and select Unflatten Schematic.
Flattening an instance by dissolving	Generates a view where the hierarchy of the selected instances is flattened, but the rest of the design is unaffected. This provides context. See *Flattening Schematic Hierarchy*, on page 600 for details about dissolving instances.

4. If the result of filtering or dissolving is a hollow box with no internal logic, try either of the following, as appropriate, to view the internal hierarchy:
   - Select Options->HDL Analyst Options->Sheet Size and increase the value of Maximum Filtered Instances. Use this option if the view is not too cluttered.
   - Use the sheet navigation commands to go to the sheets indicated in the hollow box.

   If there is too much internal logic to display in the current view, the software puts the internal hierarchy on separate schematic sheets. It displays a hollow box with no internal logic and indicates the schematic sheets that contain the internal logic.

5. To view the design context of an instance in a filtered view, select the instance, right-click, and select Show Context from the popup menu.

   The software displays an unfiltered view of the hierarchical level that contains the selected object, with the instance highlighted. This is useful when you have to go back and forth between different views during analysis. The context differs from the Expand commands, which show connections. To return to the original filtered view, click Back.

Copyright © 2013 Synopsys, Inc.

Filtering Schematics

Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter Schematic command to isolate selected objects. See Chapter 3 of the *Reference Manual* for details about these commands.

This table lists the advantages of using filtering over flattening:

Filter Schematic Command	Flatten Commands
Loads part of the design; better memory usage	Loads entire design
Combine filtering with Push/Pop mode, and history buttons (Back and Forward) to move freely between hierarchical levels	Must use Unflatten Schematic to return to top level, and flatten the design again to see lower levels. Cannot return to previous view if the previous view is not the top-level view.

1. Select the objects that you want to isolate. For example, you can select two connected objects.

   If you filter a hidden instance, the software does not display its internal hierarchy when you filter the design. The following example illustrates this.

2. Select the Filter Schematic command, using one of these methods:
   - Select Filter Schematic from the HDL Analyst menu or the right-click popup menu.
   - Click the Filter Schematic icon (buffer gate) ( ).

Copyright © 2013 Synopsys, Inc.

– Press F12.

– Press the right mouse button and draw a narrow V-shaped mouse stroke in the schematic window. See Help->Mouse Stroke Tutor for details.

The software filters the design and displays the selected objects in a filtered view. The title bar indicates that it is a filtered view. Hidden instances have an H in the lower left. The view displays other hierarchical instances as hollow boxes with nested internal logic (transparent instances). For descriptions of filtered views and transparent instances, see *Filtered and Unfiltered Schematic Views*, on page 416 and *Transparent and Opaque Display of Hierarchical Instances*, on page 421 in the *Reference Manual*. If the transparent instance does not display internal logic, use one of the alternatives described in *Viewing Design Hierarchy and Context*, on page 590, step 4.

3. If the filtered view does not display the pin names of technology primitives and transparent instances that you want to see, do the following:

   – Select Options->HDL Analyst Options->Text and enable Show Pin Name.

   – To temporarily display a pin name, move the cursor over the pin. The name is displayed as long as the cursor remains over the pin. Alternatively, select a pin. The software displays the pin name until you make another selection. Either of these options can be applied to

individual pins. Use them to view just the pin names you need and keep design clutter to a minimum.

– To see all the hierarchical pins, select the instance, right-click, and select Show All Hier Pins.

You can now analyze the problem, and do operations like the following:

Trace paths, build up logic	See *Expanding Pin and Net Logic*, on page 595 and *Expanding and Viewing Connections*, on page 599
Filter further	Select objects and filter again
Find objects	See *Finding Objects*, on page 568
Flatten, or hide and flatten	See *Flattening Schematic Hierarchy*, on page 600. You can hide transparent or opaque instances.
Crossprobe from filtered view	See *Crossprobing from the RTL/Technology View*, on page 582

4. To return to the previous schematic view, click the Back icon. If you flattened the hierarchy, right-click and select Unflatten Schematic to return to the top-level unflattened view.

For additional information about filtering schematics, see *Filtering Schematics*, on page 593 and *Flattening Schematic Hierarchy*, on page 600.

Expanding Pin and Net Logic

When you are working in a filtered view, you might need to include more logic in your selected set to debug your design. This section describes commands that expand logic fanning out from pins or nets; to expand paths, see *Expanding and Viewing Connections*, on page 599.

Use the Expand commands with the Filter Schematic, Hide Instances, and Flatten commands to isolate just the logic that you want to examine. Filtering isolates logic, flattening removes hierarchy, and hiding instances prevents their internal hierarchy from being expanded. See *Filtering Schematics*, on page 593 and *Flattening Schematic Hierarchy*, on page 600 for details.

1. To expand logic from a pin hierarchically across boundaries, use the following commands.

Copyright © 2013 Synopsys, Inc.

To...	Do this (HDL Analyst->Hierarchical/Popup menu)...
See all cells connected to a pin	Select a pin and select Expand. See *Expanding Filtered Logic Example*, on page 597.
See all cells that are connected to a pin, up to the next register	Select a pin and select Expand to Register/Port. See *Expanding Filtered Logic to Register/ Port Example*, on page 598.
See internal cells connected to a pin	Select a pin and select Expand Inwards. The software filters the schematic and displays the internal cells closest to the port. See *Expanding Inwards Example*, on page 598.

The software expands the logic as specified, working on the current level and below or working up the hierarchy, crossing hierarchical boundaries as needed. Hierarchical levels are shown nested in hollow bounding boxes. The internal hierarchy of hidden instances is not displayed.

For descriptions of the Expand commands, see *HDL Analyst Menu*, on page 304 of the *Reference Manual*.

2. To expand logic from a pin at the current level only, do the following:
   - Select a pin, and go to the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.
   - Select Expand or Expand to Register/Ports. The commands work as described in the previous step, but they do not cross hierarchical boundaries.
3. To expand logic from a net, use the commands shown in the following table.
   - To expand at the current level and below, select the commands from the HDL Analyst->Hierarchical menu or the right-click popup menu.
   - To expand at the current level only, select the commands from the HDL Analyst->Current Level menu or the right-click popup menu->Current Level.

To...	Do this...
Select the driver of a net	Select a net and select Select Net Driver. The result is a filtered view with the net driver selected (*Selecting the Net Driver Example*, on page 599).
Trace the driver, across sheets if needed	Select a net and select Go to Net Driver. The software shows a view that includes the net driver.
Select all instances on a net	Select a net and select Select Net Instances. You see a filtered view of all instances connected to the selected net.

Expanding Filtered Logic Example

Copyright © 2013 Synopsys, Inc.

Expanding Filtered Logic to Register/Port Example

Expanding Inwards Example

Selecting the Net Driver Example

Expanding and Viewing Connections

This section describes commands that expand logic between two or more objects; to expand logic out from a net or pin, see *Expanding Pin and Net Logic*, on page 595. You can also isolate the critical path or use the Timing Analyst to generate a schematic for a path between objects, as described in *Analyzing Timing in Schematic Views*, on page 722.

Use the following path commands with the Filter Schematic and Hide Instances commands to isolate just the logic that you want to examine. The two techniques described here differ: Expand Paths expands connections between selected objects, while Isolate Paths pares down the current view to only display connections to and from the selected instance.

For detailed descriptions of the commands mentioned here, see *Commands That Result in Filtered Schematics*, on page 442 in the *Reference Manual*.

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

Copyright © 2013 Synopsys, Inc.

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

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

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

Flattening Schematic Hierarchy

Flattening removes hierarchy so you can view the logic without hierarchical levels. In most cases, you do not have to flatten your hierarchical schematic to debug and analyze your design, because you can use a combination of

filtering, Push/Pop mode, and expanding to view logic at different levels. However, if you must flatten the design, use the following techniques., which include flattening, dissolving, and hiding instances.

1. To flatten an entire design down to logic cells, use one of the following commands:
   - For an RTL view, select HDL Analyst->RTL->Flattened View. This flattens the design to generic logic cells.
   - For a Technology view, select Flattened View or Flattened to Gates View from the HDL Analyst->Technology menu. Use the former command to flatten the design to the technology primitive level, and the latter command to flatten it further to the equivalent Boolean logic.

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

   Unless you really require the entire design to be flattened, use Push/Pop mode and the filtering commands (*Filtering Schematics*, on page 593) to view the hierarchy. Alternatively, you can use one of the selective flattening techniques described in subsequent steps.

2. To selectively flatten transparent instances when you analyze critical paths or use the Expand commands, select Flatten Current Schematic from the HDL Analyst menu, or select Flatten Schematic from the right-click popup menu.

   The software generates a new view of the current schematic in the same window, with all transparent instances at the current level and below flattened. RTL schematics are flattened down to generic logic cells and Technology views down to technology primitives. To control the number of hierarchical levels that are flattened, use the Dissolve Instances command described in step 4.

   If your view only contains hidden hierarchical instances or pale yellow (opaque) hierarchical instances, nothing is flattened. If you flatten an unfiltered (usually the top-level design) view, the software flattens all hierarchical instances (transparent and opaque) at the current level and below. The following figure shows flattened transparent instances.

Copyright © 2013 Synopsys, Inc.

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

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

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

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

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

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

4. To selectively flatten some hierarchical instances in your design by dissolving them, do the following:
   - If you want to flatten more than one level, select Options->HDL Analyst Options and change the value of Dissolve Levels. If you want to flatten just one level, leave the default setting.
   - Select the instances to be flattened.
   - Right-click and select Dissolve Instances.

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

Starting View	Software Generates a...
Filtered	Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of *Viewing Design Hierarchy and Context*, on page 590. Use the Back button to return to the undissolved view.
Unfiltered	New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. Select Unflatten Schematic to return to the top-level unflattened view. You cannot use the Back button to return to previous views because this is a new view.

The following figure illustrates this.

Copyright © 2013 Synopsys, Inc.

Dissolved logic for prgmcntr shown nested when started from filtered view

Dissolved logic for prgmcntr shown flattened in context when you start from an unfiltered view

Use this technique if you only want to flatten part of your design while retaining the hierarchical context. If you want to flatten most of the design, use the technique described in the previous step. Instead of dissolving instances, you can use a combination of the filtering commands and Push/Pop mode.

Minimizing Memory Usage While Analyzing Designs

When working with large hierarchical designs, use the following techniques to use memory resources efficiently.

- Before you do any analysis operations such as searching, flattening, expanding, or pushing/popping, hide (HDL Analyst->Hide Instances) the hierarchical instances you do not need. This saves memory resources, because the software does not load the hierarchy of the hidden instances.
- Temporarily divide your design into smaller working files. Before you do any analysis, hide the instances you do not need. Save the design. The `srs` and `srm` files generated are smaller because the software does not save the hidden hierarchy. Close any open HDL Analyst windows to free all memory from the large design. In the Implementation Results view, double-click one of the smaller files to open the RTL or Technology schematic. Analyze the design using the smaller, working schematics.
- Filter your design instead of flattening it. If you must flatten your design, hide the instances whose hierarchy you do not need before flattening, or use the Dissolve Instances command. See *Flattening Schematic Hierarchy, on page 600* for details. For more information on the Expand Paths and Isolate Paths commands, see *RTL View and Technology View Popup Menu Commands, on page 402* of the *Reference Manual*.
- When searching your design, search by instance rather than by net. Searching by net loads the entire design, which uses memory.
- Limit the scope of a search by hiding instances you do not need to analyze. You can limit the scope further by filtering the schematic in addition to hiding the instances you do not want to search.

Copyright © 2013 Synopsys, Inc.

Using the FSM Viewer

The FSM Viewer option is available only in the Synplify Pro and Synplify Premier tools. The FSM viewer displays state transition bubble diagrams for FSMs in the design, along with additional information about the FSM. You can use this viewer to view state machines implemented by either the FSM Compiler or the FSM Explorer. For more information, see *Running the FSM Compiler*, on page 455 and *Running the FSM Explorer*, on page 459, respectively.

1. To start the FSM viewer, open the RTL view and either
   - Select the FSM instance, click the right mouse button and select View FSM from the popup menu.
   - Push down into the FSM instance (Push/Pop icon).

   The FSM viewer opens. The viewer consists of a transition bubble diagram and a table for the encodings and transitions. If you used Verilog to define the FSMs, the viewer displays binary values for the state machines if you defined them with the ‘define keyword, and actual names if you used the parameter keyword.

2. The following table summarizes basic viewing operations.

To view...	Do...
from and to states, and conditions for each transition	Click the Transitions tab at the bottom of the table.
the correspondence between the states and the FSM registers in the RTL view	Click the RTL Encoding tab.
the correspondence between the states and the registers in the Technology View	Click the Mapped Encodings tab (available after synthesis).
only the transition diagram without the table	Select View->FSM table or click the FSM Table icon. You might have to scroll to the right to see it.

This figure shows you the mapping information for a state machine. The Transitions tab shows you simple equations for conditions for each state. The RTL Encodings tab has a State column that shows the state names in the source code, and a Registers column for the corresponding RTL encoding. The Mapped Encoding tab shows the state names in the code mapped to actual values.

Copyright © 2013 Synopsys, Inc.

	From State	To State	Condition
1	LapDisplay	LapDisplay	Laplevel&!Reset
2	Zero	LapDisplay	Lap&!Start&!Reset
3	StopSecond	StopSecond	!Lap&!Start&!Reset
4	StopFirst	StopSecond	Lap&!Start&!Reset
5	CountCont	StopSecond	Start&!Reset
6	StopSecond	StopDiff	Lap&!Start&!Reset
7	StopDiff	StopDiff	!Lap&!Start&!Reset
8	StopFirst	StopFirst	!Lap&!Start&!Reset
9	CountFirst	StopFirst	Start&!Reset
10	StopSecond	CountCont	Start&!Reset

States and Conditions

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

Mapped Encoding

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

RTL Encoding

3. To view just one selected state,
   - Select the state by clicking on its bubble. The state is highlighted.
   - Click the right mouse button and select the filtering criteria from the popup menu: output, input, or any transition.

   The transition diagram now shows only the filtered states you set. The following figure shows filtered views for output and input transitions for one state.

Similarly, you can check the relationship between two or more states by selecting the states, filtering them, and checking their properties.

4. To view the properties for a state,
   - Select the state.
   - Click the right mouse button and select Properties from the popup menu. A form shows you the properties for that state.

   To view the properties for the entire state machine like encoding style, number of states, and total number of transitions between states, deselect any selected states, click the right mouse button outside the diagram area, and select Properties from the popup menu.

5. To view the FSM description in text format, select the state machine in the RTL view and View FSM Info File from the right mouse popup. This is an example of the FSM Info File, *statemachine*.info.

```
State Machine: work.Control(verilog)-cur_state[6:0]
No selected encoding - Synplify will choose
Number of states: 7
Number of inputs: 4
Inputs:
   0: Laplevel
   1: Lap
   2: Start
   3: Reset
   Clock: Clk
```


Copyright © 2013 Synopsys, Inc.

```
Transitions: (input, start state, destination state)
   -100 S0 S6
   --10 S0 S2
   ---1 S0 S0
   -00- S0 S0
   --10 S1 S3
   -100 S1 S2
   -000 S1 S1
   ---1 S1 S0
   --10 S2 S5
   -000 S2 S2
   -100 S2 S1
   ---1 S2 S0
   -100 S3 S5
   -000 S3 S3
   --10 S3 S1
   ---1 S3 S0
   -000 S4 S4
   --1- S4 S0
   -1-- S4 S0
   ---1 S4 S0
   -000 S5 S5
   -100 S5 S4
   --10 S5 S2
   ---1 S5 S0
   1--0 S6 S6
   ---1 S6 S0
   0--- S6 S0
```

CHAPTER 12

Floorplanning with Design Planner

The Synplify Premier Design Planner tool lets you create a design plan to physically constrain portions of a design to specific regions on a device. It is important to place physical constraints carefully, and this tool helps you do this. The topics below describe how to use the tool.

Netlist restructure files usually contain primitives that have been bit sliced. The design plan (`sfp`) and netlist restructure files are used during optimization to improve the overall design performance.

The following describe the Design Planner tool and bit slicing in more detail:

- Using Design Planner, on page 612
- Assigning Pins and Clocks, on page 619
- Working with Regions, on page 629
- Working with Altera Regions, on page 641
- Working with Xilinx SSI Devices, on page 645
- Working with Xilinx Regions, on page 650
- Using Process-Level Hierarchy, on page 652
- Bit Slicing, on page 652

Copyright © 2013 Synopsys, Inc.

Using Design Planner

The Design Planner functionality is only available for certain Altera and Xilinx technologies. You can use the Design Planner tool either in combination with graph-based physical synthesis (*Altera Graph-Based Physical Synthesis with Design Planner*, on page 49) or in a design plan-based physical synthesis flow (*Design Plan-based Physical Synthesis*, on page 51). You can also use the design plan-based logic synthesis with Physical Plus (*Xilinx Physical Plus*, on page 64). The following describes the basics of using Design Planner:

- Starting Design Planner, on page 612
- Copying Objects in the Design Planner Tool, on page 614
- Controlling Pin Display in the Design Plan Editor, on page 615
- Creating and Using a Design Plan File for Physical Synthesis, on page 618

Starting Design Planner

After the design is compiled, you create a design plan by doing the following:

1. Start with a compiled design.

   It is best if you run logic synthesis first to ensure that there are no errors before you start physical synthesis, but you can run Design Planner on a design that has just been compiled but not synthesized.

2. Click the New Design Plan icon ( ) in the Project view. Alternatively, you can also create a new design plan file using File->New from the Project menu.

3. If you have not run area estimation, or the area estimation file is out-of-date, the Estimation Needed dialog box appears asking if you want to run estimation.
   - If you do not see this box, the No area estimate warning check box on the Assignments tab of Tools->Design Planner Preferences is disabled.
   - If you click No, the Design Planner is displayed.
   - If you click Yes, the tool first runs area estimation, and the Running Estimation dialog opens and displays the runtime of the job. Once estimation is complete, the Design Planner opens.

Copyright © 2013 Synopsys, Inc.

The following figure shows the Design Planner and RTL views.

Design Plan Views

Copying Objects in the Design Planner Tool

You can use the cut, copy, and paste functions in the Design Plan Editor and Design Plan Hierarchy Browser views instead of drag and drop. Note the following caveats:

- You can only use cut, copy, and paste on assignments (modules, primitives, and nets).
- You cannot cut or copy regions using the Design Planner tool and, you cannot paste to multiple regions.

The following table summarizes the cut and paste operations.

To...	Do this...
Assign a module or primitive from a HDL Analyst view	• Select the module/primitive in HDL Analyst and press Ctrl-c to copy it. • Select the destination region in Design Planner and paste with Ctrl-v.
Assign a net to an I/O block from HDL Analyst	• Select the net in HDL Analyst and copy. • Select the I/O block region and paste it.
Assign a module or primitive from the Hierarchy Browser Unassigned Bin	• Select the module or primitive in the hierarchy browser and copy it. • Select the destination region and paste it.
Replicate a module or primitive using the Hierarchy Browser	• Select the module or primitive within the region using the hierarchy browser and copy. • Select the destination region and paste. This displays the Replication dialog box.
Move an assignment using the Hierarchy Browser view	• Select the module or primitive in the region using the Hierarchy Browser, and cut it using Ctrl-x. • Select the destination region and paste.

Controlling Pin Display in the Design Plan Editor

The Design Plan Editor contains the device floorplan and region view in the Design Planner. It lets you view and assign external ports or internal nets to I/O pins on the device.

1. To expand the pin view, do the following:
   – Open the Design Planner and toggle on View->Expanded Pin View, or use Ctrl-e. This enables the expanded pin view in the Design Plan Editor. The following figure shows the enabled and disabled views for a design.

Copyright © 2013 Synopsys, Inc.

2. To adjust the size of the pins in the view, do the following:
   - Select View->Adjust Pin View... the Adjust Pin View dialog box appears.

   - Adjust the view by moving the slider to either a smaller or larger view of the pins.
   - Click OK to save your new pin view setting or Cancel to restore your original pin view setting.

3. To display the device I/O pin names, in different views, see the table below:

To ...	Do this...
List the pin names in the Design Plan Hierarchy Browser	Select the expand icon next to Pins.
List the pin names in the Design Plan view	Click the design name in the Design Plan Hierarchy Browser. This lists the design objects in the Design Plan view. Double-click the Pins folder in the Design Plan view.
View information about the pins in the Design Plan view	Right-click and select Show/Hide columns, then select the columns you need in the dialog box: Clock, Name, Side, Seq, Dir, or Port/Net.
Display the pin number	Place your cursor over the pin in the Design Plan Editor.

When you select a pin in the Design Plan Hierarchy Browser, the corresponding pin location is highlighted in the other views. The following figure shows an example of I/O pins displayed in all three views of the Design Planner.

Copyright © 2013 Synopsys, Inc.

Creating and Using a Design Plan File for Physical Synthesis

To create a design plan file, you must have the Design Planner option. Even if you have this option, you do not need to use a design plan file for graph-based synthesis. However, for older Altera technologies, the design plan file is required to run physical synthesis. The following procedure shows you how to generate a design plan file

1. Use the Design Plan editor to interactively assign RTL modules, paths, or components to regions on the device.

   For information about working with regions and assignment of logic, see *Assigning Pins and Clocks*, on page 619 and *Working with Regions*, on page 629.

   For additional, technology-specific information on assigning logic to regions, see *Working with Altera Regions*, on page 641 and *Working with Xilinx Regions*, on page 650.

   When you have finished assigning the logic, the tool generates an `sfp` physical constraint file.

2. When you create an RTL region, select Block Region Tool and then configure the region. See *Creating Regions*, on page 629 for details.

3. Use the design plan file for physical synthesis.
   - Add the file to the project.
   - Go to Implementation Options ->Design Planning and enable the file.
   - Run physical synthesis.

   The physical synthesis tool uses the placement information in the design plan file as physical constraints for synthesis.

Copyright © 2013 Synopsys, Inc.

Assigning Pins and Clocks

This section discusses the following general guidelines for displaying and assigning pin assignments for design planning:

- Assigning Pins Interactively, on page 619
- Assigning Clock Pins, on page 622
- Modifying Pin Assignments, on page 623
- Using Temporary Pin Assignments, on page 624
- Viewing Assigned Pins in Different Views, on page 626
- Viewing Pin Assignment Information, on page 627

Assigning Pins Interactively

You can manually assign pins using the methods described here. You can either assign the pins in the SCOPE window or use Design Planner to assign pins.

1. To assign pins directly in the SCOPE window, do the following:
   - Open the SCOPE Attributes tab.
   - Select a port. Assign it to a pin location using a pin location constraint appropriate to your technology or the syn_loc constraint.
   - Alternatively, manually add constraints to the constraint file.
2. To assign a pin in Design Planner, do the following:
   - Open the Design Plan window and make sure you can see the pins clearly. See *Viewing Pin Assignment Information*, on page 627 for information on displaying the pins.
   - Select a pin in the Design Plan RTL view or the Hierarchy Browser for that RTL view.
   - Drag the pin to the location you want in the Design Plan. You can drag it to the appropriate pin in the graphic Design Plan editor view, or to the appropriate pin name in the Design Plan Hierarchy Browser or Design Plan view.

Copyright © 2013 Synopsys, Inc.

The design plan views reflect the new status of the pin. For details, see *Viewing Assigned Pins in Different Views*, on page 626.

This example shows a pin assignment:

3. To assign a bus port (group of signals), drag a bus port from the RTL view and drop it to one device pin in the Design Plan Editor view.

   The software allocates the remaining pin(s) depending upon its location on the device. Pins located on the left and right sides of the device are allocated from bottom to top. Pins located on the top and bottom of the device are allocated from left to right. Pins that are occupied are skipped. All devices allocate pins using this convention.

   For information about viewing pin assignments and related information, see *Viewing Assigned Pins in Different Views*, on page 626.

   The following figure shows bus port assignment:

Copyright © 2013 Synopsys, Inc.

Assigning Clock Pins

Clock pins are displayed in green in the Design Plan Editor to distinguish them from the signal I/O pins.

There are several methods of assigning I/O pins, so you might encounter pin assignment conflicts like the following:

- The constraint file might contain I/O pin locations that conflict with the pin locations specified in the `sfp` file. The SCOPE constraint file typically takes precedence over the Design Plan file (`sfp`) when conflicts exist after pin assignments.
- If pin assignments from the back end place-and-route tool are added into the `sfp` file, potential pin lock conflicts may occur. In case of a conflict, the tool generates an appropriate warning message.
- Design rule checks are implemented if there are multiple assignments to the same I/O pins or ports.

The following procedure shows you how to assign clock pins.

1. To assign a clock pin, drag and drop a signal to a clock pin, as described in *Assigning Pins Interactively*, on page 619.

   A message asks you to confirm the assignment to ensure that the correct signal gets assigned to the clock pin. After assignment, the pin changes to pink.

   You cannot drag and drop a bus (group of signals) to a clock pin. If you drag and drop a bus to an I/O pin near a clock pin, the tool skips the clock pin when it assigns the bus to the I/O pins.

2. To view the information in the Design Plan view, enable the Clock column on the Select Columns dialog box. This displays whether or not a pin is a clock (Yes or No).

Modifying Pin Assignments

The following shows you how to change pin assignments once they have been made.

1. To undo an assignment, click on the pin in any of the three views, right-click, and select Delete Pin Assignment.

2. To reassign a port, drag and drop it at a new location in the Design Plan editor.

3. To change the current order of pin assignments from clockwise to counter-clockwise or vice versa, do the following:

   – Select a set of pins in any view of the Synplify Premier Design Planner.

Copyright © 2013 Synopsys, Inc.

- Right-click and select Reverse Pin Assignments from the pop-up menu. The reversed pin assignments are displayed in the Design Planner views.

4. To rearrange or reorder pin assignments for nets or ports, do the following:
   - Move the pins to temporary assignments. See *Using Temporary Pin Assignments*, on page 624 for details.
   - Assign them to the desired locations.

Using Temporary Pin Assignments

Use temporary assignments to rearrange or reorder pin assignments for nets or ports

1. To create a temporary assignment, drag and drop an assigned pin from the Design Plan editor to the Temporary Assigns icon ( ) in the Hierarchy Browser.

   The Temporary Assigns container lists the pins with temporary assignments. Note that you *cannot* drag and drop assignments from the HDL Analyst RTL view to Temporary Assigns.

2. To re-assign a pin with a temporary assignment, do either of the following:
   - For assignment to a new location, drag and drop the pin from the Temporary Assigns container to the new pin or region location in the Design Plan Editor. You can also reassign the pin using the methods described in *Assigning Pins Interactively*, on page 619.
   - To return the pin to its original placement location, select the assignment in the Temporary Assigns. Then, right-click and select Reassign from the pop-up menu.

   As pins in the Temporary Assigns container are reassigned, they are automatically removed from Temporary Assigns.

3. To remove assignments from all the pins, select the Temporary Assigns icon, right-click, and select Empty.

Copyright © 2013 Synopsys, Inc.

4. To sort pin assignments by description, name, or origin in the Design Plan View, do the following:
   - Display the appropriate column by right-clicking and selecting Show Columns->Description/Origin from the popup menu.
   - Click on the column heading in the Design Plan View to sort.
5. To undo or redo operations in the Temporary Assigns container, use the Edit->Undo or Edit->Redo commands.

The following figure shows a temporary assignment:

Copyright © 2013 Synopsys, Inc.

Viewing Assigned Pins in Different Views

The following table summarizes how pin assignments are displayed in the Design Plan views:

Design Plan Hierarchy Browser	Assigned pins include assignment information. Selected assigned pins are red.
Design Plan view	To view information for the pins, select Show/Hide columns from the popup menu and choose the kinds of information you want to display for the pins, like pin direction and port or net information.

Name	Port/Net	Side	Seq	Dir	Clock
pin_A3		right	42	Not assigned	No
pin_A5		right	34	Not assigned	No
pin_A6		right	33	Not assigned	No
pin_A7		top	14	Not assigned	No
pin_A8		top	10	Not assigned	No
pin_A10		top	2	Not assigned	No
pin_A11		top	1	Not assigned	No
pin_A13		top	9	Not assigned	No

Design Plan editor	Orange: Selected assigned pins Red: Unselected assigned pins Blue: Selected unassigned pins Green: Unassigned clock pins Pink: Unselected assigned clock pins To view the pin number and assignment for a pin, place the cursor over the pin.
RTL view	Assigned ports are displayed in blue. Place your cursor over a pin to display information about it.

Viewing Pin Assignment Information

This section lists different methods you can use to obtain information about your pin assignments. These methods are in addition to the display information described in *Viewing Pin Assignment Information*, on page 627.

1. To view information for a particular pin, use the following:
   - Tooltips
     To display a tooltip, move your cursor over a pin or port in the Design Plan editor or in the RTL view.
   - The visual clues described in *Viewing Pin Assignment Information*, on page 627.
2. To display connectivity between the I/O pads and the assigned logic for regions on the device, do the following:
   - To view connectivity for all regions, right-click in the Design Plan Editor and select Rats Nest->Show from the pop-up menu.
   - To view connectivity for one region, select it and right-click in the Design Plan Editor. Select Rats Nest->Show Selected from the menu.
   - To disable the connectivity display, right-click in the Design Plan Editor and select Rats Nest->Hide.

   Alternatively, you can also select View->Rats Nest from the Project menu, then choose the Show, Hide, or Show Selected command.

   The display shows lines (rat's nesting) to indicate the connectivity.
3. To view pin assignment statistics for the design, right-click on the Pins folder in the Design Plan Hierarchy Browser, and select Properties from the pop-up menu.

   The Properties dialog box shows the total number of pins, the number of assigned pins, and the percentage of pins assigned.

Copyright © 2013 Synopsys, Inc.

4. Use crossprobing.

   When you select assigned ports in any of the Design Planner views or the HDL Analyst RTL view, the corresponding pins are highlighted in the other views view. Similarly, if you select a net that has an assigned pin in the RTL view, the corresponding pin is highlighted in the Design Planner views. If you select the assigned pin in a Design Planner view, the corresponding internal net is highlighted in the RTL view.

Working with Regions

This section discusses the following general guidelines for placing and editing regions in the Design Planner before running physical synthesis:

- Creating Regions, on page 629
- Using Region Tunneling, on page 631
- Viewing Intellectual Property (IP) Core Areas, on page 634
- Assigning Logic to Top-level Chip Regions, on page 634
- Assigning Logic to Regions, on page 638
- Replicating Logic Manually, on page 638
- Checking Utilization, on page 639

Creating Regions

Region placement depends on the data flow and pin locations in your design. The following procedure shows you how to create a region.

1. If needed, select View->Expanded Pin View and adjust the view to display the device with or without I/O pins.
2. To create a region, right-click in the Design Plan Editor and select Block Region Tool to begin the region drawing process.

   For more information about creating technology-specific regions, refer to the following table depending on the technology you have selected.

For...	See
Altera designs	Creating Design Planner Regions for Altera Designs, on page 642
Xilinx designs	Working with Xilinx SSI Devices, on page 645 and Working with Xilinx Regions, on page 650.

3. Position the cursor where you want to create the region and then drag the cursor diagonally to create a rectangular area for the region.

Copyright © 2013 Synopsys, Inc.

– Do not create regions that overlap or are contained within an area reserved for IP (see *Viewing Intellectual Property (IP) Core Areas*, on page 634).

See the following vendor-specific guidelines to determine where to place the regions.

Altera	
Critical path placement	Run the target place-and-route tool with no constraints to obtain the placement of the critical path. Use the Design Plan Editor to create a region in this area. This is a good starting point to determine what row to begin with when placing the critical path on the logic device using the Synplify Premier Design Planner tool.
Overlapping regions	You can overlap regions to optimize placement. However, be aware that that the Synplify Premier Design Planner software treats overlapping regions no differently than regions that do not overlap.
Xilinx	
Region size	The size and location of Xilinx regions can be easily modified, so a rough estimate is usually sufficient.
Critical path placement	You can get a good starting point for region placement from the Xilinx floorplanner. Run placement and routing without constraints, then use the floorplanner to determine where the critical path logic is placed. Use this information to create a region in the same general area on the logic device using the Design Planner tool.
Overlapping regions	The Synplify Premier Design Planner software supports overlapping regions, but the Xilinx place-and-route tool cannot always place these designs. Overlapped regions can potentially create an error.

4. You can configure the region to apply selected tunneling modes. See *Using Region Tunneling*, on page 631 for ways to configure regions.

5. Then, assign logic to the region. For more information, see *Assigning Logic to Regions*, on page 638.

Using Region Tunneling

The Synplify Premier software can apply tunneling optimizations to region assignments. To do this:

1. Highlight a region in the Design Planner, then right-click and select Region Type.
2. You can configure the region by selecting one of the following modes shown in the table below. Some options are vendor-specific.

Set the option to...	To...
Soft (Tunneling On) – (Xilinx)	Allows components to be moved across the region boundaries in both directions. This is the default.
Hard (Tunneling Off)	Ensures that components are not moved out of the region, but allow other objects to be moved into the region.
Keep-out (Xilinx)	Ensures that no placement occurs in the region. Use this option to create decongestion areas for optimizing your design.
IP Block (Altera)	Ensures that the region only contains the IP block. Use this for encrypted IP, to ensure that nothing except for the IP logic is placed in this region.

   Note:

   – It is possible to highlight multiple regions and then select a tunneling option for those regions simultaneously.

   – For Altera designs, you cannot forward-annotate soft regions. However, hard region constraints can be forward-annotated.

3. To view tunneling status for the region, do the following:

   – To view tunneling status for a region, highlight the region, then right-click and select the Properties option. From this dialog box, the tunneling status for the region is displayed.

   – To display the tunneling status for all the regions, go to the Design Plan view, right-click and select Show/Hide columns. From the Select Columns dialog box, check the box next to Tunneling. The view displays a column with the tunneling status for all the regions.

Copyright © 2013 Synopsys, Inc.

The Design Planner can display how tunneling is implemented. Also, the status of the region is saved and written out to the Synplify Premier physical constraint file (`sfp`).

Moving and Sizing Regions

You can move and resize regions using the cursor arrow keys or the mouse button. The following procedure provides details.

1. To move a regions use the arrow keys or the mouse button.
   - Select the region.
   - To use the arrow keys:, use the left, right, up, or down keys to reposition the region.
   - To use the mouse button, press the left mouse button while dragging the region to the desired position on the device.

     The tool displays WYSIWYG region boundaries that show you exactly what you are doing when you move or resize the boundaries.
   - To preserve the logic and memory resources of a region when it is moved, hold down the Shift key when you move it.

     The tool preserves the logic and memory resources when you move a region. For example, Xilinx devices can preserve the number of CLBs and BRAMs and Altera devices can preserve the number of LABs and ESBs in a region.
2. To resize a region, use the arrow keys or the mouse button.
   - Select the region.
   - To use the arrow keys:, press and hold the Ctrl and Shift keys simultaneously. An initial resizing arrow appears along the edge of the region. Continue to hold down Ctrl and Shift while pressing the appropriate arrow keys (left, right, up, or down) to resize the region in the direction you want. Release the Shift key. You can no longer resize the region.
   - To resize a region with the mouse button, press the left mouse button on any of the handles of the rectangle while dragging the region in the direction you want the region resized.

   The tool does not preserve logic and memory resources when you resize a region.

Copyright © 2013 Synopsys, Inc.

Viewing Intellectual Property (IP) Core Areas

Dedicated areas on the device reserved for IP cores appear as gray boxes in the Design Plan Editor. A device can contain up to can contain up to four IP core areas depending on the part specified for the device.

- To view information about the IP core, move your cursor over the gray box to display a tooltip with information. Do *not* create regions that overlap or are contained within an IP core area.

Assigning Logic to Top-level Chip Regions

You can specify the top-level device as a region and then assign logic to this chip region. To do this:

1. Open the Design Plan view.

   Notice the Chip Region hierarchy under the device part and package designation in the Design Plan Hierarchy view.

Design Plan Hierarchy View

2. Assign logic to the chip region. To do this, you can:
   - Highlight logic in the RTL view, then right-click and select Assign to->Chip from the popup menu.

Copyright © 2013 Synopsys, Inc.

– Highlight logic to assign from Logic in the Design Plan Hierarchy view, then right-click and select Assign to->Chip from the popup menu.

– Otherwise, simply drag-and-drop highlighted logic from the Logic to Chip hierarchy tree in the Design Plan Hierarchy view.

Design Plan Hierarchy View

The chip assignments are reflected in the Design Plan Hierarchy view as shown in the following figure.

Copyright © 2013 Synopsys, Inc.

Assigning Logic to Regions

1. To assign individual instances to regions, make sure that Edit Regions is enabled and do either of the following:
   - Drag and drop the logic into the region.
   - Select the instance in an HDL Analyst, Design Plan Hierarchy Browser, or Design Plan view. Right-click and select Assign to-> *regionName*. The regions are listed in order of recent use.

   For technology-specific tips about assigning logic to regions, see the following:
   - *Assigning Logic to Altera Design Planner Regions*, on page 643
2. To assign a critical path to a region, do the following:
   - Select the critical path, filtering it if necessary. You can do this from the log file.
   - Drag the selected critical path from the RTL view into a region in Design Planner.

   By assigning the critical path instances to the same region, you can optimize the timing.
3. For critical paths from pin-locked I/Os, assign the critical path to a region that is close to the pin positions.

   Physically constraining logic close the to locked pins minimizes routing delays.

For information about device utilization, see *Checking Utilization*, on page 639.

Replicating Logic Manually

When the fanout from an instance fans goes to instances in several other instances, you might want to replicate the instance to avoid the routing delay between the regions. You can use the methods described here.

1. Replicate logic by copying and pasting.
   - Copy (`Ctrl-c`) the logic to be replicated from the original region.
   - Paste (`Ctrl-v`) the replicated logic in the region where it is required.

Each region now contains a local copy of the instance. If you replicate an in a region where the instance does not drive any logic, the tool does not create a copy of the instance in that region. Therefore, when you look at the RTL netlist of the region, the replica of the instance does not appear.

2. Assign the same instance logic from the HDL Analyst RTL view to different regions.

   The Instance Replication dialog box opens. Confirm whether or not you want to replicate the selected logic instance in the specified region.

Checking Utilization

Use the following tips and guidelines for device and region utilization when assigning logic to regions.

1. To view device utilization, select Run->Estimate Area.

   Utilization is reported in the log file.

2. To estimate region utilization, do the following:
   - To estimate utilization for all the regions, right-click in the Design Plan Editor view, right-click and select Estimate All Regions.
   - To estimate region utilization for an individual region, right-click with a selected region and select Estimate Regions.

   Estimates are in terms of instances. As the job runs, the region is greyed-out and a label in the upper-left corner of the region displays the elapsed time of the estimation job. The label remains until estimation is complete. Est Pending appears in the upper-left corner of all regions waiting for region estimation.

   Regions displayed in red have a utilization higher than 80 percent, See step 4 for more information about utilization.

Copyright © 2013 Synopsys, Inc.

3. To view region utilization information, use one of these methods:
   - For utilization information for the whole design, view the log file.
   - For utilization information about the current estimation run, view the status in the Tcl Script window. or select Run->Job Status immediately after an estimation run.
   - To view utilization information for a selected region, right-click and select Properties. You can also view the tooltip information.
   - To display utilization information for regions in the Design Plan view, click Regions in the Design Plan Hierarchy Browser. This updates the Design Plan view with statistics for the regions.
   - To determine which statistics to display in the Design Plan view, right-click in this view and select Show/Hide Columns. Select the options for utilization you want to display. The available options can vary with the technology.
4. Follow these guidelines for device and region utilization.
   - Keep device utilization below 90 percent. Higher utilization rates can lead to problems with timing closure.
   - If device utilization is over 90 percent, and if the design contains several finite state machines, try using the sequential encoding style, (instead of one-hot) to free up more space on the device.
   - Keep region utilization below 80 percent to allows for Synplify Premier Design Planner area estimations and for additional area required for routing and replicating. The place-and-route tools consider the design plan to be a hard constraint, so if there is not enough area in the region for routing, the place-and-route tool will error out.
5. If utilization exceeds the guidelines, resize the region to ensure that it is not over utilized.

Copyright © 2013 Synopsys, Inc.

Working with Altera Regions

The guidelines in this section provide tips and strategies for using the Synplify Premier Design Planner for design planning with Altera Stratix and Cyclone devices. It supports the following technologies:

- Stratix families
  (Stratix V, Stratix IV, Stratix III, Stratix II GX, Stratix II, Stratix GX, and Stratix)
- Cyclone families
  (Cyclone II and Cyclone)

You can use the Design Planner to view the Altera devices, then create regions and assign critical path logic to them. The Stratix and Cyclone family of devices use a row and column coordinate system, with the origin (1,1) located at the lower-left corner of the device. All components align with row and column boundaries. The device features can include LAB sites (logic blocks), RAM (for example, M20K, MRAM, M144K, or M9K), and DPS locations. Depending on the device and part and package used, the number of these blocks on the device may vary.

Copyright © 2013 Synopsys, Inc.

The following shows an Altera Stratix V device in the Design Plan Editor.

This section describes the following:

- Creating Design Planner Regions for Altera Designs, on page 642
- Assigning Logic to Altera Design Planner Regions, on page 643

Creating Design Planner Regions for Altera Designs

This section contains Stratix- and Cyclone-specific information about creating regions. For information about how to create a region, see *Creating Regions*, on page 629.

- For Stratix V devices, FF PLL and the voided spine region are displayed in the Design Planner. These are keep out locations on the device, so when you create regions:

  - They can overlap with the FF PLL site or voided spine region. However, logic assigned to these regions cannot use these resources.
  - You cannot create a region that is completely contained in the boundaries of the FF PLL or voided spine region.

- You can create a region around any number of LAB, RAM, and DSP structures on the device. You can create regions that contain only LABs, only RAMs, only DSPs, or regions that include any combination of LABs, RAMs, and DSPs, as required. However, you cannot create a region that is completely contained within the boundaries of one of these blocks.
- Regions can be moved or resized.
- When you create, move, or resize regions, they snap to the row/column grid on the device.

Assigning Logic to Altera Design Planner Regions

This section contains tips and guidelines for mapping MACs, RAMs, and ROMs to regions.

Mapping MACs

1. Enable the Create MAC Hierarchy optimization on the GCC & Prototyping Tools tab of the Implementation Options dialog box.

   For Stratix devices, this option is enabled by default. When enabled, this option maps MAC configurations together into one MAC block so that this block can be easily assigned to DSP regions for physical synthesis.

2. Follow these guidelines when assigning MACs:
   - Place MAC blocks in a region containing DSP resources. If you do not do this, the MAC block is mapped to logic and a warning message is generated in the log file (`srr`). You can display DSP resources after you estimate utilization. See *Checking Utilization*, on page 639 for a procedure.
   - Do not place signed and unsigned multipliers in the same DSP block.

3. If you are using the syn_multstyle attribute, note the following:
   - Do not set the attribute value to logic for a MAC. If the attribute is set to logic, the tool maps the MAC to logic and generates a warning message in the log file (`srr`).

Copyright © 2013 Synopsys, Inc.

- Do not place a multiplier with a syn_multstyle=lpm_mult attribute in a region without DSP resources. If you do, the tool maps the multiplier to the MAC block and generates a warning message in the log file.

Mapping RAMs and ROMs

1. Place RAM/ROM logic in a region containing RAM/ROM resources.

   If you do not, the tool maps the RAM/ROM to logic and generates a warning message in the log file (`srr`). You can display RAM and ROM resources after you estimate utilization. See *Checking Utilization*, on page 639 for a procedure.

2. Ensure that the register driving the address or the output register is assigned to the same region.

   If not, the RAM will not be inferred.

3. If you are using the syn_ramstyle attribute, note the following:
   - Do not set syn_multstyle=logic, and then assign the RAM/ROM logic to a region with RAM/ROM resources. The tool maps the ROM/ROM instance to logic, and generates a warning message in the log file (`srr`).
   - Do not assign RAM/ROM logic with an attached syn_ramstyle=blockram attribute to a region without RAM/ROM resources. If you do so, the tool maps the RAM/ROM to altsyncram and generates a warning message in the log file.

Copyright © 2013 Synopsys, Inc.

Working with Xilinx SSI Devices

Xilinx Stacked Silicon Interconnect (SSI) technology allows multiple die to be combined in a single chip. Specific Xilinx Virtex-7 devices contain multiple dies (for example, 2000T, 1500T, 1140XT, 870HT, and 580HT). Each die is called a Super Logic Region (SLR) that is connected together between the adjacent SLRs. Devices can contain 2, 3, or 4 SLRs.

The Design Planner supports SLR assignments to pre-defined regions. From the Design Planner you can choose an instance and assign it to an SLR. These results are written to the design floorplan file (`sfp`).

Assigning Logic to SLR Regions

To assign logic to SLR regions, do the following:

1. Click on the New Design Plan ( ) icon to bring up the Design Planner tool.

Copyright © 2013 Synopsys, Inc.

Depending on the number of SLRs for the device, SLR Regions appear in the Design Plan Tree View as pre-defined regions. For example, they are named SLR0, SLR1, and SLR2. The bottom-most SLR is 0 and the top-most SLR is number 3.

2. You can assign logic to an SLR region from the HDL Analyst View in the following ways:
   - Select an instance and drag-and-drop it to the SLR Regions bin in the Design Plan Tree View over the specified SLR.
   - Select an instance and drag-and-drop it near the edge of an SLR region in the Design Plan Editor View.

     Notice that the SLR region turns the color blue prior to dropping the instance.

Drop instance to Design Plan Tree View

Drop instance to Design Plan Editor View

   - Select an instance, right-click and select the Assign to menu; then choose from the list of SLR regions (for example, SLR0, SLR1, or SLR2).

OR

3. You can assign logic to an SLR region from the Design Planner View in the following ways:
   - Select an instance from the Logic bin and drop it to an SLR region over the SLR Region bin in the Design Plan Tree View.

     Note: You can assign an instance from the Unassigned Bin, Logic bin, Temporary Assigns bin, or from an assignment in another SLR region.
   - Select an instance from the Logic bin and drop it near the edge of an SLR region in the Design Plan Editor View.

     Notice that the SLR region turns the color blue prior to dropping the instance.
   - Select an instance, right-click and select the Assign to menu; then choose from the list of SLR regions (for example, SLR0, SLR1, or SLR2).

Copyright © 2013 Synopsys, Inc.

4. Save this Design Plan file.

SLR Dependencies

Here are some dependencies to consider when assigning logic to SLRs:

- All assignments can be replicated to an SLR region and other regions.
- User-defined regions must be fully contained in an SLR region and cannot cross SLR boundaries.
- The SLR assignments are saved to the design floorplan file (`sfp`). The content of the SFP file is shown below.

```
1  assign_target {xc7v1500tflg1761}
2  create_block_partition {XC7V1500TFLG1761.Chip} 1 1 450 259 RAMB:1_1_90_11 MULT:1_1_180_9
3  create_block_partition {XC7V1500TFLG1761.SLR0} 1 1 150 259 RAMB:1_1_29_11 MULT:122_1_180_9
4  create_block_partition {XC7V1500TFLG1761.SLR1} 151 1 300 259 RAMB:31_1_59_11 MULT:62_1_120_9
5  create_block_partition {XC7V1500TFLG1761.SLR2} 301 1 450 259 RAMB:61_1_89_11 MULT:2_1_60_9
6  assign_partition {PrgmCntr} {XC7V1500TFLG1761.SLR0}
7  assign_location {XC7V1500TFLG1761.Chip} 0 0 0 0
8  assign_location {XC7V1500TFLG1761.SLR0} 0 0 0 0
9  assign_location {XC7V1500TFLG1761.SLR1} 0 0 0 0
10 assign_location {XC7V1500TFLG1761.SLR2} 0 0 0 0
11 assign_property -speedgrade {XC7V1500TFLG1761} -1
```

- When SRL assignment conflicts exist, the software honors lower-level hierarchical instances within a hierarchy. For example, suppose instance A contains instances B and C. If instance A is assigned to SLR0 and instance B is assigned to SLR1, then the assignments to SLR0 are maintained for instances A and C. However, the synthesis software still honors instance B assignment to SLR1.

For more information about assigning SLR regions for the device, see *Xilinx Stacked Silicon Interconnect (SSI) Technology*, on page 2065.

Copyright © 2013 Synopsys, Inc.

Working with Xilinx Regions

Here are guidelines for using the Synplify Premier Design Planner with Xilinx devices. You can interactively create regions and assign logic to them for Xilinx devices in the Design Plan Editor view. The Design Planner supports Xilinx Virtex-5 and later technologies.

To manually create regions, you can use the syn_assign_to_region attribute. For details, see *syn_assign_to_region*, on page 1230.

For more information see the following:

- Xilinx Device Resources, on page 650
- Creating Regions for Xilinx Designs, on page 651

Xilinx Device Resources

The Xilinx devices can include resources like DSP elements, RAM blocks, I/O banks, IP, and PCIE bus. The resources are located within the device or along its perimeter, depending on the technology family you select. The number of resources vary with the technology family.

Creating Regions for Xilinx Designs

The following vendor-specific guidelines are intended to supplement the procedure described in *Creating Regions*, on page 629. Use the following recommendations to help you design plan regions in the Design Plan Editor for Xilinx devices:

Copyright © 2013 Synopsys, Inc.

- Base your placement on the Configurable Logic Block (CLB) coordinate system. Typically, the bottom-left corner is the row 1, column 1 location on the device.
- Physical Plus guides placement for the regions you create on the Xilinx device.

Using Process-Level Hierarchy

Depending on the technology you use, process-level hierarchy can affect design performance positively or negatively. The tendency is to affect Xilinx designs positively and Altera designs slightly negatively.

Process-level hierarchy is turned off by default in the Synplify Premier UI. The mapper treats designs with and without process-level hierarchy in the same way. However, if you have process-level hierarchy, there are extensive name changes, which can affect the mappers and the place-and-route tools.

Bit Slicing

Bit slicing is a technique you can use when a primitive is too large to fit into a region, or when you want more granularity to control placement. It allows you to break up large primitives into smaller ones, which you can then place in different regions.

The following describe bit slicing in more detail

- Using Bit Slicing, on page 652
- Bit Slice Examples, on page 657

Using Bit Slicing

1. In the Synplify Premier project window, open a new (File->New->Netlist Restructure File) or an existing `nrf` file.

The `nrf` file is a netlist restructure file that defines the logical division of primitive outputs. The tool reads the slice_primitive commands in this file which define the division. For information about this command and its use in a script, see slice_primitive in the Chapter 17, *Batch Commands and Scripts* of the *Reference Manual*.

If you have an existing `tcl` file, you can view it in this graphical interface by renaming it with a `nrf` extension, and then opening it as described.

2. Type in or drag and drop the instance to slice from the RTL view into this tab.

   For bit slicing, you can only divide bus primitives of the following types:

buf	or	register
inv	xor	mux
tristate	and	latch

3. Set bit-slicing preferences.
   - To slice an instance by a specified number of bits per slice or by a specified number of slices, see the details in *Slicing an Instance into a Specified Number of Slices*, on page 654.
   - To divide an instance into slices of varying widths, see the procedure in *Custom Slicing*, on page 655.
   - To globally bit slice all instances of the same type in the netlist, select Slice all instances of this type.

Copyright © 2013 Synopsys, Inc.

4. Save the file.

   The Project view now shows the netlist restructure folder.

5. Select (Project->Implementation Options) and click the GCC & Prototyping Tools tab. Make sure that the netlist restructure file that you just created is checked in the Netlist prototype files section, and click OK.

6. Select Run->Compile Only (F7) to run netlist restructuring on your design. The sections of the sliced element are displayed and can now be individually assigned.

Slicing an Instance into a Specified Number of Slices

The following procedure shows you how to slice an instance by a specified number of bits per slice or into a specified number of slices. To slice into varying bit widths, see *Custom Slicing*, on page 655.

1. Open the Bit Slices dialog box by opening an `nrf` file as described in *Using Bit Slicing*, on page 652.

2. Enter a value.

To create...	Do this..
Slices with a specified number of bits per slice	Click the Bits per Slice button and enter a value for the number of bits. The tool allocates n instance for each group of bits, and allocates any remaining bits to the last instance. For an example, see *Slicing into Primitives of Equal Size*, on page 657.
A specific number of slices	Click the Slices button and enter a value for the number of slices. For an example, see *Slicing into Predefined Primitives*, on page 658. The tool divides the bits equally between the specified number of instances, and assigns any partial numbers to the last instance.

You can now return to the rest of the bit-slicing procedure described in *Using Bit Slicing*, on page 652.

Custom Slicing

The following procedure shows you how to define slices of varying widths. For a specified number of slices, see *Slicing an Instance into a Specified Number of Slices*, on page 654.

1. Open the Bit Slices dialog box by opening an `nrf` file as described in *Using Bit Slicing*, on page 652.

2. Click the Custom button. This enables the MSB/LSB table and the Slice button.

3. To define a slice, do the following:

   – Select the top entry in the table, then click on the Slice button. This displays the Select New Slice MSB.

   – Either click OK to slice the number of bits into two or enter the starting MSB for the next slice.

   The upper limit of the bit range is always one less than the previously assigned MSB so that each slice is at least one bit wide. When you click OK, the table is updated and the Slice button is again enabled, so you can define a new slice.

4. Continue to select entries in the table and click Slice to redisplay the Select New Slice MSB popup menu and define the additional slices.

   See *Slicing into Predefined Primitives*, on page 658 for an example of custom slicing.

5. To undo an entry, merge the entries by doing the following:

Copyright © 2013 Synopsys, Inc.

- Select two (or more) adjacent slice definitions by holding down the Ctrl key and clicking the table entries to select them.
- Click Join.

You can now return to the rest of the bit-slicing procedure described in *Using Bit Slicing*, on page 652.

Setting Viewing Options for Bit Slices

This procedure shows you how to set some viewing options.

1. To keep the current level as it is and only view the effects of bit slicing one level down in the design hierarchy, go to the Bit Slices tab of the netlist restructure file GUI and enable Preserve the Hierarchical View.
2. To view information about the bits, do the following:
   - Select a group of bits in an HDL Analyst view.
   - Right-click and select Properties. A dialog box displays the bit slicing properties for the primitive. Click OK to dismiss this dialog box.

Bit Slice Examples

The following examples illustrate two different cases of bit slicing a 96-bit bus XOR primitive. The following figure shows the primitive before bit slicing.

Slicing into Primitives of Equal Size

In this example, the Bits per Slice value is set to 36. The tool divides the output of the y[95:0] primitive into three individual primitives. The first two primitives each contain the requested 36 bits; and the last primitive contains the remaining 24 bits (y[95:72]). The following figure shows the results of this bit slicing RTL view.

Copyright © 2013 Synopsys, Inc.

Slicing into Predefined Primitives

In this example, the Custom setting is used to define three individual primitives with widths of 48, 32, and 16. For more explanation about defining custom slices, see *Custom Slicing*, on page 655.

The RTL view for this bit slicing example is shown in the following figure.

Copyright © 2013 Synopsys, Inc.

CHAPTER 13

Analyzing Designs in Physical Analyst

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

- Analyzing Physical Synthesis Results, on page 662
- Using Physical Analyst, on page 665
- Displaying and Selecting Objects, on page 671
- Querying Physical Analyst Objects, on page 681
- Finding Objects, on page 686
- Crossprobing in Physical Analyst, on page 695
- Analyzing Netlists in Physical Analyst, on page 703
- Using Implementation Maps in Physical Analyst, on page 710

For information about analyzing timing in the Synplify Premier Physical Analyst tool, see *Using Auto Constraints*, on page 739.

Copyright © 2013 Synopsys, Inc.

Analyzing Physical Synthesis Results

This section contains information about tools you can use to analyze physical synthesis results. See the following:

- Analyzing Physical Synthesis Results Using Various Tools, on page 662
- Running Multiple Implementations, on page 663
- Running Multiple Implementations, on page 663
- Checking Altera Pre-Placement Physical Synthesis Results, on page 663

Analyzing Physical Synthesis Results Using Various Tools

Default timing and area reports are presented in the `htm` or `srr` log file for the design project. To view this information, click the View Log button in the Project view (or View->View Log File). The following `htm` log file shows both the Table of Contents and the HTML log file contents for the design.

See *Viewing the Log File*, on page 514 for complete information on how to interpret the log file results.

In addition, you can generate a stand-alone timing report to display more or less information than what is provided in the log file. See the following:

- Generating Custom Timing Reports with STA, on page 729

Also, check the place-and-route results to determine if further synthesis is required. For example, click on Xilinx P&R Report to check the xflow_par.log file to verify that all constraints were met as shown below. For graph-based physical synthesis, you can also click on Initial Placement Report to check the xflow_gp.log file. Click on Quartus P&R Report to check the quartus.log file for place-and-route results for Altera devices.

Running Multiple Implementations

You can create multiple implementations of the same design so that you can compare the results of each implementation and place-and-route run. This lets you experiment with different settings for the same design with different place-and-route options. Implementations are revisions of your design within the context of the Synplify Premier software and do not replace external source code control software and processes.

For the Graph-based physical synthesis with a design plan flow, you can run the first pass using the Synplify Premier software without a design plan file (`sfp`) to synthesize the design. Placement and routing runs automatically. Then, create a new implementation and apply a design plan for Design plan-based physical synthesis.

See *Working with Multiple Implementations*, on page 226 for more information.

Checking Altera Pre-Placement Physical Synthesis Results

In Altera designs, graph-based physical synthesis generates an intermediate file that you can display in the Technology View and use for debugging. Do the following:

1. After physical synthesis, open the preplace.srm file by double-clicking or right-clicking the file in the Project view and selecting Open.

Copyright © 2013 Synopsys, Inc.

The preplace.srm file is an intermediate file that captures the netlist after RTL physical synthesis and immediately before global placement, showing the same results as would be obtained from logic synthesis. This file is located in the physical synthesis implementation results directory.

2. To view the corresponding timing slack for all the clocks in the design, open the log file, and check the Pre-placement Timing Snapshot section of the log file for the critical path reflected in the preplace.srm file.

```
C:/synplify_premier/tutorial/rev_2/par_1/xflow_par.log
284   Constraint                                  |  Check   | Worst Case |  Best Case | Timing |    Timing
285                                               |          |    Slack   | Achievable | Errors |    Score
286 ----------------------------------------------------------------------------------------------------------
287   TS_clock = PERIOD TIMEGRP "clock" 8 ns HI | SETUP    |     0.062ns|     7.938ns|       0|
288   CH 50%                                    | HOLD     |     0.513ns|            |       0|
289 ----------------------------------------------------------------------------------------------------------
290   PATH "TS_special_regs_status_7__path" TIG | SETUP    |         N/A|     9.547ns|     N/A|
291 ----------------------------------------------------------------------------------------------------------
292
293
294 All constraints were met.
295 INFO:Timing:2761 - N/A entries in the Constraints list may indicate that the
296    constraint does not cover any paths or that it has no requested value.
297
298
299 Generating Pad Report.
300
301 All signals are completely routed.
302
303 Total REAL time to PAR completion: 2 mins 5 secs
304 Total CPU time to PAR completion: 41 secs
305
306 Peak Memory Usage:  234 MB
307
308 Placement: Completed - No errors found.
309 Routing: Completed - No errors found.
310 Timing: Completed - No errors found.
311
312 Number of error messages: 0
313 Number of warning messages: 0
```

For more information on analyzing synthesis results graphically, see the following:

- Using Physical Analyst, on page 665
- Chapter 11, *Analyzing with HDL Analyst and FSM Viewer*

Using Physical Analyst

After you have placed and routed your design with backannotated information, you can use the Physical Analyst tool to analyze the placement and global routing. For descriptions of the interface, see the *Reference Manual*.

The Physical Analyst functionality is only available for the Synplify Premier flows:

- Graph-based physical synthesis (*Altera Graph-Based Physical Synthesis*, on page 46)
- Graph-based physical synthesis with a design plan (*Altera Graph-Based Physical Synthesis with Design Planner*, on page 49)

For both these flows, you can also include a place-and-route implementation with backannotation. The tool will display the placement information after physical synthesis is run, for the following technologies:

Xilinx	Virtex-6, Virtex-5, and Virtex-4
Altera	Stratix IV, Stratix III, Stratix II GX,and Stratix II

See the following for more information:

- Opening the Physical Analyst Interface, on page 665
- Zooming in the Physical Analyst, on page 667
- Moving Between Views in the Physical Analyst, on page 668
- Using the Physical Analyst Context Window, on page 669

Opening the Physical Analyst Interface

The following procedure shows you how to open the tool and the control panel.

1. Open the Physical Analyst view in any of the following ways:
   - Click on the Physical Analyst icon ( ) from the Physical Analyst toolbar.
   - Select HDL Analyst->Physical Analyst in the Project view.

Copyright © 2013 Synopsys, Inc.

– Select the `srm` file, then right-click and select Open Using Physical Analyst from the popup menu.

2. To display the control panel for the Physical Analyst, do one of the following:
   – Click on the Physical Analyst Control Panel icon ( ) in the Physical Analyst toolbar.
   – Select Options->Physical Analyst Control Panel.
   – Use the keyboard shortcut key Ctrl-k.

3. To close the Physical Analyst control panel, use any of the toggle methods listed in the previous step, or right-click in the control panel and select Hide from the popup menu.

Zooming in the Physical Analyst

Since the objects displayed in the Physical Analyst full view might be very small, a handy command to use is Zoom Selected. After selecting one or more objects in the Physical Analyst view, you can access this command by

1. If you do not have objects selected, use any of the following global zoom commands to change the display:
   - View->Zoom In from the menu or the Zoom In ( )icon.
   - View->Zoom Out from the menu or the Zoom Out ( )icon.
   - View->Full View from the menu or the Full View ( ) icon.
   - View->Normal View from the menu or the Normal View ( ) icon.
   - Appropriate mouse strokes.

Copyright © 2013 Synopsys, Inc.

For a description of the zoom options, see *View Menu*, on page 160 in the *Reference Manual*. For a description of the mouse strokes, see Help-> Mouse Stroke Tutor.

2. To zoom into a particular object or area, select the objects and then use the Zoom Selected command in one of the following ways:
   - Click the Zoom Selected ( ) icon.
   - Right-click and select Zoom Selected from the popup menu.
   - Use the following mouse stroke.

The Zoom Selected command centers the selected object or objects in the view.

Moving Between Views in the Physical Analyst

When you filter or expand your design, you move through a number of different design views in the same window. For example, you might start with a view of the entire design, zoom in on an area and filter an object, and finally expand a connection in the filtered view, for a total of three views.

1. To move back to the previous view, click the Back icon or draw the appropriate mouse stroke in the Device window.

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

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

   The software displays the next view in the display history.

Using the Physical Analyst Context Window

The Physical Analyst context window occupies the lower portion of the Control Panel view. The context window provides a point-of-reference to your location on the device.

For example, suppose you:

1. Zoom in the Physical Analyst tool to get a better view of the objects you selected on the device.
2. The context window in the control panel displays a rectangle around the relevant area on the device. This is helpful because you now have a point-of-reference to your location on the device.

Copyright © 2013 Synopsys, Inc.

3. Once a rectangle area is drawn in the context window, you can then:
   - move
   - scroll
   - stretch/shrink

   this rectangle in the context window. This view will be reflected in the Physical Analyst view.

4. To reinstate the full context window view, right-click and select Refresh in this view.

Displaying and Selecting Objects

This section describes how to display and select objects in the Physical Analyst view.

- Setting Visibility for Physical Analyst Objects, on page 671
- Displaying Instances and Sites in Physical Analyst, on page 672
- Displaying Nets in Physical Analyst, on page 676
- Selecting Objects in Physical Analyst, on page 678

Setting Visibility for Physical Analyst Objects

You determine object visibility and selectability by setting it in the control panel.

1. Select Options->Physical Analyst Control Panel to display the control panel.
2. To make an object visible, select the Vis box for that object.

   This makes the boundaries for the selected type of object visible in the Device view. You cannot select a visible object, unless it has been made selectable. For details about object selection, see *Displaying Instances and Sites in Physical Analyst*, on page 672, and *Displaying Nets in Physical Analyst*, on page 676.
3. To make an object selectable, select the Vis and Sel boxes for that object.

   The object must be visible before you can make it selectable. When an object is selectable, you can get detailed information by rolling the mouse over it to get a tool tip.

Copyright © 2013 Synopsys, Inc.

The Control Panel will display the following in the Physical Analyst View:

- Instances visible and selectable
- Instance internals visible
- Do not show internal signal pins
- Do not show enhanced view for instances
- Signal nets visible and selectable
- Do not show signal flow
- Show pruned signals
- Sites visible

Displaying Instances and Sites in Physical Analyst

The following procedure shows you how to display objects and sites in the Physical Analyst window.

1. To display instances, open the PhysGical Analyst control panel and select the Vis box for Instances.

   The window displays instance bounds, instance locations, and signal pins if all instances with placement information. It does not display the signal pins of the instances.

The following figure shows cell boundaries.

2. To display core cells at a fixed size regardless of zoom level, do the following:
   – With the Physical Analyst window active, select View->Configure Enhanced Instance Display.

Copyright © 2013 Synopsys, Inc.

– Enable the Enhance Instance Shape for Better Visibility option.

– Set any other options you want and click OK.

The tool displays the core cells as diamonds of a fixed size.

3. To display instances and their signal pins, select the Vis boxes for Instances, Inst Display.

Copyright © 2013 Synopsys, Inc.

The Vis box for Signal Pins is selected automatically, and the tool displays the signal pins. The following figure shows signal pins displayed:

4. To display sites, do the following:
   - Select the Vis box for Sites.
   - To view sites more clearly, turn off the visibility of instances.

Displaying Nets in Physical Analyst

When nets are routed, they are connected to their respective instances. Because of the long load time and the limited visibility when nets are superimposed on the view, net routes are not displayed by default. Nets are routed from output pins to input pins and are shown with their corresponding point-to-point connections on one layer of the device in the Physical Analyst view.

1. Route the nets, using one of these methods:
   - Use an on-demand routing command like Expand or Show Critical Path. For more netlist commands, see *Analyzing Netlists in Physical Analyst*, on page 703.

- Right-click and select Route Selected Instances. You can choose to display nets for connected instances only, as well as, from input pins, to output pins, or to all pins for selected instances.

2. Set the visibility options on the control panel. Select the Vis box for Nets. You must select this to make the other options available.

   The following example shows point-to-point net routing:

3. To reduce clutter in the display, try the following techniques:
   - Isolate a critical path, or filter the design to show just a few paths you want to analyze.
   - In the control panel, select the Vis box for Prune Signals. When enabled, this option displays net segment that connect to an instance that is invisible because of filtering, for example, in a diminished color.
4. To view the direction of a signal, do either of the following:
   - Place the cursor over a net to view the predominant direction of its signal flow (right, left, up, or down).

Copyright © 2013 Synopsys, Inc.

– Select the Vis box for Signal Flow. The net is displayed with arrows showing the direction of its signal flow. The Signal Flow option is useful when you display the critical path. You can follow the arrows and lines along the critical path from the start point to the end point.

5. To hide all nets in the display, do either of the following:
   – Right-click in the Physical Analyst and select Unfilter->Show All Instances, Hide All Nets from the popup menu.
   – Click the Reset filter icon ( ).

Selecting Objects in Physical Analyst

Selected objects are highlighted in the Physical Analyst view. If you have other windows open and crossprobing enabled, you can highlight the selected object in the other windows too.

The following procedure shows you ways to select objects.

1. To select an object, do the following:
   - In the control panel, select the Sel box for the object to make it selectable.
   - Click on the object in the Device view.
2. To select multiple objects, use one of these methods:
   - Draw a rectangle around the objects.
   - Select an object, press Ctrl, and click other objects you want to select. You can also deselect from the list of currently selected objects while holding the Ctrl key.
   - Position the cursor over an object and click the right mouse button; the object is automatically selected in the view. To preserve a prior selection, hold the Ctrl key and press the right mouse button.
   - Right-click and choose one of the Select commands from the popup menu.
3. To limit the selection range, use these techniques:
   - Use the Physical Analyst control panel to enable or disable a class of objects to be selected. For example you can disable selection for nets, and only select instances in an area.
   - Use the Filter and Unfilter commands to restrict the scope of selection.
   - Use Find to select the objects you want. You can also use the Find command to select a subset of objects of a particular type. See *Using Find to Locate Physical Analyst Objects)*, on page 686.
   - Use the Go to Location command and specify an object location in microns to go directly to a coordinate pair location. See *Finding Physical Analyst Objects by Their Locations*, on page 690.
4. To select a net or instance that overlaps another, do the following.
   - Move your cursor over the overlapping object you want to select. The cursor changes shape to indicate that you need to resolve the selection.
   - Click the cursor, and the Resolve Selection dialog box opens.

Copyright © 2013 Synopsys, Inc.

– Select the object you want from the list and click Close.

Once you have selected an object, the software highlights the selected object in the Physical Analyst window. If you have other windows open and crossprobing enabled, the object is highlighted in the other windows.

Querying Physical Analyst Objects

The following procedures describe how to view object properties in the Physical Analyst:

- Viewing Properties in Physical Analyst, on page 681
- Using Tool Tips to View Properties in Physical Analyst, on page 684

Viewing Properties in Physical Analyst

You can view properties for the device design and for selected objects displayed in the view using the popup menu commands described here. For site properties, use tooltips, as described in *Using Tool Tips to View Properties in Physical Analyst*, on page 684.

1. To view general information for the design, right-click anywhere in the Physical Analyst view, and select Physical Analyst Properties from the popup menu.

   The dialog box shows information like the design name, the number of instances, unplaced instances, routed nets in the design, and the location of the netlist and floorplan (`def`) files.

Copyright © 2013 Synopsys, Inc.

2. To view properties for an instance, right-click the instance and select Properties (Core Cell).

   The dialog box lists information like the instance name, type, and pins; placement information like its placement location and device-specific location; and its delay, slack, and clock signal. It also indicates whether the instance is included in the critical path.

3. To view properties for a net, right-click the net and select Properties (Net).

The dialog box lists information like the net name, logical nets, pin count, and fanout. It also indicates if the net is a clock and if it has been globally routed.

4. Click on an item in the list to view a definition of that term below.

Copyright © 2013 Synopsys, Inc.

Using Tool Tips to View Properties in Physical Analyst

You can use this method to view properties for any objects in the design, as well as for various UI features. For instance, net, and design properties, you can also use the popup commands described in *Viewing Properties in Physical Analyst*, on page 681.

1. Enable View->Tool Tip.

2. Move your mouse over an object.

   As you move the mouse over an object, you see information about that object. Coordinates for the objects are in microns. The following is an example of information for different objects.

3. To display the tooltip information in the Tcl window, do the following:
   - Select an object directly. Do not use this method with area selections or when objects are selected using other commands such as Expand or Find.
   - Either select View->Selection Transcription or right-click in the Physical Analyst view and select Selection Transcription from the popup menu.

   The tooltip information is displayed in the Tcl window. You can copy and paste the information from the TCL window into other windows or files like the SCOPE window, the Find Object dialog box, or a text file.

   The following figure shows an example of an object selected on the device and its tool tip information displayed in the TCL window.

Copyright © 2013 Synopsys, Inc.

Finding Objects

To find and display objects in the Physical Analyst view, use the following options:

- Using Find to Locate Physical Analyst Objects), on page 686
- Finding Physical Analyst Objects by Their Locations, on page 690
- Using Markers to Find Physical Analyst Objects, on page 691
- Identifying Encrypted IP Objects in Physical Analyst, on page 693

Using Find to Locate Physical Analyst Objects)

This procedure shows you how to use the Find command to do a search on the entire design. The view displayed is flat, although the hierarchy of instance names is retained. The Find command does not include physical instances in its search.

1. To find nets, first make sure their display is enabled by selecting View-> Unfilter->Show All or the corresponding command from the popup menu.
2. Right-click and select Find from the popup menu or press Ctrl-f. Move the dialog box so you can see both the view and the dialog box.

   You can also open the dialog box by selecting Edit->Find from the menu or by clicking the Find icon in the tool bar.
3. Select the tab (at the top of the dialog box) for the type of object. The Unhighlighted box on the left will list objects of the selected type (instances, symbols, nets, or ports).

4. You can optionally restrict the scope of your search for the design in the following ways:
   - To filter the search using wildcards, see *Using Wildcards with the Find Command*, on page 688.
   - To further filter the object type, see *Using Object Filters with the Find Command*, on page 689.

   The Unhighlighted box shows available objects within the scope you set when you click Find 200 or Find All. Objects are listed in alphabetical order.

5. Do the following to select objects from the list.
   - Click First 200 or Find All. The Unhighlighted box shows available objects (in alphabetical order) within the scope you set when you click Find 200 or Find All. The former finds the first 200 matches, and then you can click the button again to find the next 200.
   - Move the objects you want to the Highlighted box by double-clicking them, or by clicking them to select them, and then clicking the right arrow. Right-click on the objects you want from the list.

Copyright © 2013 Synopsys, Inc.

– If the object name exceeds the width of the Unhighlighted box, check the object name by clicking the entry in the list, and viewing the entire name in the field below the Unhighlighted box.

Objects transferred to the Highlighted box are automatically highlighted in the view.

You can leave the dialog box open to do successive Find operations. Close the dialog box when you are done.

Using Wildcards with the Find Command

The following procedure shows you how to use patterns or wildcards to restrict the search range with the Find command when you are locating objects in the Physical Analyst view. You can use wildcards to avoid typing long path names. Start with a general pattern, and then make it more specific.

1. Follow the first three steps described in *Using Find to Locate Physical Analyst Objects)*, on page 686.

2. Type a pattern in the Search By Name field.

*	The asterisk matches any sequence of characters.
?	The question mark matches any single character.
.	The dot (period) explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not as a hierarchy separator, type a backslash (\) before the dot.

When you use wildcards between hierarchies, all pattern matching is displayed from the top level to the lowest level hierarchy, inclusively.

3. Click First 200 or Find All.

The Unhighlighted box lists the objects that match the wildcard pattern criteria. If you selected First 200, it lists the first 200 matches, and then you can click the button again to find the next 200.

4. Select and move the objects you want to the Highlighted box by doing one of the following:
   - Select the objects in the Unhighlighted box and click the right arrow.
   - Double-click individual items in the Unhighlighted box.

   The objects are automatically highlighted in the view.

Using Object Filters with the Find Command

Use the Filter Search option on the Find Object dialog box to limit the objects you are searching for to a particular subcategory. This can be very useful in large designs.

1. Follow the first three steps described in *Using Find to Locate Physical Analyst Objects)*, on page 686.

   At this point, you have already restricted your search to a certain type of object by selecting one of the tabs at the top.

2. Select a subcategory from the pull-down list in Filter Search.

   The listed subcategories are for the kind of object you have already selected. For descriptions of the various subcategories, see *Object Filter Search for Find Command*, on page 589 of the Reference Manual.

3. Click First 200 or Find All.

   The Unhighlighted box lists the objects that match the filtered object criteria. If you selected First 200, it lists the first 200 matches, and then you can click the button again to find the next 200.

4. Select and move the objects you want to the Highlighted box by doing one of the following:
   - Select the objects in the Unhighlighted box and click the right arrow.
   - Double-click individual items in the Unhighlighted box.

   The objects are automatically highlighted in the view.

Copyright © 2013 Synopsys, Inc.

Finding Physical Analyst Objects by Their Locations

The Go to Location command allows you to specify a coordinate pair location or location of an object, and then zoom in on this location if requested.

This procedure shows you how to use the Go to Location command to search for objects, such as instances.

1. In the Physical Analyst view, type Ctrl-g or right-click and select Go to Location from the popup menu.

   The command displays the Goto Location dialog box.

2. Do one of the following:
   - Enter the coordinates of the location (see step 3 for details).
   - Select a marker from the Marker pull-down. For information about creating markers at object locations, see *Using Markers to Find Physical Analyst Objects*, on page 691.

3. Enter a coordinate pair (X and Y) location value in microns. You can do this in any of these ways:
   - Type a coordinate pair in the field.

The syntax is very flexible, providing various ways to separate coordinates. You can use a space, or one of the following punctuation marks: a comma, semi-colon, or colon. Optionally, enclose the coordinate pair location in parentheses.

- Copy and paste a coordinate pair location from a log file (`srr`) or timing analyst file (`ta`).
- Copy a location from a `def` file. The unit of measurement in the `def` file is database units. Use the UNITS DISTANCE MICRONS factor from the `def` file to convert database units to microns, before using it here.
- If you have used the command before and have a history of locations, select a location from the pull-down list in the History field.

A description of the object shows in the dialog box window, if applicable.

4. Select a zoom mode.
   - To center the location, without zooming, select Scroll.
   - To zoom into the selected area, select Zoom to Object.
   - To zoom at the 100% level, select Zoom Normal.
5. Click OK.

   The Physical Analyst view shows the location you specified, at the zoom level you specified. The command keeps a running history of the locations you specified, and they appear in the History pull-down the next time you use the command.

Using Markers to Find Physical Analyst Objects

Markers are bookmarks for physical coordinates in the Physical Analyst view. Markers are useful for analyzing floorplan placement in the Physical Analyst view. You can find an object, such as an instance, then create a marker on this instance. When multiple markers are defined, you can move from marker to marker, as well as, measure the distance from a marker or between any two markers.

1. You can create a marker in either of these ways:
   - Select an object or click on the spot where you want to place the marker. Type Ctrl-m, or right-click and then select Markers->Add Marker from the popup menu. If a marker is created at an instance or net

Copyright © 2013 Synopsys, Inc.

location, the marker's name is Marker_*objectName*. All other markers are named Marker1, Marker2, etc.

– Click on the spot where you want to create the marker. Either type Ctrl-g or right-click and select Go to Location to open the Go to location dialog box. Your coordinates appear in the coordinates field. Check the Create Marker box in the dialog box. Either specify a name for the marker, or use the default marker name, which is GotoMarker1, GotoMarker2, etc. Click OK.

A marker symbol ( ) appears in the Physical Analyst view at the location specified. As you move the cursor over the marker, a tool tip shows the marker name and its X and Y coordinates. You can also view this information by selecting the marker, right-clicking, and selecting Properties. The marker is automatically added to the list in the Go to Location dialog box, and you can use it to locate objects, as described in *Finding Physical Analyst Objects by Their Locations*, on page 690.

2. To move a marker, select the symbol ( ). Press the mouse button, drag the marker to its new location, and then release the mouse button.

3. To delete a marker, select the marker and press the Del key. Alternatively, right-click and select Markers->Remove Selected from the popup menu. To delete all markers, right-click and select Markers-> Remove All from the popup menu.

4. To use markers to locate an object, see the procedure described in *Finding Physical Analyst Objects by Their Locations*, on page 690.

5. Do the following to use markers for measuring distances:
   - To measure the distance from a marker to the cursor location, select the marker and position the cursor. The status bar at the bottom of the Physical Analyst view displays the manhattan distance (X+Y) between the two points, calculated in microns. It also displays the XY coordinates for the cursor.
   - To measure the distance between two markers, select two markers. The distance is displayed in the status bar. If you have more than two markers selected, the distance is not calculated.

6. To navigate from one marker to another, do the following:
   - To advance to the next marker, right-click and select Markers->Go to Next from the popup menu, or use the F2 key.
   - To go to the previous marker, right-click and select Markers->Go to Previous from the popup menu or use the Shift+F2 keys.

   The sequence for the markers is the order in which they were created. If the view is zoomed, the selected marker is centered in the view.

Identifying Encrypted IP Objects in Physical Analyst

You can use the Physical Analyst to identify cells that belong to an encrypted IP. When a cell belongs to an encrypted IP, it is implemented as the type LUT.

To view objects that might belong to encrypted IPs in the Physical Analyst view, perform the following tasks:

1. Select the cell.
2. Right-click and select Properties (Core Cell) from the popup menu.
3. Notice that the Type property is set to LUT in the dialog box.

Copyright © 2013 Synopsys, Inc.

Core Cell Properties
Properties
u1.out
Core Cell
Property
Value
Name
u1.out
Type
LUT
Pins
5
Inputs
4
Outputs
1
Location
(60.00,197.50)
Device Location
SLICE_X0Y4
Delay
0.0000
Slack
0.0000
Property Description
OK
Cancel
Apply
Help

Crossprobing in Physical Analyst

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. The Physical Analyst responds to incoming cross probes as well as sending out cross probes in response to selections.

For details, see

- Crossprobing from the Physical Analyst View, on page 695
- Crossprobing from a Text File to Physical Analyst, on page 698
- Crossprobing from the RTL View to Physical Analyst, on page 699
- Crossprobing from the Technology View to Physical Analyst, on page 701

Crossprobing from the Physical Analyst View

1. Set crossprobing options.
   - To crossprobe automatically from the Physical Analyst view, check that View->Send Crossprobes when selecting is enabled.
   - To crossprobe only on demand, disable View->Send Crossprobes when selecting. This can be more efficient with large designs.
2. To crossprobe to different files and views, follow the appropriate steps:

To Crossprobe to..	Procedure
Source code	• Make sure that View->Crosspobing->Crossprobing to HDL Source is enabled. • In the Physical Analyst view, double-click the object you want to crossprobe. If the source code file is not open, a Text Editor window opens to the appropriate section of code (for example, modules or instances). If the source file is already open, the software scrolls to the correct section of the code and highlights it.

Copyright © 2013 Synopsys, Inc.

To Crossprobe to..	Procedure
RTL view	• Open the RTL view. • Make sure the crossprobing options (see step 1) are set for your needs. • In the Physical Analyst view, select the object you want to crossprobe. The software highlights the corresponding object in the RTL view.
Technology view	• Open the Technology view. • Make sure the crossprobing options (see step 1) are set for your needs. • In the Physical Analyst view, select the object you want to crossprobe. The software highlights the corresponding object in the Technology view.

Copyright © 2013 Synopsys, Inc.

The following shows on-demand crossprobing from the Physical Analyst view:

Copyright © 2013 Synopsys, Inc.

Crossprobing from a Text File to Physical Analyst

Instances from a text file, such as the HDL source code (Verilog/VHDL) or log file (`srr`) can be highlighted in the Physical Analyst. Make sure the Physical Analyst view is already open.

1. Open the Physical Analyst view.
2. In the text file, highlight the appropriate portion of the text, like the hierarchical instance name.

   For some objects in source code files, you might have to select an entire block of text.
3. Crossprobe the object.
   - To show only the object selected, click the Filter Schematics icon in the toolbar.
   - Right-click in the text file and select Select in Analyst from the popup menu.
4. Check the Physical Analyst view.

   The selected instances are highlighted in this view. If the selected object does not have visibility enabled for it in the Control panel, the visibility will be automatically enabled.

Copyright © 2013 Synopsys, Inc.

Log File

```
Ending Points with Worst Slack
******************************

                                Starting                                                    Required
Instance                        Reference     Type     Pin     Net                          Time        Slack
                                Clock
-----------------------------------------------------------------------------------------------------------------
UC_ALU.aluz                     Clk           FDC      D       N_123_i                      6.667       -0.660
PrgmCntr.pc[5]                  Clk           FDPE     D       pc_8[5]                      6.404       0.046
Dmux.alua[1]                    Clk           FDC      D       ALUA_d[1]                    6.441       0.132
Dmux.alua_fast[0]               Clk           FDC      D       ALUA_d[0]                    6.404       0.177
Dmux.alub_fast[5]                             FDC      D       ALUB_d[5]                    6.404       0.288
Dmux.alub[5]                                           D       ALUB_d[5]                    6.441       0.336
Dmux.alub[3]                                           D       ALUB_d[3]                    6.441       0.406
SPECIAL_REGS.portbr                                    CE      portbregister_1_sqmuxa_1     6.206       0.432
SPECIAL_REGS.portbr                                    CE      portbregister_1_sqmuxa_1     6.206       0.432
SPECIAL_REGS.trisb_                                    CE      trisb_reg_1_sqmuxa           6.206       0.462
=================================================================================================================
```

Physical Analyst View

5. If needed, use the Reset filter icon to re-display the unfiltered objects, if you filtered the view in step 3.

Crossprobing from the RTL View to Physical Analyst

Follow this procedure to crossprobe from the RTL view to the Physical Analyst view.

1. Open the Physical Analyst view.
2. Enable the View->Cross Probing->Cross Probing from RTL Analyst option.
3. Click the object (instance or macro) in the RTL view to highlight and crossprobe it. You can use the schematic view, hierarchy browser, or the Object Query dialog box to select the object.

Copyright © 2013 Synopsys, Inc.

You can cross probe hierarchical objects in the RTL view to the set of objects for which the hierarchy is synthesized in the Physical Analyst view. You cannot cross probe primitives in the RTL view which do not have a counterpart in the mapped netlist. The tool highlights all objects relating to the RTL object. For example, if you selected a module, all mapped objects with physical information that implement the module in the Physical Analyst view are highlighted.

Crossprobing from the Technology View to Physical Analyst

Follow this procedure to crossprobe from the RTL view to the Physical Analyst view.

1. Open the Physical Analyst view.
2. Enable the View->Cross Probing->Cross Probing from Tech Analyst option.
3. Click the object in the Technology view to highlight and crossprobe. You can use the schematic view, hierarchy browser, or the Object Query dialog box to select the object.

   When you select an instance in the Technology view, the placed primitive that corresponds to that instance in the Physical Analyst is highlighted.
4. To automatically route cross-probed instances, enable the View->Cross Probing->Auto route cross probe insts option (the option is enabled by default).

   The following shows cross probing from the Technology view to the Physical Analyst view when this option is enabled.

Copyright © 2013 Synopsys, Inc.

Physical Analyst View

Analyzing Netlists in Physical Analyst

In the Physical Analyst view, there are a number of commands for tracing logic and analyzing the netlist, which can be accessed from the right-click popup menus. These commands are context-sensitive, depending on the selected object and where you click. See Chapter 3, *User Interface Commands* of the *Reference Manual* for a complete list of View menu and Physical Analyst popup menu commands.

See the following for details:

- Filtering the Physical Analyst View, on page 703
- Expanding Pin and Net Logic in Physical Analyst, on page 704
- Expanding and Viewing Connections in Physical Analyst, on page 709

Filtering the Physical Analyst View

Filtering is a useful first step in analysis, because you can focus on the relevant parts of the design. Some commands, like the Expand Paths commands, automatically generate filtered views. This procedure only discusses manual filtering, where you use the Filter command to isolate selected objects.

1. Select the objects that you want to isolate.
2. Select the filter command, using one of these methods:
   - Select Filter->Show Selected from the Physical Analyst View menu or from the right-click popup menu.
   - Click the Filter Schematics icon ( ).
   - Press Alt and draw a narrow V-shaped mouse stroke in the schematic window. See Help->Mouse Stroke Tutor for an illustration.

   The software filters the design and displays the selected objects in a filtered view. You can now analyze the objects and perform operations like tracing paths, building up logic, filtering further, finding objects, hiding objects, or crossprobing.
3. To return to the previous view, click the Back ( ) icon.

Copyright © 2013 Synopsys, Inc.

Expanding Pin and Net Logic in Physical Analyst

When you are working in a filtered view, you might need to include more logic in your selected set to analyze your design. This section describes commands that expand logic fanning out from pins or nets; to expand paths, see *Expanding and Viewing Connections in Physical Analyst*, on page 709.

Use the Expand commands with the Filter and Nets->Visible commands to isolate and connect the logic that you want to examine.

1. To expand logic from a pin, do the following in the Physical Analyst view.

To view..	Do this..
All cells connected to a pin	• Select a pin on the cell instance. • Right-click and select Expand->Selected Pins. See *Expanding Logic Example*, on page 705. If you change your selection to all output pins, all input pins, or all pins, you can use the same command to expand from these points.
All cells that are connected to a pin, up to the next register/port	• Select a pin on the cell instance. • Right-click and select Expand to Register/Port->Selected Pins. See *Expanding Logic to Register/Port Example*, on page 706. If you change your selection to all output pins, all input pins, or all pins, you can use the same command to expand to registers and ports from these points.

Expanding Logic Example

Copyright © 2013 Synopsys, Inc.

Expanding Logic to Register/Port Example

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

To...	Do this...
Select all instances on a net	Select a net and select Select Net Instances->All Pins. The software shows an unfiltered view that includes all the instances connected to the net along the signal path. You can also choose to show output pins or input pins with this command. See the following example.
Highlight all visible instances on a net	Select a net and select Highlight Visible Net Instances->All Pins. You see a filtered view of all instances connected to the selected net along the signal path. You can also select to show output pins or input pins.
Select the net driver	Select a net and select Select Net Driver. The software shows an unfiltered view that includes the driver of the net.
Go to the net driver	Select a net and select Go to Net Driver. The software shows and scrolls to the driver of the net.

Copyright © 2013 Synopsys, Inc.

This example shows instances on a critical path. First, the critical path was filtered. Then one of the nets on the critical path selected and the Select Net Instances->All Pins selected. The figure shows the results.

Expanding and Viewing Connections in Physical Analyst

This section describes commands that expand logic between two or more objects. To expand logic out from a net or pin, see *Expanding Pin and Net Logic in Physical Analyst*, on page 704. You can also isolate the critical path or use the Timing Analyst to generate a schematic for a path between objects, as described in *Using Auto Constraints*, on page 739.

Use the following path commands with the Filter and Nets->Visible commands to isolate and connect the logic that you want to examine.

To expand and view connections between selected objects, do the following:

1. Select two or more objects.
2. To expand the logic, select Expand Paths->All Pins from the popup menu. Alternatively, you can select to expand from selected pins.

Copyright © 2013 Synopsys, Inc.

Using Implementation Maps in Physical Analyst

Currently, the Implementation Maps are a Beta feature in the Physical Analyst tool.

Physical Analyst can display color-coded maps which overlay the device providing information about the design after it has been synthesized. The following Implementation Maps are available in the Synplify Premier Physical Analyst tool for Xilinx devices:

- Routing Congestion
- Block Utilization
- Block Inputs
- Slack

To bring up the implementation maps:

1. Open the Physical Analyst view ( ).
2. Click on the Maps tab in the Physical Analyst control panel.

3. Select the type of map to display from Select Map pull-down menu on the Physical Analyst control panel.

You can choose:

– None - No maps are displayed in the Physical Analyst view.

– Routing Congestion - See *Using the Routing Congestion Map*, on page 713

– Block Component Utilization - See *Using the Block Component Utilization Map*, on page 715

– Block Input Utilization - See *Using the Block Input Utilization Map*, on page 716

– Slack Distribution - See *Using the Slack Distribution Map*, on page 717

4. You can use these maps in conjunction with congestion or utilization map controls and settings. For more information, see *Implementation Maps Controls and Settings*, on page 712.

Copyright © 2013 Synopsys, Inc.

5. You can crossprobe from any implementation map to the HDL Analyst view. For more information, see *Crossprobing from Implementation Maps*, on page 719.

Implementation Maps Controls and Settings

The implementation maps include the following controls and settings on the Maps control panel:

- Display mode
- Slider scale
- Histogram

Two display modes are available. When you enable Show visible design objects, both the map and instances are displayed in the Physical Analyst view. Otherwise, only the map without instances is shown. Show visible design objects is turned off by default.

Two slider scale modes are available; they are Linear and Log.

- In the Linear mode, the threshold and alert sliders and histograms move at a linear rate. The Linear mode is the default setting.
- In the Log mode, the threshold and alert sliders and histograms move at a logarithmic rate.

The Threshold and Alert sliders adjust the threshold and the alert percentage levels, respectively, for the implementation maps. The combination of these values is reflected in the color bar which shows a progression of colors from the threshold level to the alert level. The thermal map uses colors ranging from blue (cool) to red (burn); magenta is used as the alert color.

Histogram - The histograms display bar charts that shows the distribution of routing congestion, block or input pin utilization, or slack utilization for the design. You can select either Linear or Log mode.

Copyright © 2013 Synopsys, Inc.

Using the Routing Congestion Map

The Routing Congestion map displays estimated routing congestion based on placement. Input and output nets for each CLB contribute to routing demands for the design as compared to the routing resources of the technology. Routing congestion areas most likely will cause detours for routing the design. Use the routing congestion map to give you quick feedback on how likely this design can be routed using the back-end tool. The colors of the map alert you to potential congestion and routing problems you might encounter.

To display the Routing Congestion map, select Routing Congestion from the drop-down menu on the control panel's Maps pane. When routing congestion is selected, the routing congestion map is displayed and tool tips reporting the congestion for cell locations are available. You can zoom in or out of this view at any time.

Copyright © 2013 Synopsys, Inc.

The thermal map for congestion applies the following conditions:

- Threshold percentage value is fixed at 40 and Alert percentage value is fixed at 49. You cannot use the slider to change the limits.
- Thermal map displays the following colors:
  - Congestion value of 0 is black
  - Congestion values between 1 and 40 use the coolest color of dark blue
  - Congestion values between 40 and 49 use thermal colors that range from dark blue to red
  - Congestion values greater than 49 use the alert color of magenta

- Use the tool tip to display the severity for the congestion.
  - <40—Low
  - 40-45—Med
  - >45—High

Using the Block Component Utilization Map

The Block Component Utilization map provides visual and analytical display of the percentage of utilization for each CLB in the placeable area. The block utilization data is available after the synthesis tool places the design.

To display the Block Utilization map, open the Physical Analyst and select Block Utilization from the drop-down menu on the control panel's Maps pane. When Block Utilization is selected, the block utilization map is displayed and tool tips reporting the percentage of utilization for each CLB are available. You can zoom in or out of this view at any time.

Copyright © 2013 Synopsys, Inc.

The thermal map for block utilization fixes the Threshold percentage value at 0 and Alert percentage value at 100. You cannot use the slider to change the limits.

Using the Block Input Utilization Map

The Block Input Utilization map provides visual and analytical display of the number of input pins for each CLB in the placeable area. The block input pin data is available after the synthesis tool places the design.

To display the Block Inputs map, open the Physical Analyst and select Block Inputs from the drop-down menu on the control panel's Maps pane. When Block Inputs is selected, the block input pins map is displayed and tool tips reporting the number of input pins used for each CLB location are available. You can zoom in or out of this view at any time.

The thermal map for block inputs fixes the Threshold percentage value at 40 and Alert percentage value at 49. You cannot use the slider to change the limits.

Using the Slack Distribution Map

The Slack map provides visual and analytical display of the instances with slack values between 2.0 ns and -2.3237 ns. Use the threshold and alert limits to adjust the display for slack values within these specified limits. The available instance slack is displayed after the design is synthesized.

Copyright © 2013 Synopsys, Inc.

To display the Slack Distribution map, open the Physical Analyst and select Slack Distribution from the drop-down menu on the control panel's Maps pane. When Slack Distribution is selected, the slack map is displayed and tool tips reporting information about the core cell and its corresponding slack value are included. You can zoom in or out of this view at any time.

Crossprobing from Implementation Maps

Crossprobing is feasible from any of the implementation maps.

1. Bring up any of the implementation maps from the Physical Analyst view: Congestion map, block utilization map, block inputs map, or slack map.

2. Turn on the Show visible design objects option, so that instance locations are displayed in the Physical Analyst view as well.

3. Zoom in to a hot spot region. Select an instance or instance region.

Copyright © 2013 Synopsys, Inc.

4. Open the HDL Analyst RTL or Technology view. Then, select Filter Schematic ( ).

CHAPTER 14

Analyzing Timing

This chapter describes typical analysis tasks. It describes graphical analysis with the HDL Analyst tool as well as interpretation of the text log file. It covers the following:

- Analyzing Timing in Schematic Views, on page 722
- Generating Custom Timing Reports with STA, on page 729
- Using Analysis Design Constraints, on page 732
- Using Auto Constraints, on page 739
- Using the Timing Report View, on page 744
- Analyzing Timing with Physical Analyst, on page 752

Copyright © 2013 Synopsys, Inc.

Analyzing Timing in Schematic Views

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

- Viewing Timing Information, on page 722
- Annotating Timing Information in the Schematic Views, on page 723
- Analyzing Clock Trees in the RTL View, on page 725
- Viewing Critical Paths, on page 725
- Handling Negative Slack, on page 728
- Generating Custom Timing Reports with STA, on page 729

Viewing Timing Information

Some commands, like Show Critical Path, Hierarchical Critical Path, Flattened Critical Path, automatically enable Show Timing Information and display the timing information. The following procedure shows you how to do so manually.

1. To analyze timing, enable HDL Analyst->Show Timing Information.

   This displays the timing numbers for all instances in a Technology view. It shows the following:

Delay	This is the first number displayed. • Combinational logic This first number is the cumulative path delay to the output of the instance, which includes the net delay of the output. • Flip-flops This first number is the path delay attributed to the flip-flop. The delay can be associated with either the input or output path, whichever is worse, because the flip-flop is the end of one path and the start of another.
Slack Time	This is the second number, and it is the slack time of the worst path that goes through the instance. A negative value indicates that timing constraints can not be met.

Annotating Timing Information in the Schematic Views

You can annotate the schematic views with timing information for the components in the design. Once the design is annotated, you can search for these properties and their associated instances.

1. On the Device tab of the Implementation Options dialog box, enable Annotated Properties for Analyst.

For each synthesis implementation and each place-and-route implementation, the tool generates properties and stores them in two files located in the project folder:

`.sap`	Synplify Annotated Properties Contains the annotated design properties generated after compilation, like clock pins.
`.tap`	Timing Annotated Properties Contains the annotated timing properties generated after compilation.

2. To view the annotated timing, open an RTL or Technology view.
3. To view the timing information from another associated implementation, do the following:
   - Open an RTL or Technology view. It displays the timing information for that implementation.
   - Select HDL Analyst->Select Timing, and select another implementation from the list. The list contains the main implementation and all

Copyright © 2013 Synopsys, Inc.

associated place-and-route implementations. The timing numbers in the current Analyst view change to reflect the numbers from the selected implementation.

In the following example, an RTL View shows timing data from the test implementation and the test/pr_1 (place and route) implementation.

4. Once you have annotated your design, you can filter searches using these properties with the find command.
   - Use the find -filter {@*propName*>=*propValue*} command for the searches. See *Find Filter Properties*, on page 1786 in the *Reference Manual* for a list of properties. For information about the find command, see *Tcl find Command*, on page 1774 in the *Reference Manual*.
   - Precede the property name with the @ symbol.

   For example to find fanouts larger than 60, specify find -filter {@fanout>=60}.

Analyzing Clock Trees in the RTL View

To analyze clock trees in the RTL view, do the following:

1. In the Hierarchy Browser, expand Clock Tree, select all the clocks, and filter the design.

   The Hierarchy Browser lists all clocks and the instances that drive them under Clock Tree. The filtered view shows the selected objects.

2. If necessary, use the filter and expand commands to trace clock connections back to the ports and check them.

   For details about the commands for filtering and expanding paths, see *Filtering Schematics*, on page 593, *Expanding Pin and Net Logic*, on page 595 and *Expanding and Viewing Connections*, on page 599.

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

   If you do not define your clock constraints accurately, you might not get the best possible synthesis optimizations.

Viewing Critical Paths

The HDL Analyst tool makes it simple to find and examine critical paths and the relevant source code. The following procedure shows you how to filter and analyze a critical path. You can also use the procedure described in *Generating Custom Timing Reports with STA*, on page 729 to view this and other paths.

1. If needed, set the slack time for your design.
   - Select HDL Analyst->Set Slack Margin.
   - To view only instances with the worst-case slack time, enter a zero.
   - To set a slack margin range, type a value for the slack margin, and click OK. The software gets a range by subtracting this number from the slack time, and the Technology view displays instances within this range. For example, if your slack time is -10 ns, and you set a slack margin of 4 ns, the command displays all instances with slack times between -6 ns and -10 ns. If your slack margin is 6 ns, you see all instances with slack times between -4 ns and -10 ns.

Copyright © 2013 Synopsys, Inc.

2. Display the critical path using one of the following methods. The Technology view displays a hierarchical view that highlights the instances and nets in the most critical path of your design.
   - To generate a hierarchical view of the critical path, click the Show Critical Path icon (stopwatch icon ( ), select HDL Analyst->Technology->Hierarchical Critical Path, or select the command from the popup menu. This is a filtered view in the same window, with hierarchical logic shown in transparent instances. History commands apply, so you can return to the previous view by clicking Back.
   - To flatten the hierarchical critical path described above, right-click and select Flatten Schematic. The software generates a new view in the current window, and flattens only the transparent instances needed to show the critical path; the rest of the design remains hierarchical. Click Back to go the top-level design.
   - To generate a flattened critical path in a new window, select HDL Analyst->Technology->Flattened Critical Path. This command uses more memory because it flattens the entire design and generates a new view for the flattened critical path in a new window. Click Back in this window to go to the flattened top-level design or to return to the previous window.

3. Use the timing numbers displayed above each instance to analyze the path. If no numbers are displayed, enable HDL Analyst->Show Timing Information. Interpret the numbers as follows:

   Delay
   For combinational logic, it is the cumulative delay to the output of the instance, including the net delay of the output. For flip-flops, it is the portion of the path delay attributed to the flip-flop. The delay can be associated with either the input path or output path, whichever is worse, because the flip-flop is the end of one path and the start of another.

   Slack time
   Slack of the worst path that goes through the instance. A negative value indicates that timing has not been met.

   8.8, 1.2

4. View instances in the critical path that have less than the worst-case slack time. For additional information on handling slack times, see *Handling Negative Slack*, on page 728.

   If necessary change the slack margin and regenerate the critical path.

5. Crossprobe and check the RTL view and source code. Analyze the code and the schematic to determine how to address the problem. You can add more constraints or make code changes.

6. Click the Back icon to return to the previous view. If you flattened your design during analysis, select Unflatten Schematic to return to the top-level design.

   There is no need to regenerate the critical path, unless you flattened your design during analysis or changed the slack margin. When you flatten your design, the view is regenerated so the history commands do not apply and you must click the Critical Path icon again to see the critical path view.

7. Rerun synthesis, and check your results.

   If you have fixed the path, the window displays the next most critical path when you click the icon.

   Repeat this procedure and fix the design for the remaining critical paths. When you are within 5-10 percent of your desired results, place and route your design to see if you meet your goal. If so, you are done. If your vendor provides timing-driven place and route, you might improve your results further by adding timing constraints to place and route.

Copyright © 2013 Synopsys, Inc.

Handling Negative Slack

Positive slack time values (greater than or equal to 0 ns) are good, while negative slack time values (less than 0 ns) indicate the design has not met timing requirements. The negative slack value indicates the amount by which the timing is off because of delays in the critical paths of your design.

The following procedure shows you how to add constraints to correct negative slack values. Timing constraints can improve your design by 10 to 20 percent.

1. Display the critical path in a filtered Technology view.
   - For a hierarchical critical path, either click the Critical Path icon, select HDL Analyst->Show Critical Path, or select HDL Analyst->Technology->Hierarchical Critical Path.
   - For a flat path, select HDL Analyst->Technology->Flattened Critical Path.
2. Analyze the critical path.
   - Check the end points of the path. The start point can be a primary input or a flip-flop. The end point can be a primary output or a flip-flop.
   - Examine the instances. Use the commands described in *Expanding Pin and Net Logic*, on page 595 and *Expanding and Viewing Connections*, on page 599. For more information on filtering schematics, see *Filtering Schematics*, on page 593.
3. Determine whether there is a timing exception, like a false or multicycle path. If this is the cause of the negative slack, set the appropriate timing constraint.

   If there are fewer start points, pick a start point to add the constraint. If there are fewer end points, add the constraint to an end point.
4. If your design does not meet timing by 20 percent or more, you may need to make structural changes. You could do this by doing either of the following:
   - Enabling options like pipelining (*Pipelining*, on page 426), retiming (*Retiming*, on page 430), or resource sharing (*Sharing Resources*, on page 448).
   - Modifying the source code.
5. Rerun synthesis and check your results.

Generating Custom Timing Reports with STA

The log file generated after synthesis includes a timing report and default timing information. Use the stand-alone timing analyst (STA) when you need to generate a customized timing report (`ta`) for the following situations:

- You need more details about a specific path
- You want results for paths other than the top five timing paths (log file default)
- You want to modify constraints and analyze, without resynthesizing. See *Using Analysis Design Constraints*, on page 732 for details.

The following procedure shows you how to generate a custom report:

1. Select Analysis->Timing Analyst or click on the Timing Analyst icon( ).

   If you are working in Physical Synthesis mode, make sure that the Physical Synthesis switch is still enabled when you run stand alone timing analysis to ensure proper results.

2. Fill in the parameters.
   - You can type in the from/to or through points, or you can cut and paste or drag and drop valid objects from the Technology view (not the RTL view) into the fields. See *Timing Report Generation Parameters*, on page 291 in the *Reference Manual* for details on timing analysis parameters and how they can be filtered.
   - Set options for clock reports as needed.
   - Specify a name for the output timing report (`ta`).

Copyright © 2013 Synopsys, Inc.

3. Click Generate to run the report.

   The software generates a custom report file called *projectName*.ta, located in the implementation directory (the directory you specified for synthesis results). The software also generates a corresponding output netlist file, with an `srm` extension.

4. Analyze results.
   - View the report (Open Report) in the Text Editor. The following figure is a sample report showing analysis results based on maximum delay for the worst paths.

```
##### START OF TIMING REPORT #####[
# Timing Report written on Mon Jun 05 11:05:33 2006
#


Top view:                  sw
Requested Frequency:       10.0 MHz
Wire load mode:            top
Paths requested:           8
from:                      i:swmult2o[11]
Constraint File(s):


Worst From-To Path Information
******************************


Path information for path number 1:
    Requested Period:                       100.000
    - Setup time:                           0.245
    = Required time:                        99.755

    - Propagation time:                     2.215
    = Slack   :                             97.540

    Number of logic level(s):               12
    Starting point:                         swmult2o[11] / regout
    Ending point:                           o_o[12] / datain
    The start point is clocked by           clk_i [rising] on pin clk
    The end   point is clocked by           clk_i [rising] on pin clk

Instance / Net                               Pin        Pin                Arrival     No. of
Name                 Type                    Name       Dir     Delay      Time        Fan Out(s)
-----------------------------------------------------------------------------------------------------
swmult2o[11]         stratixii_lcell_ff      regout     Out     0.094      0.094       -
swmult2o[11]         Net                     -          -       0.311      -           2
swmult1_carry_9      stratixii_lcell_comb    dataf      In      -          0.405       -
```

– View the netlist (View Critical Path) in a Technology view. This Technology view, labeled Timing View in the title bar, shows only the paths you specified in the Timing Analyst dialog box. Note that the Timing Analyst and Show Critical Path commands (and equivalent icons and shortcuts) are disabled whenever the Timing View is active.

Copyright © 2013 Synopsys, Inc.

Using Analysis Design Constraints

Besides generating custom timing reports (see *Generating Custom Timing Reports with STA*, on page 729), you can also use the Stand-alone Timing Analyst to create constraints in an adc file. You can use these constraints to experiment with different timing values, or to add or modify timing constraints.

The advantage to using analysis design constraints (ADC) is that you do not have to resynthesize the whole design. This reduces debugging time because you can get a quick estimate, or try out different values. The Standalone Timing Analyst (STA) puts these constraints in an Analysis Design Constraints file (adc). The process for using this file is summarized in the following flow diagram:

See the following for details:

- Scenarios for Using Analysis Design Constraints, on page 733
- Creating an ADC File, on page 734
- Using Object Names Correctly in the adc File, on page 738

Scenarios for Using Analysis Design Constraints

The following describe situations where you can effectively use `adc` constraints to debug, explore options or modify constraints. For details about creating these constraints, see *Creating an ADC File*, on page 734.

- What-if analysis of design performance
  If your design meets the target frequency, you can use `adc` constraints to analyze higher target frequencies, or analyze performance of a module in a different design/technology/target device.
- Constraints on enable registers
  Similarly, you can apply syn_reference_clock on enable registers to analyze if the enables have a regular pattern like clock, or if they operate on a frequency other than clock. For example:

FDC	`create_clock {clk} -name {clk} -freq 100 -clockgroup clk_grp_0`
ADC	`define_attribute {n:en} syn_reference_clock {clk2}` `create_clock {clk2} -name {clk2} -freq 50 -clockgroup clk_grp_1`

- Adding additional timing exceptions
  When you analyze the results of the first synthesis run, you often find functional or clock-to-clock timing exceptions, and you can handle these with `adc` constraints. For example:
  - Applying false paths on synchronization circuitry
  - Adding false paths between clocks belonging to different clock groups

  You must add these constraints to see more critical paths in the design. The `adc` constraints let you add these constraints on the fly, and helps you debug designs faster.
- Modifying timing exceptions that were previously applied
  For example you might want to set a multicycle path constraint for a path that was defined as a false path in the constraint file or vice versa. To modify the timing exception, you must first ignore or reset the timing exception that was set in the constraint file, as described in *Using Analysis Design Constraints*, on page 732, step 3.

Copyright © 2013 Synopsys, Inc.

Creating an ADC File

The following procedure explains how to create an `adc` file.

1. Select File->New.
2. Do the following in the dialog box that opens:
   - Select Analysis Constraint File.

   - Type a name and location for the file. The tool automatically assigns the `adc` extension to the filename.
   - Enable Add to Project, and click OK. This opens the text editor where you can specify the new constraints.
3. Type in the constraints you want and save the file. Remember the following when you enter the constraints:
   - Keep in mind that the original `fdc` file has already been applied to the design. Any timing exception constraints in this file must not conflict with constraints that are already in effect. For example, if there is a

conflict when multiple timing exceptions (false path, path delay, and multicycle timing constraints) are applied to the same path, the tool uses this order to resolve conflicts: false path, multicycle path, max delay. See *Conflict Resolution for Timing Exceptions*, on page 513 for details about how the tool prioritizes timing exceptions.

C:/synplify_pro/test.adc *

```
1 #
2 # Delay Paths
3 #
4 set_multicycle_path  -to {i:c_5_[12]} 2
5 set_multicycle_path -to {dmantn[*]} 2
6 #
7 #
```

– The object names must be mapped object names, so use names from the Technology view, not names from the RTL view. Unlike the constraint file (RTL view), the `adc` constraints apply to the mapped database because the database is not remapped with this flow. For more information, see *Using Object Names Correctly in the adc File*, on page 738.

– If you want to modify an existing constraint for a timing exception, you must first reset the original `fdc` constraint, and then apply the new constraint. In the following example the multicycle path constraint was changed to 3:

Original FDC	set_multicycle_path –to [get_cells{a_reg*}] 2
ADC	reset_path –to {get_cells{a_reg*}] set_multicycle_path –to [get_cells{a_reg*}] 3

– When you are done, save and close the file. This adds the file to your project.

Copyright © 2013 Synopsys, Inc.

– You can create multiple `adc` files for different purposes. For example, you might want to keep timing exception constraints, I/O constraints, and clock constraints in separate files. If you have an existing `adc` file, use the Add File command to add this file to your project. Select Analysis Design Constraint Files (*.adc) as the file type.

4. Run timing analysis.

   – Select Analysis->Timing Analyst or click the Timing Analyst icon ( ). The Timing Analyst window will look like the example below, with pointers to the `srm` file, the original `fdc` and the new `adc` files you created.

– If you have multiple `adc` files, enable the ones you want.

– If you have a previous run and want to save that report, type a new name for the output `ta` file. If you do not specify a name, the tool overwrites the previous report.

– Fill in other parameters as appropriate, and click Generate.

The tool runs static timing analysis in the same implementation directory as the original implementation. The tool applies the `adc` constraints on top of the `fdc` constraints. Therefore, `adc` constraints affect timing results only if there are no conflicts with `fdc` constraints.

The tool generates a timing report called *_adc.ta and an *_adc.srm file by default. It does not change any synthesis outputs, like the output netlist or timing constraints for place and route.

Copyright © 2013 Synopsys, Inc.

5. Analyze the results in the timing report and *_adc.srm file.
6. If you need to resynthesize after analysis, add the adc constraints as an fdc file to the project and rerun synthesis.

Using Object Names Correctly in the adc File

Constraints and collections applied in the constraint file reference the RTL-level database. Synthesis optimizations such as retiming and replication can change object names during mapping because objects may be merged.

The standalone timing analyst does not map objects. It just reads the gate-level object names from the post-mapping database; this is reflected in the Technology view. Therefore, you must define objects either explicitly or with collections from the Technology view when you enter constraints into the adc file. Do not use RTL names when you create these constraints (see *Creating an ADC File*, on page 734 for details of that process).

Example

Assume that register en_reg is replicated during mapping to reduce fanout. Further, registers en_reg and en_reg_rep2 connect to register dataout[31:0]. In this case, if you define the following false path constraint in the adc file, then the standalone timing analyzer does not automatically treat paths from the replicated register en_reg_rep2 as false paths.

```
set_false_path -from {{i:en_reg}} -to {{i:dataout[31:0]}}
```

Unlike constraints in the fdc file, you must specify this replicated register explicitly or as a collection. Only then are all paths properly treated as false paths. So in this example, you must define the following constraints in the adc file:

```
set_false_path  -from {{i:en_reg}}  -to {{i:dataout[31:0]}}

set_false_path  -from {{i:en_reg_rep2}}
   -to {{i:dataout[31:0]}}
```

or

```
define_scope_collection  en_regs {find -seq {i:en_reg*}
   -filter (@name == en_reg || @name == en_reg_rep2)}

set_false_path  -from {{$en_regs}}  -to {{i:dataout[31:0]}}
```

Using Auto Constraints

You can use auto constraints in the Synplify Pro and Synplify Premier tools, however, the Physical Synthesis option must be disabled in the Synplify Premier product.

Auto constraining lets you synthesize with automatic constraints as a first step to get an idea of what you can achieve. Automatic constraints generate the fastest design implementation, so they force the timing engine to work harder. Based on the results from auto-constraining, you can refine the constraints manually later. For an explanation of how auto constraints work, see *Results of Auto Constraints*, on page 741

1. To automatically constrain your design, first do the following:
   - Set your device to a technology that supports auto-constraining. With supported technologies, the Auto Constrain button under Frequency in the Project view is available.

   - Do not define any clocks. If you define clocks using the SCOPE window or a constraint file, or set the frequency in the Project view, the software uses the user-defined create_clock constraints instead of auto constraints.
   - Make sure any multi-cycle or false path constraints are specified on registers.
2. Enable the Auto Constrain button on the left side of the Project view. Alternatively, select Project->Implementation Options->Constraints, and enable the Auto Constrain option there.

Copyright © 2013 Synopsys, Inc.

3. If you want to auto constrain I/O paths, select Project->Implementation Options->Constraints and enable Use Clock Period for Unconstrained IO.

   If you do not enable this option, the software only auto constrains flop-to-flop paths. Even when the software auto constrains the I/O paths, it does not generate these constraints for forward-annotation.

4. Synthesize the design.

   The software puts each clock in a separate clock group and adjusts the timing of each clock individually. At different points during synthesis it adjusts the clock period of each clock to be a target percentage of the current clock period, usually 15% - 25%.

   After the clocks, the timing engine constrains I/O paths by setting the default combinational path delay for each I/O path to be one clock period.

   The software writes out the generated constraints in a file called AutoConstraint_*designName*.sdc in the run directory. It also forward-annotates these constraints to the place-and-route tools.

5. Check the results in AutoConstraint_*designName*.sdc and the log file. To open the constraint file as a text file, right-click on the file in the Implementation Results view and select Open as Text.

The flop-to-flop constraints use syntax like the following:

```
create_clock -name {c:leon|clk} -period 13.327 -clockgroup
Autoconstr_clkgroup_0 -rise 0.000 -fall 6.664 -route 0.000
```

6. You can now add this generated constraint file to the project and rerun synthesis with these constraints.

Results of Auto Constraints

This section contains information about the following:

- Stages of the Auto Constrain Algorithm, on page 741
- I/O Constraints, Timing Exceptions, DLLs, DCMs, and PLLs, on page 742
- Reports and Forward-annotation, on page 742
- Repeatability of Results, on page 743

Stages of the Auto Constrain Algorithm

To auto constrain, do not define any clocks. When you enable the Auto Constrain option, the synthesis software goes through these stages:

1. It infers every clock in the design.
2. It puts each clock in its own clock group.
3. It invokes mapper optimizations in stages and generates the best possible synthesis results.
   - Clocks derived from DCM/PLLs will be in the clock group of the parent clock (DCM/PLL input clock).
   - You should only use Auto Constrain early in the synthesis process to get a general idea of how fast your design runs. This option is not meant to be a substitute for declaring clocks.
4. For each clock, including the system clock, the software maintains a negative slack of between 15 and 25 percent of the requested frequency.

Copyright © 2013 Synopsys, Inc.

I/O Constraints, Timing Exceptions, DLLs, DCMs, and PLLs

The auto constrain algorithm infers all the clocks, because none are defined. It handles the following timing situations as described below:

- I/O constraints

  You can auto constrain I/O paths as well as flop-to-flop paths by selecting Project->Implementation Options->Constraints and enabling Use Clock Period for Unconstrained IO. The software does not write out these I/O constraints.

- Timing exceptions like multicycle and false paths

  The auto constraint algorithm honors SCOPE multicycle and false path constraints that are specified as constraints on registers.

- Altera PLLs and Xilinx DCMs and DLLs

  The software infers the input frequency and derives the outputs. The inputs and outputs are put in the same clock group because they are considered to be synchronized clocks that are related to each other. Note that the auto constrain algorithm might not preserve the exact relationship between the input and outputs (2x, for example) depending on the other logic around it.

Auto Constrain Limitations

The Auto Constrain feature has the following limitations:

- Does not respect the vendor-provided maximum frequency constraints for clock generators (DCMs and DLLs).
- Over constrains designs with output critical paths.

Reports and Forward-annotation

In the log file, the software reports the Requested and Estimated Frequency or Requested and Estimated Period and the negative slack for each clock it infers. The log file contains all the details.

The software also generates a constraint file in the run directory called AutoConstraint_*designName*.sdc, which contains the auto constraints generated. The following is an example of an auto constraint file:

```
#Begin clock constraint
```

```
create_clock -name {c:leon|clk} -period 13.327 -rise 0.000 -fall
6.664

#End clock constraint
```

The software forward-annotates the create_clock constraints, writing out the appropriate file for the place-and-route tool.

Repeatability of Results

If you use the requested frequency resulting from the Auto constrain option as the requested frequency for a regular synthesis run, you might not get the same results as you did with auto constraints. This is because the software invokes the mapper optimizations in stages when it auto constrains. The results from a previous stage are used to drive the next stage. As the interim optimization results vary, there is no guarantee that the final results will stay the same.

Copyright © 2013 Synopsys, Inc.

Using the Timing Report View

Synplify Premier only; Altera and Xilinx devices

The Timing Report View displays timing reports, lets you view and query critical timing paths, and correlate your timing results. You can compare synthesis timing results with P&R Static Timing Analysis (STA) results and determine if paths for timing end points, start points, and requested periods match.

You can either view the synthesis timing results alone, or in a composite view where they are correlated with the place-and-route timing results. The following sections describe the details:

- Viewing and Analyzing the Synthesis Timing Report, on page 745
- Viewing and Analyzing the P&R Timing and Correlation Report, on page 747

Clock Name / End Instance	End Type	Slack	Start Instance	Start Type	End Clock	Start Clock	Clock Skew	Required Period
dcm|clk2x_derived_clock		3.107					0.0	5.0
moduley_inst/q[0]	reg	3.687	hold_a	reg	clka:f	dcm|clk2x_...	0.0	5.0
modulex_inst/qa[0]	reg	3.107	hold_a	reg	dcm|clk0fx...	dcm|clk2x_...	0.0	5.0
dcm|clk0fx_derived_clock		3.917					-3.279	10.0
x[0]	port	3.917	modulex...	reg	clka:r	dcm|clk0fx_...	-3.279	10.0
modulex_inst/q[7]	reg	18.273	modulex...	reg	dcm|clk0fx...	dcm|clk0fx_...	0.0	20.0
clka		4.949					-3.279	5.0
y[0]	port	18.887	moduley...	reg	clka:r	clka:f	-3.279	25.0
modulex_inst/qa[0]	reg	6.026	ena	port	dcm|clk0fx...	clka:f	3.279	5.0
modulex_inst/qa[0]	reg	36.949	a[0]	port	dcm|clk0fx...	clka:f	3.279	35.0
modulex_inst/qa[0]	reg	39.949	a[0]	port	dcm|clk0fx...	clka:r	3.279	40.0
hold_a	reg	4.949	a[0]	port	dcm|clk2x_...	clka:r	3.279	5.0
hold_a	reg	6.949	a[0]	port	dcm|clk2x_...	clka:f	3.279	5.0

For a complete description of the Timing Report View options, see *Timing Report View*, on page 285 in the *Reference Manual*.

Viewing and Analyzing the Synthesis Timing Report

The following procedure describes how to use the synthesis timing report and analyze the information in it. For information about comparing synthesis timing information to place-and-route information, see *Viewing and Analyzing the P&R Timing and Correlation Report*, on page 747.

1. Run synthesis. From an open implementation, display the Timing Report View by selecting Analysis->Timing Report View or by clicking the Timing Report view icon ( ).

   By default, the view shows the synthesis timing summary on the Synthesis Timing tab. The summary matches the critical paths shown in the Clock Relationships section of the log file after synthesis.

Clock Name / End Instance	End Type	Slack	Start Instance	Start Type	End Clock	Start Clock	Clock Skew	Required Period
dcm|clk2x_derived_clock		3.107					0.0	5.0
moduley_inst/q[0]	reg	3.687	hold_a	reg	clka:f	dcm|clk2x_...	0.0	5.0
modulex_inst/qa[0]	reg	3.107	hold_a	reg	dcm|clk0fx...	dcm|clk2x_...	0.0	5.0
dcm|clk0fx_derived_clock		3.917					-3.279	10.0
x[0]	port	3.917	modulex...	reg	clka:r	dcm|clk0fx_...	-3.279	10.0
modulex_inst/q[7]	reg	18.273	modulex...	reg	dcm|clk0fx...	dcm|clk0fx_...	0.0	20.0
clka		4.949					-3.279	5.0
y[0]	port	18.887	moduley...	reg	clka:r	clka:f	-3.279	25.0
modulex_inst/qa[0]	reg	6.026	ena	port	dcm|clk0fx...	clka:f	3.279	5.0
modulex_inst/qa[0]	reg	36.949	a[0]	port	dcm|clk0fx...	clka:f	3.279	35.0
modulex_inst/qa[0]	reg	39.949	a[0]	port	dcm|clk0fx...	clka:r	3.279	40.0
hold_a	reg	4.949	a[0]	port	dcm|clk2x_...	clka:r	3.279	5.0
hold_a	reg	6.949	a[0]	port	dcm|clk2x_...	clka:f	3.279	5.0

2. Set reporting and display options.
   - For Xilinx devices, select the kind of timing report summary to display by enabling the Logic Synthesis or Physical Plus Synthesis buttons. For Altera designs you can only display a logic synthesis report.
   - To expand all paths for the clocks and instances, click Expand All.
   - To collapse all paths for the clocks and instances, click Collapse All.

Copyright © 2013 Synopsys, Inc.

– To change the number of paths displayed per end point, select the path and click Options. Set the number of paths to display in the Synthesis Timing Options dialog box that opens.

3. To locate objects like clocks or instances, click Find button and type in a string to locate:

4. To view and analyze individual paths, use the following features in the view:
   – To generate a filtered HDL Analyst schematic for a path, select the path and click Synthesis Schematic.
   – To generate a timing report summary for a path, select the path and click Synthesis Report.

Viewing and Analyzing the P&R Timing and Correlation Report

The following procedure describes how to generate and analyze the information in the P&R timing report, and compare it to the results from synthesis. If you only want to view the synthesis timing results, use the procedure described in *Viewing and Analyzing the Synthesis Timing Report*, on page 745.

Copyright © 2013 Synopsys, Inc.

1. To generate the timing report, follow these steps:
   - Run synthesis and place-and-route. Make sure the P&R tool is enabled.
   - Open the implementation you want.
   - Display the Timing Report View by selecting Analysis->Timing Report View or by clicking the Timing Report view icon ( ).
   - Select the P&R Timing and Correlation tab.
2. Define the scope of the correlation results and run correlation.
   - For Xilinx devices, specify the synthesis results you want to compare by enabling either Logic Synthesis or Physical Plus Synthesis at the top of the window. The only choice for Altera designs is Logic Synthesis.
   - In the P&R Directory field, specify another implementation for the comparison. Select Other to specify the location of a project outside the current project.
   - To correlate all paths, click the All Paths button on the right.
   - To selectively correlate paths, click Selected Path and select the paths you want.

   The window displays the place-and-route timing information, correlated to the synthesis information. For example, you can compare synthesis timing results with P&R Static Timing Analysis (STA) results. The view reports the status of end points, start points, and required periods. Paths are reported against the end clock. The P&R path is shown first, with the synthesis path to be correlated below.

   A green check mark ( ) in the Correlated Status column indicates a match between synthesis timing and P&R timing results for end points, start points, and requested periods on that path. A red x ( ) indicates a mismatch. Float over the mismatched cell for a tool tip that describes the error. For a complete description of the Timing Report View options, see *Timing Report View*, on page 285 in the *Reference Manual*.

Clock Name / End Instance	End Type	Slack	Correlated Status	Start Instance	Start Type	End Clock	Start Clock	Clock Skew	Required Period
clkb		-0.361						-4.384	5.0
moduley_inst/qa[0]	reg	-0.361	✓	hold_a	reg	clkb:r	CLK0_BUF_1:r	-4.384	5.0
Synthesis	reg	1.359		hold_a	reg	clkb:r	dcm|CLK0_BUF_1_derived_clock_CLKIN1:r	-2.328	5.0
moduley_inst/qa[0]	reg	6.664	✓	c[0]	port	clkb:r	clka:r	1.514	10.0
Synthesis	reg	8.375		c[0]	port	clkb:r	clka:r	0.951	10.0
clka		-0.015						-6.024	10.0
x[2]	port	-0.015	✓	modulex_inst/q[2]	reg	clka:r	CLK0_BUF:r	-6.024	10.0
Synthesis	port	3.672		modulex_inst/q[2]	reg	clka:r	dcm|CLK0_BUF_derived_clock_CLKIN1:r	-1.524	10.0
CLK0_BUF_1		-0.809						4.219	5.0
hold_a	reg	-0.809	✓	a[0]	port	CLK0_BUF_1:r	clka:r	4.219	5.0
Synthesis	reg	5.449		a[0]	port	dcm|CLK0_BUF_1_derived_clock_CLKIN1:r	clka:r	3.279	5.0
CLK0_BUF		1.205						-1.678	5.0
moduley_inst/q[7]	reg	9.556	✓	moduley_inst/qb[3]	reg	CLK0_BUF:f	clkb:r	0.833	10.0
Synthesis	reg	9.149		moduley_inst/qb[0]	reg	dcm|CLK0_BUF_derived_clock_CLKIN1:f	clkb:r	2.328	10.0
modulex_inst/qb[4]	reg	3.687	✓	enb	port	CLK0_BUF:r	clka:r	4.219	10.0
Synthesis	reg	8.471		enb	port	dcm|CLK0_BUF_derived_clock_CLKIN1:r	clka:r	3.279	10.0
modulex_inst/qa[0]	reg	1.205	✓	hold_a	reg	CLK0_BUF:f	CLK0_BUF_1:r	-1.678	5.0
Synthesis	reg	1.107		hold_a	reg	dcm|CLK0_BUF_derived_clock_CLKIN1:f	dcm|CLK0_BUF_1_derived_clock_CLKIN1:r	0.0	5.0
modulex_inst/q[5]	reg	6.775	✓	modulex_inst/qa[1]	reg	CLK0_BUF:r	CLK0_BUF:f	-1.613	10.0
Synthesis	reg	4.574		modulex_inst/qa[0]	reg	dcm|CLK0_BUF_derived_clock_CLKIN1:r	dcm|CLK0_BUF_derived_clock_CLKIN1:f	-1.755	10.0

3. To compare synthesis and P&R clock names, click the Clock Mappings button.

   The clock correlation table links synthesis clock names to P&R clock names. The synthesis clock is a clock alias name that appears in the synthesis log file (`srr`) or timing report file (`ta`).

4. To view and analyze individual paths, use the following features in the view:
   - To generate a filtered HDL Analyst schematic for a path, select the path and click Synthesis Schematic.
   - To generate a timing report summary for a path, select the path and click Synthesis Report.
   - To generate a P&R summary for a path, select the path and click P&R Report.

Copyright © 2013 Synopsys, Inc.

```
1
2  Slack (MET) :            0.160ns
3    Source:                hold_a/Q
4                               (rising edge-triggered cell FDCE clocked by clk2x_pre  {rise@0.000ns
5    Destination:           modulex_inst/qa[0]/CE
6                               (rising edge-triggered cell FDCE clocked by clk0fx_pre  {rise@0.000ns
7    Path Group:            clk0fx_pre
8    Path Type:             Max at Slow Process Corner
9    Requirement:           2.500ns
10   Data Path Delay:       1.899ns  (logic 0.605ns (31.867%)  route 1.294ns (68.133%))
11   Logic Levels:          1  (LUT2=1)
12   Clock Path Skew:       -0.250ns (DCD - SCD + CPR)
13     Destination Clock Delay (DCD):  5.536ns
14     Source Clock Delay      (SCD):  5.931ns
15     Clock Pessimism Removal (CPR):  0.145ns
16   Clock Uncertainty:     0.192ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
17     Total System Jitter     (TSJ):  0.071ns
18     Discrete Jitter          (DJ):  0.125ns
19     Phase Error              (PE):  0.120ns
20
21     Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
22   -------------------------------------------------------------------    -------------------
23                          (clock clk2x_pre rise edge)
24                                                     0.000     0.000 r
25     AL29                                            0.000     0.000 r  clka
26     AL29                 IBUFG (Prop_ibufg_I_O)     0.734     0.734 r  clka_IBUFG/O
27     BUFGCTRL_X0Y3        BUFG (Prop_bufg_I_O)       3.343     4.077 r  CLK_BUF2/O
28     MMCME2_ADV_X0Y3      MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT1)
29                                                    -2.162     1.915 r  dcm_inst/DCM0/CLKOUT1
30     BUFGCTRL_X0Y2        BUFG (Prop_bufg_I_O)       2.162     4.077 r  CLK_BUF1/O
31     SLICE_X46Y77         net (fo=1)                 1.854     5.931 r  hold_a/C
32   -------------------------------------------------------------------    -------------------
33     SLICE_X46Y77         FDCE (Prop_fdce_C_Q)       0.308     6.239 f  hold_a/Q
34     SLICE_X46Y87         LUT2 (Prop_lut2_I1_O)      0.811     7.050 f  un1_ena/O
35     SLICE_X47Y89         net (fo=8)                 0.535     7.585 f  modulex_inst/qa[0]/CE
36     SLICE_X47Y89         FDCE (Setup_fdce_C_CE)     0.244     7.829    modulex_inst/qa[0]
```

5. Set other reporting and display options as needed.
   - Order the paths in ascending or descending order of slack values by sorting on the Slack column.
   - Change P&R timing correlation options by clicking the Options button and specifying the settings you want. For example, you can specify the number of paths to report per end point. If you do not enable any of the limiting options, the report includes paths for all these constraints. Rerun correlation by clicking All Paths or Selected Paths to view the updated results.

- To expand all paths for the clocks and instances, click Expand All.
- To collapse all paths for the clocks and instances, click Collapse All.
- To change the number of paths displayed per end point, select the path and click Options. Set the number of paths to display in the Synthesis Timing Options dialog box that opens.
- To locate objects like clocks or instances, click Find and type in the string to locate.

Copyright © 2013 Synopsys, Inc.

Analyzing Timing with Physical Analyst

You can use the Physical Analyst functionality to analyze timing. This section describes how to view and use the critical path for further physical synthesis.

- Viewing Critical Paths in Physical Analyst, on page 752
- Tracing Critical Paths Forward in Physical Analyst, on page 755
- Tracing Critical Paths Backward in Physical Analyst, on page 757

Viewing Critical Paths in Physical Analyst

The Physical Analyst tool makes it easy to find and examine critical paths and the relevant logic in the HDL Analyst schematic view. Make sure the HDL Analyst view is open, for example, by selecting HDL Analyst->Technology->Flattened View or HDL Analyst->RTL->Flattened View. The following procedure shows you how to filter and analyze a critical path.

1. To generate a view of the critical path with the Physical Analyst tool, click the Show Critical Path icon (stopwatch icon ( ) or select the command from the popup menu. To zoom in on the critical path, right-click and select Zoom Selected from the popup menu.

2. Check the Technology view.

   You can also cross probe the critical path from the flattened Technology view to the Physical Analyst view by clicking on the Show Critical Path icon ( ). Then, right-click and select Select All Schematic->Instances. Make sure the Physical Analyst view is open.

3. Check the Physical Analyst view. Critical path instances and nets should be highlighted in this view.

4. In the HDL Analyst view that is already open, click on the Filter Schematics icon ( ). Only the instances and nets belonging to the critical timing path are displayed, as shown below.

Copyright © 2013 Synopsys, Inc.

5. In the HDL Analyst view, right-click and select Expand Paths from the popup menu. Then, you can drag-and-drop this logic into a region on the device design plan (`sfp`) file for further physical synthesis.

Tracing Critical Paths Forward in Physical Analyst

The following procedure shows you how to trace a critical path forwards starting from the instance containing the critical start point.

1. Do one of the following in the Physical Analyst view:
   - Right-click and select Critical Path->Expand Path Forward from the popup menu
   - Press F3.

   The instance containing the critical path start point is displayed in green and highlighted. Move the cursor over the instance to display a tool tip that specifies its name and identifies this as the critical start point.

   You can also use the Filter Search option of the Find command to locate the Critical path start point.

2. Select Critical Path->Expand Path Forward or press F3 again.

   The next instances on the critical path and input ports that feed into the path are displayed and highlighted and shown connected to the critical path start point.

Copyright © 2013 Synopsys, Inc.

3. Repeat the previous step to continue tracing the path to the next instance in the path. Continue until you reach the end point.

The following figure shows you how the critical path is finally displayed.

Tracing Critical Paths Backward in Physical Analyst

The following procedure shows you how to trace a critical path backward. See the figures in *Tracing Critical Paths Forward in Physical Analyst*, on page 755, which also apply to this procedure.

1. Do one of the following in the Physical Analyst view:
   - Right-click and select Critical Path->Expand Path Backward from the popup menu.
   - Press Shift+F3.

Copyright © 2013 Synopsys, Inc.

The instance containing the critical path end point is displayed and highlighted. Move the cursor over the instance to display a tool tip that specifies its name and identifies this as the critical end point.

You can also use the Filter Search option of the Find command to locate the Critical path end point. The cell location of the critical path end point is displayed in red in the Physical Analyst view.

2. Use one of the methods described in the previous step to continue to trace the net to the next instance in its path.

   The next instance containing the critical path and output ports that feed into the path are displayed and highlighted and shown connected to the critical path end point.

3. Repeat the previous step until you reach the start point. See the figure in step 3 of *Tracing Critical Paths Forward in Physical Analyst*, on page 755 for an example of how the critical path is displayed.

CHAPTER 15

Optimizing for Specific Targets

This chapter covers techniques for optimizing your design for various vendors. The information in this chapter is intended to be used together with the information in Chapter 7, *Inferring High-Level Objects*.

This chapter describes the following:

- Optimizing Altera Designs, on page 760
- Optimizing Lattice Designs, on page 772
- Optimizing Lattice iCE65/iCE40 Designs, on page 786
- Optimizing Microsemi Designs, on page 789
- Optimizing Xilinx Designs, on page 794

Copyright © 2013 Synopsys, Inc.

Optimizing Altera Designs

This section includes some Altera technology-specific tips for optimizing your design. These tips are in addition to the general guidelines described in *Tips for Optimization*, on page 422. This section discusses the following topics that are specific to Altera technologies:

- Working with Altera PLLs, on page 760
- Specifying Altera I/O Locations, on page 763
- Packing I/O Cell Registers in Altera Designs, on page 763
- Specifying HardCopy and Stratix Companion Parts, on page 765
- Specifying Core Voltage in Stratix III Designs, on page 766
- Using LPMs in Simulation Flows, on page 767
- Improving Altera Physical Synthesis Performance, on page 769
- Working with Quartus II, on page 769

In addition, you can use the techniques described in these other topics, which apply to other vendors as well as Altera:

- Defining Black Boxes for Synthesis, on page 330
- Inferring RAMs, on page 369
- Inferring Shift Registers, on page 405
- Working with LPMs, on page 411
- Passing Information to the P&R Tools, on page 830
- Generating Vendor-Specific Output, on page 834

Working with Altera PLLs

The synthesis software recognizes the Altera PLL component, altpll, from the Stratix, Cyclone, and Arria GX device families. The following procedure shows you how to use this component in your designs. The procedure uses the Altera Megafunction wizard to generate structural VHDL or Verilog files for the Altera PLLs.

1. If you are using VHDL, the altpll component normally will be declared in the MegaWizard file, and you can comment out the LIBRARY and USE clauses in the file. The following shows an example of the lines to be commented out:

   ```
   LIBRARY altera_mf;
   USE altera_mf.altera_mf_components.all;
   ```

   If the component declaration in the MegaWizard file is not compatible with a particular Quartus software version, use the appropriate vhd file packaged with the Synopsys software in the corresponding lib/altera/quartus_II*nn* directory. For example, the altera_mf.vhd file for use with Quartus 10.1 is in the quartus_II101 subdirectory.

2. If you are using Verilog, no action is necessary as the mapper understands the altpll component.

   For compatibility with different Quartus versions, altera_mf.v files are packaged with the software in the lib/altera/quartus_II*nn* directory. Use the file from the directory that corresponds to the Quartus version that you are using.

3. Instantiate the altpll component in your design.

4. Add the MegaWizard Verilog or VHDL files to your project.

5. Open SCOPE and define the PLL input frequency in the SCOPE window. The synthesis software does not use the input frequency from the Altera MegaWizard software. Based on the input value you supply, the software generates the PLL outputs. All PLL outputs are assigned to the same clock group.

6. Set the target technology and the Quartus version (Implementation Options->Implementation Results), and synthesize as usual. The software uses the altpll component information and the constraints when synthesizing your design. The synthesis software forward-annotates the PLL input constraints to Quartus.

Instantiating Special Buffers as Black Boxes in Altera Designs

You can instantiate special buffers, like global buffers for clocks, sets/resets, and other heavily loaded signals, as black boxes in your design. See the Altera documentation for details about special buffers and the number of resources available for the part you are using.

Copyright © 2013 Synopsys, Inc.

1. Define a black-box module for your special buffer with the syn_black_box directive.

   See the examples below and *syn_black_box*, on page 1235in the *Reference Manual* for syntax details.

2. Use this black-box module to buffer the signals you want assigned to special buffers.

3. Synthesize the design and place-and-route as usual.

   The Altera tools accept the black box.

Verilog Example of Instantiating Special Buffers as Black Boxes

```
module global(a_out, a_in) /* synthesis syn_black_box */ ;
output a_out;
input a_in;

/* This continuous assignment is used for simulation,
   but is ignored by synthesis. */
assign a_out = a_in;
endmodule

module top(clk, pad_clk) ;
output pad_clk;
input clk;
// pad_clk is the primary input
global clk_buf(pad_clk, clk);
endmodule
```

VHDL Example of Instantiating Special Buffers as Black Boxes

```
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;

entity top is
port (clk : out std_logic;
pad_clk : in std_logic);
end top;

architecture structural of top is
-- In this example, "global" is an Altera vendor macro directly
-- instantiated in the Altera VHDL design as a black box.
```

```
component global
port(a_out : out std_logic; a_in : in std_logic) ;
end component;

-- Set the syn_black_box attribute on global to true.
attribute syn_black_box of global: component is true;

-- Declare clk, the internal global clock signal
begin
-- pad_clk is the primary input
clk_buf: global port map (clk, pad_clk);
end structural;
```

Specifying Altera I/O Locations

You can specify I/O locations in Altera designs using the syn_loc attribute. If you do not specify I/O locations, the P&R tool automatically assigns them locations.

1. If you used the QSF2SDC utility, to translate Altera QSF set_location_assignment and set_instance_assignment constraints, do nothing.

   The utility automatically assigns the syn_loc attribute to the pins with constraints.

2. To define an I/O location manually, use the following syntax:

Top-level sdc file	**define_attribute {***portName***} syn_loc {***pinNumbers***}**
Verilog	*object /** **synthesis syn_loc = "***pinNumbers***"** **/*
VHDL	**attribute syn_loc of** *object* **:** *objectType* **is "***pinNumbers***"**

Packing I/O Cell Registers in Altera Designs

You can improve input or output path timing in designs by packing registers into I/O cells with the syn_useioff attribute.

1. To pack the registers globally, set syn_useioff=1 on the top-level module or architecture. Specify the attribute in the source code, the SCOPE interface, or directly in the constraint file.

Copyright © 2013 Synopsys, Inc.

Format	Example
Verilog	`module test(d, clk, q) /* synthesis syn_useioff=1 */;`

VHDL	`architecture rtl of test is`
	`attribute syn_useioff : boolean;`
	`attribute syn_useioff of rtl : architecture is true;`

Constraint file syntax	`define_global_attribute syn_useioff 1`

2. To set the attribute locally, set syn_useioff=1 on a port.

Format	Example

Verilog	`module test(d, clk, q);`
	`input [3:0] d;`
	`input clk;`
	`output [3:0] q /* synthesis syn_useioff=1 */;`
	`reg q;`

VHDL	`entity test is`
	`port (d : in std_logic_vector (3 downto 0);`
	`clk : in std_logic;`
	`q : out std_logc_vector (3 downto 0);`
	`attribute syn_useioff : boolean;`
	`attribute syn_useioff of q : signal is true;`
	`end test;`

Constraint file syntax	`define_attribute {p:q[3:0]} syn_useioff 1`

3. Synthesize the design.

   The order of precedence used when there are conflicts for packing is registers, followed by ports, and finally global.

   The software infers I/O cells with presets or clears, and an embedded flip-flop in the I/O cell. The software packs registers with asynchronous clear pins and asynchronous preset pins for APEX20KE I/O cells.

   If syn_useioff is enabled for Arria GX and Stratix families, registers are not packed into Multiply/Accumulate (MAC) blocks.

Copyright © 2013 Synopsys, Inc.

Specifying HardCopy and Stratix Companion Parts

For Stratix II, Stratix III, and Stratix IV devices, you can specify an associated HardCopy II, HardCopy III, or HardCopy IV companion part to allow you to migrate from Stratix to HardCopy in Quartus. By default, no companion part is specified for a Stratix device family. However, for a HardCopy device family, a Stratix companion part must be specified.

You select the companion device in the Device tab of the Implementation Options dialog box as in the following example:

You can use any of the following methods to specify companion parts:

- Select the companion device in the Device tab of the Implementation Options dialog box as shown here:

- Use this Tcl command, where *partName* is the part name and number:

  set_option -part_companion *partName*

Copyright © 2013 Synopsys, Inc.

When you specify a companion part, the mapper targets the device with the least resources. For example, if your Stratix device has five memories and the companion HardCopy device has four memories, the mapper only uses four memory resources and maps the rest to logic.

Specifying Core Voltage in Stratix III Designs

For some Stratix III devices, you can specify core voltage. Do the following:

1. Click Implementation Options and do the following:
   - On the Device tab, set Technology to a Stratix III device.
   - Set Speed to -4.

   This makes the Core Voltage option available.

2. Set Core Voltage to the value you want, and click OK.

Alternatively, you can use the corresponding Tcl command: set_option -voltage *voltageValue*.

For example:

```
set_option -voltage 1.1V
set_option -voltage none
```

Using LPMs in Simulation Flows

This section describes how to use instantiated LPMs in simulation flows. For information about instantiating LPMs, see *Working with LPMs*, on page 411.

Simulation Flows

The simulation flows vary, depending on the method used to instantiate the LPMs. For information about instantiating LPMs, see *Instantiating Altera LPMs Using VHDL Prepared Components*, on page 416, *Instantiating Altera LPMs as Black Boxes*, on page 412, and *Instantiating Altera LPMs Using a Verilog Library*, on page 418. The following table summarizes the differences between the flows:

	Black Box Flow	**Verilog Library/VHDL Prepared Component Flows**
Applies to any LPM	Yes	No
Synthesis LPM timing support	No	Yes
Synthesis procedure	Many steps	Simple
RTL simulation	Complicated steps	Easy
Post-synthesis (`.vm`) simulation	Yes	No
Post-P&R (`.vo`) simulation	Yes	Yes
Software version	Any version Max+PlusII Quartus II 1.0 or earlier	Quartus II 1.1 or later

Black Box Method Simulation Flow

Use the following flow when you instantiate LPMs as Verilog or VHDL black boxes. You can use this procedure for any LPM supported by Altera.

1. Use the Altera MegaWizard Plug-In Manager to create an LPM megafunction with the same module and port names as the black-box module in your synthesis design.

Copyright © 2013 Synopsys, Inc.

2. Compile the following:
   - Test bench
   - The design (RTL, post-synthesis vm file, or the post-P&R vo file)
   - The v file you generated in the previous step
3. Compile the LPM megafunction simulation model: 220model.v or altera_mf.v.
4. For vm or vo simulation, compile the primitive simulation model. For example apex20Ke_atoms.v.
5. Simulate the design.

Library/Prepared Component Simulation Flow

Use this simulation procedure when you use a Verilog library or VHDL prepared components to instantiate the LPMs. You can use this flow for vo simulation if your design contains the supported LPMs.

1. Instantiate the LPMs.
   - For VHDL designs, use the prepared components methods described in *Instantiating Altera LPMs Using VHDL Prepared Components*, on page 416 or *Instantiating Altera LPMs as Black Boxes*, on page 412.
   - For Verilog designs, use the library methods described in *Instantiating Altera LPMs Using a Verilog Library*, on page 418 or *Instantiating Altera LPMs as Black Boxes*, on page 412.
2. Compile the test bench and design. The design can be either RTL or the post-P&R vo file.
3. Compile the LPM megafunction simulation model: 220model.v or altera_mf.v.
4. For vo simulation, compile the primitive simulation model. For example apex20Ke_atoms.v.
5. Simulate the design.

Improving Altera Physical Synthesis Performance

The Synplify Premier tool is timing-driven; optimizations depend on timing constraints and are applied until all constraints are met. Therefore, it is very important that you adequately apply timing constraints and not over-constrain the tool. This section includes guidelines for applying constraints.

- Verify the consistency of constraints between synthesis and P&R:
  - Clock constraints
  - Clock-to-clock constraints
  - IO delays
  - IO standard, drive, slew and pull-up/pull-down
  - Multi-cycle and false paths
  - Max-delay paths
  - DCM parameters
  - Register packing into IOB
  - SYN_LOC on IO pins and pad types
  - Placement constraints on instances
- Ensure that the final physical synthesis slack is negative, but no more than 10-15% of the clock constraint.

Working with Quartus II

The following procedures show you how to use the synthesis information to run Quartus II in an integrated mode with the Synopsys FPGA synthesis tool, directly from the synthesis interface, or in a standalone batch mode. Each procedure assumes that you have set the QUARTUS_ROOTDIR environment variable to point to your Quartus II installation directory. After synthesis, the Verilog netlist (`vqm`), forward annotated timing constraints and pin assignments (`tcl/scf`) are placed in the named Quartus project.

Integrated Mode

To run Quartus II in an integrated mode:

1. In the project view, click the Add P&R Implementation button to display the Add New Place & Route Job dialog box.

Copyright © 2013 Synopsys, Inc.

2. Optionally assign a P&R job name and click OK. The job is displayed in the project view under the active implementation.

3. Right click on the RTL source file and select File Options to display the File Properties dialog box.

4. In the File type field drop-down menu, select either Clearbox Verilog or Clearbox VHDL according to the RTL file type and then click OK.

5. Click the Run button; the clearbox netlist is copied to the PR_1 directory, the design is synthesized, and then placed and routed.

Synthesis Interface

To place and route interactively from the synthesis interface, select Quartus II-> Launch Quartus from the Options menu. This command opens the Quartus II GUI and automatically runs Quartus II with the project settings from the

synthesis run. You can monitor placement and routing as it progresses, see errors and warning messages, check what percentage of the job has completed, and execute other Quartus II commands.

Batch Mode

To run Quartus II in batch mode, select Quartus II->Run Background Compile from the Options menu. This command runs place and route using the default Quartus settings and the information in the *projectName*_cons.tcl and *projectName*.tcl files to set up and compile the Quartus project and to read the forward-annotated information from the prior synthesis run. Quartus log files are updated with placement, routing, and timing data as the design compiles.

Copyright © 2013 Synopsys, Inc.

Optimizing Lattice Designs

The Synplify and Synplify Pro synthesis tools include support for Lattice technologies. This section describes the following techniques for working with Lattice designs:

- Instantiating Lattice Macros, on page 772
- Using Lattice GSR Resources, on page 774
- Inferring Carry Chains in Lattice XPLD Devices, on page 775
- Inferring Lattice PIC Latches, on page 775
- Controlling I/O Insertion in Lattice Designs, on page 783
- Forward-Annotating Lattice Constraints, on page 784

For additional information about working with Lattice designs, see *Passing Information to the P&R Tools*, on page 830 and *Generating Vendor-Specific Output*, on page 834.

Instantiating Lattice Macros

You can instantiate Lattice macros that are predefined in the Lattice libraries that come with the tool, in the *installDirectory*/lib directory.

1. To use a Verilog macro library, add the appropriate library to your project, making sure that it is the first file in the source files list.

   The Verilog macro libraries are under the *installDirectory*/lib directory: Add the library appropriate to the technology and language (v or vhd) you are using

ORCA device families	*installDirectory*/lib/lucent/orca* Replace the asterisk with either 2, 3, or 4, according to the ORCA series you are using
LatticeXP device families	*installDirectory*/lib/lucent/xp *installDirectory*/lib/lucent/xp2 *installDirectory*/lib/lucent/xp3
LatticeSC/SCM device families	*installDirectory*/lib/lucent/sc *installDirectory*/lib/lucent/scm

MachXO device families	*installDirectory*/lib/lucent/machxo *installDirectory*/lib/lucent/machxo2
ECP/EC device families	*installDirectory*/lib/lucent/ec *installDirectory*/lib/lucent/ecp *installDirectory*/lib/lucent/ecp2 *nstallDirectory*/lib/lucent/ecp3
ispXPGA devices	*installDirectory*/lib/lattice/lava1
CPLD devices	*installDirectory*/lib/cpld/lattice

2. To use a VHDL macro library, add the appropriate library and use clauses to your VHDL source code at the beginning of the design units that instantiate the macros.

   You only need the VHDL macro libraries for simulation, but it is good practice to add them to the code. The library names may vary, depending on the map file name, which is often user-defined. The simulator uses the map file names to point to a library.

CPLD devices	`library lattice;` `use lattice.components.all;`

ORCA device families	Replace the asterisk with the series number (2, 3, or 4) for the Lattice ORCA Series 2, Series 3, or Series 4 macro library you are using.
	`library orca*;`
	`use orca*.orcacomp.all;`

LatticeXP device families	`library xp;`
	`use xp.components.all`
	`library xp2;`
	`use xp.components.all`
	`library xp3;`
	`use xp.components.all`

LatticeSC/SCM device families	`library sc;`
	`use sc.components.all`
	`library scm;`
	`use sc.components.all`

Copyright © 2013 Synopsys, Inc.

MachXO device families	`library machxo;`
	`use machxo.components.all`
	`library machxo2;`
	`use machxo.components.all`

ECP/EC device families	`library ec;`
	`use ec.components.all`
	`library ecp;`
	`use ecp.components.all;`
	`library ecp2;`
	`use ecp2.components.all;`
	`library ecp3;`
	`use ecp2.components.all;`

ispXPGA device families	`library lava;` `use lava.components.all;`

3. Instantiate the macros from the library as described in *Instantiating Black Boxes and I/Os in Verilog*, on page 330 and *Instantiating Black Boxes and I/Os in VHDL*, on page 332.

Using Lattice GSR Resources

The following procedure describes how to use GSR (global set/reset) resources and check resource usage. The GSR resource is a prerouted signal that connects to the reset input of every flip-flop, regardless of any other defined reset signals.

1. For the LatticeECP/ECP2/EC, LatticeXP2/XP, LatticeSC/SCM, MachXO, and ORCA families, you can control the use of GSR resources as follows:
   - To improve routability and performance, use the dedicated GSR resource. Select Project ->Implementation Options and enable the Force GSR Usage option on the Device tab.

     When you set this option, the synthesis software creates a GSR instance to access the resource. It uses the GSR resource for reset signals, instead of general routing. All registers are reset. when the GSR is activated, even if some flip-flops do not have a reset.
   - If a global set/reset does not correctly initialize the design, turn off the option. Select Project ->Implementation Options and disable the Force

GSR Usage option on the Device tab. When this option is off, the software does not use the GSR resource unless all flip-flops have resets, and all resets use the same signal.

2. To optimize area, set the Resource Sharing option, as described in *Sharing Resources*, on page 448.

3. To check resource usage, do the following:
   - Synthesize the design.
   - Select View Log and check the Resource Usage section. For ORCA families, you can compare the LUTs in the synthesis usage report to the occupied PFUs (function units) in the report generated after placement and routing. Each PFU consists of four 4-input LUTs and four registers. An occupied PFU means that least one LUT or register was used.

Inferring Carry Chains in Lattice XPLD Devices

For XPLD devices, you can control the inference of carry chains with the syn_use_carry_chain attribute. By default, all counters are implemented as carry chains when they are over 4 bits wide. To override this, set the syn_use_carry_chain attribute with a value of 0 on the registers of the counter or adder.

Inferring Lattice PIC Latches

The following procedure shows you how to control the inference of programmable I/O cells (PICs) in Lattice designs.

1. For the software to automatically infer PICs, make sure of the following:
   - The latch must be at the input port.
   - The latch must be directly driven by the input FPGA pad.
   - The design has one of the supported input control schemes: no clear or reset controls, and asynchronous clear or asynchronous resets.

Copyright © 2013 Synopsys, Inc.

After synthesis, the tool implements the following primitives for the PICs:

IF1S1D	Latches with asynchronous clear Latches with GSR used for clear
IF1S1B	Latches with asynchronous reset Latches with GSR used for reset.

See *Examples of PIC Latches*, on page 777 for examples of inferred PIC latches.

2. If you do not want to infer a PIC, set syn_keep on the input data net for the latch.

   After synthesis, the tool implements the latch as either a core latch with the LATCH primitive or as a mux, depending on the Lattice technology you selected. The following figure shows an input latch with no reset or clear implemented as a mux and a core latch in different technologies.

With syn_keep: Implemented as Mux

With syn_keep: Implemented as Core Latch

Examples of PIC Latches

The following examples show how the tool infers PICs from the code:

- Positive Level Data Latch with No Resets or Clears, on page 778
- Negative Level Data Latch with No Resets or Clears, on page 779
- Positive Level Data Latch with Asynchronous Reset, on page 780
- Positive Level Data Latch with Asynchronous Clear, on page 782

Copyright © 2013 Synopsys, Inc.

Positive Level Data Latch with No Resets or Clears

VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity inlatch is port
(clk : in std_logic;
din : in std_logic;
dout: out std_logic);
end entity inlatch;

architecture bhve of inlatch is
begin
 process(clk,din)
 begin
   if clk='1' then
    dout <= din;
   end if;
 end process;

end bhve;
```

Verilog

```
module inlatch (clk,din,dout);
input clk;  input din;  output dout;

reg dout;

always @(clk)
begin
 if(clk)
    dout <= din;
end

endmodule
```

With this code, the tool implements the IFS1S1B latch primitive in the Technology view:

Negative Level Data Latch with No Resets or Clears

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
(clk : in std_logic;
din : in std_logic;
dout: out std_logic);
end entity inlatch;
architecture bhve of inlatch is
begin
  process(clk,din)
  begin
    if clk='0' then
      dout <= din;
    end if;
  end process;
end bhve;
```

Verilog

```
module inlatch (clk,din,dout);
input clk;  input din;  output dout;
reg dout;
always @(clk)
begin
  if(!clk)
     dout <= din;
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

With this code, the tool implements the IFS1S1B latch primitive in the Technology view:

Positive Level Data Latch with Asynchronous Reset

The tool imfers the IFS1S1B latch primitive in the Technology view with the code shown below:

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
(clk : in std_logic; aset: in std_logic;
din : in std_logic; dout: out std_logic);
end entity inlatch;
architecture bhve of inlatch is
begin
 process(clk,din,aset)
 begin
   if aset ='1' then
     dout <='1';
   elsif clk='1' then
     dout <= din;
   end if;
 end process;
end bhve;
```

Verilog

```
module inlatch(clk,din,aset,dout);
input  clk; input  din; input aset; output dout;
reg  dout;
always @(clk or aset)
begin
 if(aset)
   dout <= 1'b1;
 else if (clk)
   dout <= din;
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

Positive Level Data Latch with Asynchronous Clear

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity inlatch is port
(clk : in std_logic; aclr: in std_logic;
din : in std_logic; dout: out std_logic); }
end entity inlatch;
architecture bhve of inlatch is
begin
 process(clk,din,aclr)
 begin
   if aclr ='1' then
     dout <='0';
   elsif clk='1' then
     dout <= din;
   end if;
end process;
end bhve;
```

Verilog

```
module inlatch(clk,din,aclr,dout);
input  clk; input  din; input aclr; output dout;
reg  dout;
always @(clk or aclr)
begin
  if(aclr)
    dout <= 1'b0;
  else if (clk)
    dout <= din;
end
endmodule
```

With this code, the tool infers the IFS1S1D primitive in the Technology view:

Controlling I/O Insertion in Lattice Designs

You can control I/O insertion globally, or on a port-by-port basis.

1. To control the insertion of I/O pads at the top level of the design, use the Disable I/O Insertion option as follows:
   - Select Project->Implementation Options and click the Device panel.
   - If you do not want to insert any I/O pads in the design, enable Disable I/O Insertion

     Do this if you want to check the area your blocks of logic take up, before you synthesize an entire FPGA. If you disable automatic I/O insertion, you do not get *any* I/O pads in your design, unless you manually instantiate them.
   - If you want to insert I/O pads, disable the Disable I/O Insertion option.

     When this option is set, the software inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist. Once inserted, you can override the I/O pad inserted by directly instantiating another I/O pad.
2. To force I/O pads to be inserted for input ports that do not drive logic, follow the steps below.
   - To force I/O pad insertion at the module level, set the syn_force_pads attribute on the module. Set the attribute value to 1. To disable I/O pad insertion at the module level, set the syn_force_pads attribute for the module to 0.

Copyright © 2013 Synopsys, Inc.

– To force I/O pad insertion on an individual port, set the syn_force_pads attribute on the port with a value to 1. To disable I/O insertion for a port, set the attribute on the port with a value of 0.

Enable this attribute to preserve user-instantiated pads, insert pads on unconnected ports, insert bi-directional pads on bi-directional ports instead of converting them to input ports, or insert output pads on unconnected outputs.

If you do not set the syn_force_pads attribute, the synthesis design optimizes any unconnected I/O buffers away.

Forward-Annotating Lattice Constraints

You can forward-annotate multicycle and false path constraints to Lattice place-and-route tool by following the procedure below. For additional information about forward-annotation, see *Generating Constraint Files for Forward Annotation*, on page 125.

1. To forward-annotate a from, to, or through multicycle constraint, open the SCOPE spreadsheet and do either of the following:
   – Click the Multi-Cycle Paths tab. Depending on the type of constraint you want to set, select or type the instance name under the To, From or Through column. Next, set the number of clock cycles under the Cycles column.

     When you set this constraint, the software runs timing-driven synthesis and then forward-annotates the constraint.
   – Click the Other tab. In the Command column, type define_multicycle_path. In the Arguments column, type -from and the source port or register name, and -to and the destination port or register name. For example: -from in0_int -to output 2.

     When you set this constraint from the Other tab, the software forward-annotates the constraint, but does not run timing-driven synthesis using this constraint.
2. To forward-annotate a false path constraint, open the SCOPE spreadsheet and do either of the following:
   – Click the False Paths panel. Depending on the type of constraint you want to set, select or type the instance name under the To, From or

Through column. When you set this constraint, the software runs timing-driven synthesis and then forward-annotates the constraint.

– Click the Other tab. In the Command column, type define_false_path. In the Arguments column, type -from and the source port or register name, and -to and the destination port or register name. For example:

Command	Arguments
define_false_path	-from in1_int -to output
define_false_path	-from in* -to out*

When you set this constraint from the Other tab, the software forward-annotates the constraint, but does not run timing-driven synthesis using this constraint.

3. Select Project->Implementation Options and enable the Write Vendor Constraint File option on the Implementation Results tab.
4. Run your design. The synthesis tool creates the $DESIGN_synplify.lpf file in the same directory as your results files.
5. Start the Lattice ispLEVER place-and-route tool and run the Map stage (after importing the $DESIGN_synplify.lpf file).
6. Run the PAR and BIT stages in ispLEVER.

Copyright © 2013 Synopsys, Inc.

Optimizing Lattice iCE65/iCE40 Designs

This section describes the following topics when working with Lattice iCE65/iCE40 designs:

- Using Sequential Logic
- Using Combinational Logic
- Handling Tristates
- Handling I/Os and Buffers

Using Sequential Logic

The synthesis software infers or instantiates the following Lattice iCE65/iCE40 sequential logic depending on the control signal.

Primitive	Description
SB_DFF	D flip-flop
SB_DFFE	D flip-flop with clock enable
SB_DFFSR	D flip-flop with synchronous reset
SB_DFFR	D flip-flop with asynchronous reset
SB_DFFSS	D flip-flop with synchronous set
SB_DFFS	D flip-flop with asynchronous set
SB_DFFESR	D flip-flop with clock enable and synchronous reset
SB_DFFER	D flip-flop with clock enable and asynchronous reset
SB_DFFESS	D flip-flop with clock enable and synchronous set
SB_DFFES	D flip-flop with clock enable and asynchronous set
SB_DFFN	D flip-flop - negative edge clock
SB_DFFNE	D flip-flop - negative edge clock and clock enable
SB_DFFNSR	D flip-flop - negative edge clock with synchronous reset
SB_DFFNR	D flip-flop - negative edge clock with asynchronous reset
SB_DFFSS	D flip-flop - negative edge clock with synchronous set

Copyright © 2013 Synopsys, Inc.

Primitive	Description
SB_DFFNS	D flip-flop - negative edge clock with asynchronous set
SB_DFFNESR	D flip-flop - negative edge clock, enable and synchronous reset
SB_DFFNER	D flip-flop - negative edge clock, enable and asynchronous reset
SB_DFFNESS	D flip-flop - negative edge clock, enable and synchronous set
SB_DFFNES	D flip-flop - negative edge clock, enable and asynchronous set

Limitations

The synthesis software does not support initial values on flip-flops for the Lattice iCE65/iCE40 technology. The Power ON state for Lattice iCE65/iCE40 flip-flops is 0.

Using Combinational Logic

The synthesis software handles combinational logic as follows:

- This logic is mapped to the 4-input LUT (SB_LUT4).
- All data path operators (adders, subtractors, comparators, or multipliers) is mapped to carry chain logic (SB_CARRY) and additional LUT4s (SB_LUT4).

Handling Tristates

The synthesis software handles tristates as follows:

- Internal tristates are not supported and are converted to MUXes whenever possible. An error is generated if an internal tristate cannot be converted to a MUX.
- Tristates connected to an output port are absorbed into the I/O pad.

Copyright © 2013 Synopsys, Inc.

Handling I/Os and Buffers

The synthesis software handles I/Os and buffers as follows:

- The SB_IO primitive is used for inserting I/O pads.
- Since DDR inferencing is not supported, only input, output, and enable registers are packed into the I/O pads when available.
- The synthesis software does not infer SB_IO_DS, but allows for its instantiation.
- SB_GB is used for global buffers, such as clocks.
- SB_GB_IO is used for external clocks and SB_GB is used for internal clocks.
- Instantiated buffers are retained as is.
- Undriven pins in instantiated instances are left floating as it is in the RTL. Input pins in the RTL should not be connected to a floating net.
- The SB_WARMBOOT primitive can be instantiated and is treated as a black box.

Copyright © 2013 Synopsys, Inc.

Optimizing Microsemi Designs

The Synplify and Synplify Pro synthesis tools support Microsemi designs. The following procedures Microsemi-specific design tips.

- Using Predefined Microsemi Black Boxes, on page 789
- Using Smartgen Macros, on page 790
- Working with Radhard Designs, on page 790
- Specifying syn_radhardlevel in the Source Code, on page 791

For additional Microsemi-specific information, see *Passing Information to the P&R Tools*, on page 830 and *Generating Vendor-Specific Output*, on page 834.

Using Predefined Microsemi Black Boxes

The Microsemi macro libraries contain predefined black boxes for Microsemi macros so that you can manually instantiate them in your design. For information about using ACTGen macros, see *Using Smartgen Macros*, on page 790. For general information about working with black boxes, see *Defining Black Boxes for Synthesis*, on page 330.

To instantiate an Microsemi macro, use the following procedure.

1. Locate the Microsemi macro library file appropriate to your technology and language (v or vhd) in one of these subdirectories under *installDirectory*/lib.

proasic	ProASIC (PA), ProASIC^{PLUS}, ProASIC3/3E/3L, Fusion/SmartFusion, and IGLOO/IGLOO+/IGLOOe macros
microsemi	Macros for all other Microsemi technologies.

   Use the macro file that corresponds to your target architecture. If you are targeting the 1200XL architecture, use the act2.v or act2.vhd macro library.

2. Add the Microsemi macro library *at the top* of the source file list for your synthesis project. Make sure that the library file is first in the list.

Copyright © 2013 Synopsys, Inc.

3. For VHDL, also add the appropriate library and use clauses to the top of the files that instantiate the macros:

   library *family*;
   use *family*.**components.all** ;

   Specify the appropriate technology in *family*; for example, act3.

Using Smartgen Macros

The Smartgen macros replace the ACTgen macros, which were available in the previous Designer 6.x place-and-route tool. The following procedure shows you how to include Smartgen macros in your design. For information about using Microsemi macro libraries, see *Using Predefined Microsemi Black Boxes*, on page 789. For general information about working with black boxes, see *Defining Black Boxes for Synthesis*, on page 330.

1. In Smartgen, generate the function you want to include.
2. For Verilog macros, do the following:
   - Include the appropriate Microsemi macro library file for your target architecture in your the source files list for your project.
   - Include the Verilog version of the Smartgen result in your source file list. Make sure that the Microsemi macro library is first in the source files list, followed by the Smartgen Verilog files, followed by the other source files.
3. Synthesize your design as usual.

Working with Radhard Designs

The following procedure outlines how to specify radhard values for a design with the syn_radhardlevel attribute. Remember that the attribute is not recursive. It only applies to all registers at the level where it is set and does not affect lower-level registers.

You specify radhard values in modules and architecture in both the Attributes panel in SCOPE and in the source code. However, for registers, it must be specified in the source code only.

1. Add to your project the Microsemi macro files appropriate to the radhard values you plan to set in the design. The macro files are in *installDirectory*/lib/microsemi:

Radhard Value	Verilog Macro File	VHDL Macro File
cc	cc.v	cc.vhd
tmr	tmr.v	tmr.vhd
tmr_cc	tmr_cc.v	tmr_cc.vhd

   For ProASIC3/3E devices only, you do not need to add the Microsemi macro file to your project.

2. To set a global or default syn_radhardlevel attribute, do the following:
   – Set the value in the source file for the module. The following sets all registers of module_b to tmr:

VHDL

```
library synplify;
use synplify.attributes.all;
attribute syn_radhardlevel of
  behav: architecture is "tmr";
```

Verilog

```
module module_b (a, b, sub,
  clk, rst) /*synthesis
  syn_radhardlevel="tmr"*/;
```

   – Make sure that the corresponding Microsemi macro file from step 1 is the first file listed in the project, if required.

Specifying syn_radhardlevel in the Source Code

For a module, you can attach the syn_radhardlevel attribute either in the Attributes panel of the SCOPE window or in the source code. For a register, you can only apply this attribute in the source code.

To set attributes in SCOPE, see *How Attributes and Directives are Specified*, on page 1166. The following procedure outlines how to set this attribute in the source code.

1. To set a global or default value, make sure that the corresponding Microsemi macro file is the first file listed in the project, if required.

Copyright © 2013 Synopsys, Inc.

2. To set a syn_radhardlevel value for all the registers of a module, do the following:
   - Set the value in the source file. The following sets all registers of module_b to tmr:

VHDL	Verilog

`library synplify;`
`use synplify.attributes.all;`
`attribute syn_radhardlevel of`
`  behav: architecture is "tmr";`
	`module module_b (a, b, sub,`
	`clk, rst) /*synthesis`
	`syn_radhardlevel="tmr"*/;`

   - Add the appropriate Microsemi macro file (tmr.v or tmr.vhd) to the project, unless you are working with a ProASIC3, ProASIC3E, or ProASIC3L target. You do not need to add the Microsemi macro file to your project for these devices. The macro files are in the *installDirectory*/lib/microsemi.

   The attribute is not recursive. When used at the module or architecture level, it only applies to the registers at that level, and does not affect lower-level registers.

3. To set a syn_radhardlevel value on a per-register basis, do the following:
   - Set the value on the register in the source file for the module. For example, to set the value of register bl_int to tmr, enter the following in the module source file:

VHDL	Verilog

`library synplify;`
`use synplify.attributes.all;`
`attribute syn_radhardlevel of`
`  bl_int: signal is "tmr"`
	`reg [15:0] a1_int, b1_int`
	`/* synthesis syn_radhardlevel =`
	`"tmr" */;`

   - Add the appropriate Microsemi macro file (tmr.v or tmr.vhd for this example) to the project, unless you are working with a ProASIC3, ProASIC3E, or ProASIC3L target. You do not need to add the Microsemi macro file to your project for these devices.

Use a register-level attribute to override a default value with another value, or set it to none to ensure that a global default value is not applied to the register.

4. To prevent a default from being applied to a register or module/entity, set syn_radhardlevel to none for that register, module, or entity.

Copyright © 2013 Synopsys, Inc.

Optimizing Xilinx Designs

This section contains tips for working with Xilinx designs:

- Designing for Xilinx Architectures, on page 794
- Specifying Xilinx Macros, on page 795
- Specifying Global Sets/Resets and Startup Blocks, on page 797
- Inferring Wide Adders, on page 798
- Instantiating CoreGen Cores, on page 801
- Packing Registers for Xilinx I/Os, on page 804
- Specifying Xilinx Register INIT Values, on page 807
- Specifying RLOCs, on page 817
- Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute, on page 819
- Using Clock Buffers in Virtex Designs, on page 820
- Working with Clock Skews in Xilinx Virtex-5 Physical Designs, on page 822
- Reoptimizing with EDIF Files, on page 824
- Improving Xilinx Physical Synthesis Performance, on page 825
- Running Post-Synthesis Simulation, on page 826
- Instantiating Special I/O Standard Buffers for Virtex, on page 823

For additional Xilinx-specific techniques, see *Xilinx Partition Flow*, on page 873, Xilinx Partition Flow for Versions Before ISE 12.1, on page 878, *Working with Gated Clocks*, on page 466, *Inferring RAMs*, on page 369, and *Inferring Shift Registers*, on page 405. Note that some of these features are not available in the Synplify product.

Designing for Xilinx Architectures

The tips listed here are in addition to the technology-independent design tips described in *Tips for Optimization*, on page 422.

- For critical paths, attach the xc_fast attribute to the I/Os.

- To ensure that frequency constraints from register to output pads are forward annotated to the P&R tools, add default input_delay and output_delay constraints of 0.0 in the synthesis tool. The synthesis tool forward-annotates the frequency constraints as PERIOD constraints (register-to-register) and OFFSET constraints (input-to-register and register-to-output). The place-and-route tools use these constraints.
- Run successive place-and-route iterations with progressively tighter timing constraints to get the best results possible.
- When using VHDL, specify a UNISIM library using the following syntax:

  ```
  library unisim;
  use unisim.vcomponents.all;
  ```

  Remove any other package files with user-defined UNISIM primitives.

  Note: If you are using ISE11 for a VHDL design targeting Virtex 2 or Virtex 2 Pro device, you must manually add the unisim_m10i.vhd library before synthesizing the design, because ISE11 does not support these Virtex families. For Verilog designs, the tool automatically add the corresponding unisim_m10i.v file so you do not need to do it manually.

Specifying Xilinx Macros

The synthesis tool provides Xilinx macro libraries that you can use to instantiate components like I/Os, I/O pads, gates, counters, and flip-flops. Using the macros from these libraries allows you to perform a subsequent simulation run without changing your code.

1. To use the Verilog macro library, review the unisim.v macro library in the *installDirectory*/lib/xilinx directory for the available macros. The unisim.v file is automatically added to your project file when you select a Xilinx target device.
2. To use a VHDL library, do the following:
   - Review the unisim.vhd macro library in the *installDirectory*/lib/xilinx directory to check the macros that are available.
   - Add the corresponding library and use clauses to the beginning of the design units that instantiate the macros, as in the following example:

     ```
     library unisim;
     use unisim.vcomponents.all;
     ```


Copyright © 2013 Synopsys, Inc.

You do not need to add the macro library files to your the source files for your project.

3. Instantiate the macro component in your design.
4. To instantiate an I/O pad with different I/O standards, do the following:
   - Specify the macro library as described in the first two steps.
   - Instantiate the I/O pad component in your design. You can instantiate IBUF, IBUFG, OBUF, OBUFT, and IOBUF components.
   - In the source files, define the generic or parameter values for the I/O standard. Use an IOSTANDARD generic/parameter to specify the I/O standard you want. Refer to the Xilinx documentation for a list of supported IOSTANDARDs. For certain pad types, you can also specify the output slew rate (SLEW) and output drive strength (DRIVE). See *OBUF Instantiation Example*, on page 796 for an example.

OBUF Instantiation Example

The following examples show the declaration of OBUF in macro library files:

VHDL

```
component OBUF
  generic (
     IOSTANDARD : string := "default";
     SLEW : string := "SLOW";
     DRIVE : integer := 12
  );
  port (
     O : out std_logic;
     I : in std_logic;
  );
end component;
attribute syn_black_box of OBUF : component is true
```

Verilog

```
module OBUF(O, I); /* synthesis syn_black_box */
parameter IOSTANDARD="default";
parameter SLEW="SLOW";
parameter DRIVE=12;
output O;
input I;
endmodule
```

To use the macro libraries to instantiate I/O pad types, define the generic/parameter values in the Verilog or VHDL source files. The following examples show how to instantiate OBUF pads with an I/O standard value of LVCMOS2, an output slew value of FAST, and an output drive strength of 24.

VHDL

```
Data : OBUF
   generic map (
      IOSTANDARD => "LVCMOS2",
      SLEW => "FAST",
      DRIVE => 24
   )
   port map (
      O => o1,
      I => i1
   );
```

Verilog

```
OBUF Data(.O(o1), .I(i1));
defparam Data.IOSTANDARD = "LVCMOS2";
defparam Data.SLEW = "FAST";
defparam Data.DRIVE = 24;
```

The resulting EDIF file contains the following, which corresponds to the instantiations:

```
(instance (
   rename dataZ0 "data")
   (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
   (property iostandard (string "LVCMOS2"))
   (property slew (string "FAST"))
   (property drive (integer 24) )
)
```

Specifying Global Sets/Resets and Startup Blocks

The global set/reset (GSR) resource is a pre-routed signal that goes to the set or reset input of each flip-flop in your design. Using this resource instead of general routing for a set or reset signal can have a significant positive impact on the routability and performance of your design. The synthesis tools infer this resource automatically in most cases, but you can also specify access to the GSR resource with a Xilinx startup block.

Copyright © 2013 Synopsys, Inc.

1. Specify access to the GSR as follows:
   - To automatically use the GSR if needed, select Implementation Options -> Device, and set Force GSR Usage to auto. With this setting, the tool automatically determines if it needs to use the GSR.
   - To use the GSR, set Force GSR Usage to yes.
   - If you do not want to use the GSR, set Force GSR Usage to no.
2. For Xilinx XC designs, specify global sets/resets (GSR) as follows:
   - For a design with a single GSR, the synthesis tools automatically connect it to a startup block, even if the flip-flops have no sets/resets specified. If you need to change this setting, select Project-> Implementation Options ->Device, and set Force GSR Usage to no. With this setting, all flip-flops must have a set or reset and the set or reset must be the same before GSR is used.
   - For designs with multiple GSRs, the synthesis tool does not automatically create a startup block for GSR. If you still want to use one of the set or reset signals for GSR, you must instantiate a STARTUP_GSR component manually, as described in the next step.

   For XC4000 technologies, the synthesis tool forces the creation of a startup block to access the GSR resource, if it is appropriate for your design.
3. To instantiate a start-up block manually, do the following:
   - Go to the *installDirectory*/lib/xilinx directory and locate the appropriate Xilinx startup blocks in either the Verilog (unisim.v) or VHDL (unisim.vhd) format.
   - Instantiate the startup block component in your design. Where there is more than one component listed, you can use them independently, because the blocks are merged to form a single block in the EDIF file.

Inferring Wide Adders

For Virtex-5 and Virtex-6 designs, you can map wide adder/subtractor structures to DSP48Es. Xilinx architectures let you use cascading DSP48Es and the CARRYCASCOUT pin to support a structure with up to three pipeline registers with different synchronous control signals. It supports two or three signed/unsigned inputs (with carry/borrow).

The following shows the implementation of wide adders with one pipelined register and no pipelined registers as DSP48Es:

To automatically map to DSP48Es in the synthesis tools, do the following:

1. Make sure the structure you want to map conforms with these rules:
   - The adder/subtractor does not have more than 96 bits.
   - All registers share the same control signals (enables, clocks, reset). Registers with different control signals are mapped to the DSP48E, but they are kept outside the DSP48E.
   - The adder does not have a 48-bit input and a 49-bit output.
2. Set syn_dspstyle to dsp48.

   You must set this attribute, or the tool does not infer a DSP48E. See *syn_dspstyle*, on page 1264 for the syntax for this attribute.
3. Synthesize the design.

   If your structure has less than three pipelined registers, you see an advisory message in the log file, because three pipelined registers give the best performance.

Copyright © 2013 Synopsys, Inc.

The following is an example of how the synthesis tool maps an adder-> register->register structure with 96-bit signed input and output to a DSP48E:

Instantiating CoreGen Cores

Predesigned IP cores save on design effort and improve performance. The process for handling IP cores is slightly different for CoreGen and Virtex PCI cores. The following procedure describes how to instantiate a CoreGen module. For Virtex PCI cores, see *Instantiating Virtex PCI Cores*, on page 802.

1. Use the Xilinx CORE generator to create structural EDIF netlists and generate timing and resource usage information for synthesis.
   - For legacy cores, generate a single flat `edf` netlist file.
   - For newer cores, generate a top-level flat `edn` or `edf` netlist file that instantiates `ndf` files for each hierarchical level in the design.
2. Open the synthesis software, and add the generated files (`edf` only for legacy cores; `edn` or `edf` and `ndf` for newer cores) to your project.
3. Define the core as a black box by adding the syn_black_box attribute to the module definition line, or by using the Coregen `v` file. The following is an example of the attribute:

```
module ram64x8(din, addr, we, clk, dout)/* synthesis syn_black_box
*/;
   input[7:0] din;
   input [5:0] addr;
   input we, clk;
   output [7:0] dout;
endmodule;
```

4. Make sure the bus format matches the bus format in the core generator, using the syn_edif_bit_format and syn_edif_scalar_format directives if needed.

```
module ram64x8(din, addr, we, clk, dout)
   /* synthesis syn_black_box syn_edif_bit_format = "%u<%i>"
   syn_edif_scalar format ="%u" */;
```

5. Instantiate the black box in the module or architecture.

```
ram64x8 r1(din, addr, we, clk, dout);
```

6. Synthesize the design.

   If you supplied structural EDIF netlists, the software optimizes the design based on the information in the structural netlists. The generated reports contain the optimization information.

Copyright © 2013 Synopsys, Inc.

Instantiating Virtex PCI Cores

For Virtex PCI cores, you can use either a top-down or bottom-up methodology. This figure shows a design that is used in the explanations of both methodologies, below.

Bottom-Up Method

The bottom-up method synthesizes lower-level modules first. The synthesized modules are then treated as black boxes and synthesized at the next level. The following procedure refers to the figure shown above.

1. Synthesize the user-defined application (PING64) by itself.
   - Make sure that the Disable I/O Insertion option is on.
   - Specify the syn_edif_bit_format = "%u<%i>" and syn_edif_scalar_format = "%u" attributes. These attributes ensure that the EDIF bus names

match the Xilinx upper-case, angle bracket style bus names and the Xilinx upper-case net names, respectively.

The software generates an EDIF file for this module.

2. Synthesize the top-level module that contains the PCI core, with the Disable I/O Insertion option enabled and the EDIF naming attributes described in the previous step. Use the following files to synthesize:
   - The top-level module (PCIM_LC) file, with the PCI core (PCI_LC_I) declared as a black box with the syn_black_box attribute.
   - A black box file for the core (PCI_LC_I), that only contains information about the PCI core ports. This file is the source file that is generated for simulation, not the ngo file.
   - The appropriate synthesis Virtex file (*installDirectory*/lib/xilinx) that contains module definitions of the I/O pads in the top-level module, PCIM_LC.

   The software generates an EDIF file for this module.

3. Synthesize the top level (PCIM_TOP) with Disable I/O Insertion off. Use the following files:
   - The source file for CFG.
   - A black box file for PING64.
   - A black box file for PCIM_LC.
   - A top-level file that contains black box declarations for PING64 and PCIM_LC.

   The software generates an EDIF file for the top level.

4. Place and route using the Xilinx ngo file for the core, and the three EDIF files generated from synthesis: one for each of the modules PING64 and PCIM_LC, and the top-level EDIF file. Select the top-level EDIF file when you run place-and-route.

Top-down Methodology

The top-down method instantiates user application blocks and synthesizes all the source files in one synthesis run. This method can result in a smaller, faster design than with the bottom-up method, because the tool can do cross-boundary optimizations. The following procedure refers to the design shown in the previous figure.

Copyright © 2013 Synopsys, Inc.

1. Create your own configuration file for your application model (CFG).
2. Edit the top-level source file to do the following:
   - Instantiate your application block (PING64) in the top-level source file.
   - Add the ports from your application.
3. Add the appropriate synthesis Virtex file (*installDirectory*/lib/xilinx) to the project. This file contains module definitions of the I/O pads in the PCIM_LC module.
4. Specify the top-level file in the project.
5. Synthesize your design with the following files:
   - Virtex module definition file (previous step)
   - Source files for top-level design, user application (PING64), PCIM_LC, and CFG
   - Simulation wrapper file for PCI core

   The software generates an EDIF file for the top level.
6. Place and route the design using the top-level EDIF file from synthesis and the Xilinx ngo file for the PCI core.

Packing Registers for Xilinx I/Os

When a register drives an input or output, you might want to pack it in an IOB instead of a CLB, as in these cases:

- The chip interfaces with another, and you have to minimize the register-to-output or input-to-register delay.
- You have limited CLB resources, and packing the registers in an IOB can free up some resources.

To pack registers in an IOB, you set the syn_useioff attribute.

1. To globally embed all the registers into IOBs, attach the syn_useioff attribute to the module in one of these ways:
   - Add the attribute in the SCOPE window, attaching it to the module, architecture, or the top level. Check the Enable box, set the Attribute column to syn_useioff, the Object column to <global>, and the attribute value to 1. The constraint file syntax looks like this:

```
define_global_attribute syn_useioff 1
```

– To add the attribute in the Verilog source code, add this syntax to the top level:

```
module global_test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

– To add the attribute in the VHDL source code, add this syntax to the top level architecture declaration:

```
architecture rtl of global_test is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```

For details about attaching attributes using the SCOPE interface and in the source code, see *Specifying Attributes and Directives*, on page 245.

When set globally, all boundary registers and (OE) registers associated with the data registers are marked with the Xilinx IOB property. This property is forward annotated in the EDIF netlist and used by the Xilinx place-and-route tools to determine how the registers are packed. All marked registers are packed in the corresponding IOBs.

2. To apply syn_useioff to individual registers or ports, use one of these methods:

   – Add the attribute in the SCOPE window, attaching it to the ports you want to pack, and set the attribute value to 1. The resulting constraint file syntax looks like this:

```
define_attribute {p:q[3:0]} syn_useioff 1
```

   – To add the attribute in the Verilog source code, add this syntax:

```
module test is (d, clk, q);
input [3:0] d;
input clk;
output [3:0] q /* synthesis syn_useioff = 1 */;
reg q;
```


Copyright © 2013 Synopsys, Inc.

– To add the attribute in the VHDL source code, add syntax as shown inside the entity for the local port:

```
entity test is
   port (d : in std_logic_vector(3 downto 0);
      clk : in std_logic
      q : out std_logic_vector(3 downto 0);
attribute syn_useioff : boolean;
attribute syn_useioff of q : signal is true;
end test;
```

The software attaches the IOB property as described in the previous step, but only to the specified flip-flops. Packing for ports and registers without the attribute is determined by timing preferences. If a register is to be packed into an IOB, the IOB property is attached and forward annotated. If it is to be packed into a CLB, the IOB property is not forward annotated.

In Virtex designs where the synthesis software duplicates OE registers, setting the syn_useioff attribute on a boundary register only enables the associated OE register for packing. The duplicate is not packed, but placed in a CLB. The packed registers are used for data path, and the CLB registers are used for counter implementation.

In Virtex designs where a shift register is at a boundary edge and the syn_useioff attribute is enabled, the software extracts only the initial or final SRL16 shift register from the LUT for packing. The shift register that is implemented in the technology view is smaller because of the extraction.

3. If you set multiple syn_useioff attributes at different levels of the design, the tool uses the most specific setting (highest priority).

   This table summarizes syn_useioff priority settings, from the highest priority (register) to the lowest (global):

I/O Type	syn_useioff Value	Description
Register	1	Packs registers into the I/O pad cells, overriding port or global specifications.
	0	Does not pack registers into I/O pad cells, overriding port or global specifications.

I/O Type	syn_useioff Value	Description
Port	1	Packs registers into the I/O pad cells, overriding any global specification.
	0	Does not pack registers into I/O pad cells, overriding any global specification.
Global	1	Packs registers into the I/O pad cells.
	0	Does not pack registers into I/O pad cells.

Specifying Xilinx Register INIT Values

You can specify initial values for registers so that the RTL, gate-level simulation, and the final implementation results match. You can specify INIT values for registers either with the HDL initialization specification built into Verilog or VHDL, or by adding the synthesis attribute. You can then pass the values to the Xilinx P&R tools.

Both methods are described in the following procedure, but the HDL specification method is recommended.

1. To ensure that the register is not optimized away during synthesis, set the syn_preserve directive on the register in the source code.

   Use this directive even if you define the INIT values with a constraint in the `sdc` file. If you do not have this directive, the register can be removed during optimization.

2. To set a register value using the HDL initialization feature, use the following syntax:

HDL Initialization

```
reg myreg=0;
initial myreg=0; (Verilog only)
```

Verilog HDL Initialization

```
reg error_reg = 1'b0;
reg [7:0] address_reg = 8'hff;
```

VHDL HDL Initialization

```
signal tmp: std_logic = '0';
```


Copyright © 2013 Synopsys, Inc.

This is the preferred method to pass INIT values to the Xilinx place-and-route tools.

3. To set a register value using the synthesis attribute, add the attribute to the register in the source code or the constraint file, and specify the INIT value as a string:

Verilog	`reg [3:0] rst_cntr /* synthesis INIT="1" */;`
VHDL	`attribute INIT: string;` `attribute INIT of rst_cntr : signal is "1";`
SDC	`define_attribute {i:rst_cntr} INIT {"1"}`

Xilinx ISE 8.2sp3 and later versions require that the INIT value be a string rather than an integer. For code examples, see *INIT Values*, on page 1617 in the *Reference Manual*.

4. To specify different INIT values for each register bit on a bus, do the following:
   - Set syn_preserve on the register as described in step 1, so that it is not optimized away. You can now either use the HDL specification or set an attribute.
   - To specify the values using the HDL specification, use the syntax as shown in the following examples:

Verilog HDL Bus Initialization	`reg [7:0] address_reg = 8'hff;`
VHDL HDL Bus Initialization	`signal q: std_logic_vector` `(11 downto 0) := X"755";`

   - To specify the value with the INIT attribute in the sdc constraint file, set INIT values for the individual register bits on the bus. Specify the register using the i: prefix, with periods as hierarchy separators.

     The following specifies INIT values for individual bits of rst_cntr, which is part of the init_attrver module, under the top-level module:

```
define_attribute {i:init_attrver.rst_cntr[0]} INIT {"0"}
define_attribute {i:init_attrver.rst_cntr[1]} INIT {"1"}
define_attribute {i:init_attrver.rst_cntr[2]} INIT {"0"}
define_attribute {i:init_attrver.rst_cntr[3]} INIT {"1"}
```

5. Synthesize the design.

   The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist. Note that the INIT value is forward-annotated as is (as an integer, not binary). You must ensure that the value is specified in the correct format for P&R.

   If the register is an asynchronous output register with an initial value, the mapper preserves the initial value and packs the register into the Block RAM.

Inserting Xilinx I/Os and Specifying Pin Locations

By default, the synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). You can change this by enabling Disable IO Insertion in the Device tab of the Implementation Options dialog box. You can also insert I/Os manually by instantiating them.

Whether you use the automatic or manual method, you can specify pin locations for the I/Os with the xc_loc attribute. By default, or if no location is specified, the Xilinx tool assigns pin locations automatically.

The following provide details:

- Assigning Pin Locations for Automatically Inserted Xilinx I/Os, on page 809
- Manually Inserting Xilinx I/Os in Verilog, on page 812
- Manually Inserting Xilinx I/Os in VHDL, on page 814

Assigning Pin Locations for Automatically Inserted Xilinx I/Os

The synthesis tool automatically inserts the I/Os (unless you have checked Disable IO Insertion in the Device tab of the Implementation Options dialog box). The following procedure shows you how to assign pin locations for automatically inserted I/Os in a Verilog or VHDL design.

1. Create a new top-level module or entity and instantiate it in your Verilog or VHDL design.

   This module/entity holds I/O placement information. Creating this lets you keep your vendor-specific information separate from the rest of your design. Your original design remains technology-independent.

Copyright © 2013 Synopsys, Inc.

For example, this is a Verilog counter definition:

```
module cnt4 (cout, out, in, ce, load, clk, rst);
// Counter definition
endmodule
```

You create a top-level module that instantiates your design:

```
module cnt4_xilinx (cout, out, in, ce, load, clk, rst);
```

2. If you do not want to specify locations, specify the inputs or outputs as usual. The following is an example of Verilog inputs in the top-level module:

```
input ce, load, clk, rst;
```

The Xilinx place-and-route tool automatically places these inputs.

3. Optionally, specify I/O locations in the new top-level module, by setting the xc_loc attribute.

You can specify the xc_loc attribute in the Attribute panel of the SCOPE spreadsheet, as shown below.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
2	☑	port	in1[2:0]	xc_loc	P2,P3,P	string	Port placement
3	☑	port	p:out1	xc_loc	R1	string	Port placement
4	☑						

Attributes

Alternatively, you can specify it in the HDL files, as described in *Manually Inserting Xilinx I/Os in Verilog*, on page 812 and *Manually Inserting Xilinx I/Os in VHDL*, on page 814. See *xc_loc*, on page 1608 in the *Reference Manual* for syntax details.

The following Verilog code includes xc_loc attributes that specify the following locations:

– cout at A1

– out in the top left (TL) of the chip

– in[3] at P20, in[2] at P19, in[1] at P18, and in[0] at P17

```
output cout /* synthesis xc_loc="A1" */;
output [3:0] out /* synthesis xc_loc="TL" */;
input [3:0] in /* synthesis xc_loc="P20,P19,P18,P17" */;
```

4. Instantiate the top-level module or entity with the placement information you specified in your design. For example:

```
cnt4 my_counter (.cout(cout), .out(out), .in(in),
   .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule
```

5. Synthesize the design.

   The synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). The Xilinx place-and-route tool automatically selects locations for I/Os with no xc_loc attribute defined. If you specified xc_loc settings, they are honored.

VHDL Automatic I/O Insertion Example

```
library synplify;
entity cnt4 is
   port (cout: out bit;
         output: out bit_vector (3 downto 0);
         input: in bit_vector (3 downto 0);
         ce, load, clk, rst: in bit );
end cnt4;

architecture behave of cnt4 is
begin
-- Behavioral description of the counter.
end behave;

-- New top level entity, created specifically
-- to place I/Os for Xilinx. This entity is typically
-- in another file, so that your original
-- design stays untouched and technology independent.

entity cnt4_xilinx is
   port (cout: out bit;
         output: out bit_vector (3 downto 0);
         input: in bit_vector (3 downto 0);
         ce, load, clk, rst: in bit );

-- Place a single I/O for cout at location A1.
attribute xc_loc : string;
attribute xc_loc of cout: signal is "A1";

-- Place all bits of "output" in the
-- top-left of the chip.
attribute xc_loc of output: signal is "TL";
```


Copyright © 2013 Synopsys, Inc.

```
-- Place input(3) at P20, input(2) at P19,
-- input(1) at P18, and input(0) at P17
attribute xc_loc of input: signal is "P20, P19, P18, P17";

-- Let Xilinx place the rest of the inputs.
end cnt4_xilinx;

-- New top level architecture instantiates your design.
architecture structural of cnt4_xilinx is
-- Component declaration for your entity.

component cnt4
   port (cout: out bit;
         output: out bit_vector (3 downto 0);
         input: in bit_vector (3 downto 0);
         ce, load, clk, rst: in bit );
end component;
begin

-- Instantiate your VHDL design here:
my_counter: cnt4 port map (cout, output, input,
   ce, load, clk, rst);
end structural;
```

Manually Inserting Xilinx I/Os in Verilog

To insert a Xilinx I/O manually, you must instantiate a black box macro for that I/O from the Xilinx library file. You can then choose to assign it a location or have the Xilinx tool automatically select one for it.

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the *installDirectory*/lib/xilinx/unisim.v macro library file to the *top* of the source files list for your project.

2. Create instances of I/Os by instantiating a black box in your Verilog source code.

   These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To insert an I/O manually and specify pin locations, do the following:

1. Create a new top-level module and instantiate your Verilog design.

2. Add the *installDirectory*/lib/xilinx/unisim.v macro library file to the *top* of the source files list for your project.

3. Create instances of I/Os by instantiating a black box in your Verilog source code.

4. Specify I/O locations by adding the xc_loc attribute to the I/Os.

   See *Verilog Manual I/O Insertion Example*, on page 813 for an example of the code.The Xilinx tool honors any locations assigned with the xc_loc attribute, and automatically selects locations for any remaining I/Os without definitions.

Verilog Manual I/O Insertion Example

```
module cnt4 (cout, out, in, ce, load, clk, rst);
/* Your counter definition goes here, */
endmodule
/* Create a top level to place I/Os specifically
   for Xilinx. Any top level pins which do not have
   I/Os will be automatically inserted */
module cnt4_xilinx(cout, out, in, ce, load, clk, rst);
output [3:0] out;
output cout;
input [3:0] in;
input ce, load, clk, rst;wire [3:0] out_c, in_c;
wire cout_c;

/* The xc_loc attribute can be added right after the
   instance name like that shown below, or right before
   the semicolon. */

IBUF i3 /* synthesis xc_loc="P20" */ (.O(in_c[3]), .I(in[3]));
IBUF i2 /* synthesis xc_loc="P19" */ (.O(in_c[2]), .I(in[2]));
IBUF i1 /* synthesis xc_loc="P18" */ (.O(in_c[1]), .I(in[1]));
IBUF i0 /* synthesis xc_loc="P17" */ (.O(in_c[0]), .I(in[0]));

OBUF o3 /* synthesis xc_loc="TL" */ (.O(out[3]), .I(out_c[3]));
OBUF o2 /* synthesis xc_loc="TL" */ (.O(out[2]), .I(out_c[2]));
OBUF o1 /* synthesis xc_loc="TL" */ (.O(out[1]), .I(out_c[1]));
OBUF o0 /* synthesis xc_loc="TL" */ (.O(out[0]), .I(out_c[0]));

OBUF cout_p /* synthesis xc_loc="BL" */ (.O(cout), .I(cout_c));

cnt4 it(.cout(cout_c), .out(out_c), .in(in_c),
   .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule
```


Copyright © 2013 Synopsys, Inc.

Manually Inserting Xilinx I/Os in VHDL

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the corresponding library and use clauses to the beginning of your design units that instantiate the macros.

   ```
   library unisim;
   use unisim.vcomponents.all;
   ```

   The Xilinx unisim.vhd macro library is always visible in the synthesis tool, so do not add this library file to the source files list for your project. To see which design units are available, use a text editor to view the file located in the *installDirectory*/lib/xilinx directory. Do not edit this file in any way.

2. Create instances of I/Os by instantiating a black box in your Verilog source code.

   These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To insert an I/O manually and specify pin locations, do the following:

1. Create a new top-level module and instantiate your VHDL design.
2. Instantiate the Xilinx I/Os.
3. Add the appropriate library and use clauses to the beginning of design units that instantiate the I/Os.

   ```
   library unisim;
   use unisim.vcomponents.all;
   ```

   See the source code in *VHDL Manual I/O Insertion Example*, on page 815 for an example.

4. To specify I/O locations, add the xc_loc attribute to the I/O instances for which you want to specify the locations.
5. If you leave out the xc_loc attribute, the Xilinx place-and-route tool will choose the locations.

VHDL Manual I/O Insertion Example

The following example is a behavioral D flip-flop with instantiated data input I/O. The other ports will have synthesized I/Os.

```
library ieee, synplify;
use synplify.attributes.all;
use ieee.std_logic_1164.all;
-- Library and use clauses for access to the Xilinx Macro Library.
library unisim;
use unisim.vcomponents.all;

entity place_example is
   port (q: out std_logic;
         d, clk: in std_logic );
end place_example;

architecture behave of place_example is
   signal dz: std_logic;

attribute xc_loc of I1: label is "P3";
begin
I1: IBUF port map (I=>d,O=>dz);

process (clk) begin
   if rising_edge(clk) then
      q<=dz;
   end if;

end process;

end behave;
```

Working with Xilinx Buffers

By default, the synthesis tools do not automatically infer Xilinx buffers. If you want the tools to infer Xilinx buffers, you must use attributes, as described below.

1. To infer BUFGMUX components, do the following:
   - Attach the syn_insert_buffer attribute to the mux instance. If you need information on how to do this, see *Specifying Attributes and Directives*, on page 245.
   - Set the attribute value to bufgmux. When you set this value, the tool infers a BUFGMUX_1 if the muxed clock operates on the negative edge;

Copyright © 2013 Synopsys, Inc.

otherwise it infers a BUFGMUX. If you do not specify this value, by default the tool infers the LUT that drives the BUFG.

```
module
bufgmux_1(c1,c2,sel,din,d
out);
input c1,c2,sel;
input [20:1] din;
output reg [20 : 1] dout;
wire clk;

assign clk = sel ? c1 :
```

For details about the syn_insert_buffer syntax, see *syn_insert_buffer (Xilinx),* on page 1360 in the *Reference Manual*

2. To infer IBUFDS, IBUFGDS, OBUFDS, OBUFTDS, and IOBUFDS differential buffers, do the following:
   - Attach the syn_diff_io attribute to the inputs of the buffer.
   - Set the value to 1 or true.

   For details about the syn_diff_io syntax, see *syn_diff_io,* on page 1245 in the *Reference Manual*.

Working with Xilinx Regional Clock Buffers

The Xilinx regional clock buffer (BUFR) available for Virtex-6, Virtex-5, and Virtex-4 devices is a special buffer used to connect the clock nets in the same region and adjacent regions, independent of the global clock tree. The BUFR can drive the I/O logic and logic resources (CLB, block RAM, and DSP) in the existing and adjacent clock regions. They can be driven by clock capable pins, local interconnects, Gigabit Transceiver (GT), and Mixed-Mode Clock Manager (MMCM) high-performance clocks. The BUFR can be used as a clock divider in low power and low skew operations.

By default, the synthesis tools do not automatically infer the Xilinx BUFR. If you want the tools to infer Xilinx regional clock buffers, you must use the attribute, as described below.

1. To infer BUFR primitives, do the following:
   - Apply the syn_insert_buffer attribute on a port or net.

– Set the attribute value to BUFR.

For details about the syn_insert_buffer syntax, see *syn_insert_buffer (Xilinx)*, on page 1360 in the *Reference Manual*

2. Check the log file for the number of BUFRs connected in the region. The report specifies:
   – The output of the BUFR used on the clock net should be defined as the derived clock and the timing report should be specified for the clock signal.
   – If inferred

```
Clock Buffers:
Inserting Clock buffer on net clk1_en, Inserting Clock buffer
for port clk,
```

   – Not inferred

```
Warning: BUFR not inserted on net <name>
```

   – Resource utilization

```
Mapping to part: xc6vlx75tff484-1
Cell usage:
BUFR 2 uses
```

Specifying RLOCs

RLOCs are relative location constraints. They let you control placement in critical sections, thus improving performance. You specify RLOCs using three attributes, xc_map, xc_rloc, and xc_uset. As with other attributes, you can define them in the source code, or in the SCOPE window.

You can also specify RLOCs directly, as described in *Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute*, on page 819.

1. Create the modules you want to constrain, and specify the kind of Xilinx primitive you want to map them to, using the xc_map attribute. The modules can have only one output.

Copyright © 2013 Synopsys, Inc.

Family	xc_map Value	Max. Module Inputs
XC4000, Spartan families	fmap hmap	4 3
Virtex and Spartan-3 families	lut	4

This Verilog example shows a 4-input Spartan XOR module:

```
module fmap_xor4(z, a, b, c, d) /* synthesis xc_map=fmap*/ ;
output z;
input a, b, c, d;
assign z = a ^ b ^c ^d;
endmodule
```

This is the equivalent VHDL example:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fmap_xor4 is
   port (a: in std_logic;
         b: in std_logic;
         c: in std_logic;
         d: in std_logic
         );
end fmap_xor4;

architecture rtl offmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl: architecture is "fmap";
begin
   z <= a xor b xor c xor d;
end rtl;
```

2. Instantiate the modules you created at a higher hierarchy level.
3. Group the instances together (xc_uset attribute) and specify the relative locations of instances in the group with the xc_rloc attribute.

This example shows the Verilog code for the top-level CLB that includes the 4-input module in the previous example:

```
module clb_xor9(z, a) ;
output z;
input [8:0] a;
wire x03, x47;
//Code for XC4000 or Spartan
fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
   (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
   (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
   (z, z03, z47, a[8]);
//Code for Virtex differs because it includes the slice
fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
   (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
   (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.S1" */
   (z, z03, z47, a[8]);endmodule
```

4. Create a top-level design and instantiate your design.

Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute

You can specify RLOCs and RLOC_ORIGINs with the synthesis attribute, and then pass them to the Xilinx P&R tools. Alternatively, you can specify RLOCs using the three attributes described in *Specifying RLOCs*, on page 817.

1. In the source code, use the synthesis attribute to specify the RLOC and RLOC_ORIGIN values:

Verilog
```
/* synthesis RLOC_ORIGIN="X0Y2" RLOC="X0Y0" */;
```

VHDL
```
attribute RLOC_ORIGIN : string;
attribute RLOC_ORIGIN of behave : architecture is "X0Y2";
attribute RLOC : string;
attribute RLOC of q : signal is "X0Y0";
```

For code examples, see *RLOC Constraints*, on page 1618 in the *Reference Manual*.

Copyright © 2013 Synopsys, Inc.

2. To specify different RLOC and RLOC_ORIGIN values for bits on a bus, do the following:
   - Specify the RLOC_ORIGIN for theVerilog module or VHDL architecture in the source file. See step 1 for the syntax.
   - Define RLOCs for the individual register bits as constraints in the `sdc` file. Do not define RLOCs for individual bits in the source code, or you will get a Xilinx ISE error.

```
define_attribute {i:tmp[0]} RLOC {"X3Y0"}
define_attribute {i:tmp[1]} RLOC {"X2Y0"}
define_attribute {i:tmp[2]} RLOC {"X1Y0"}
define_attribute {i:tmp[3]} RLOC {"X0Y0"}
```

3. Synthesize the design.

   The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist.

Using Clock Buffers in Virtex Designs

The software can infer a buffer called BUFGDLL that includes the CLKDLL primitive. BUFGDLL consists of an IBUFG followed by a CLKDLL (Clock Delay Locked Loop) followed by a BUFG. To use this CLKDLL primitive, you must specify the xc_clockbuftype attribute. The following steps show you how to add the attribute in SCOPE or the HDL files.

1. To specify the attribute in the SCOPE window, use the procedure described in *Specifying Attributes Using the SCOPE Editor*, on page 250 to add the xc_clockbuftype attribute to a port.

   The software infers a buffer as shown in the following figure.

The output EDIF netlist contains text like the following:

(instance clk_ibuf (viewRef PRIM (cellRef BUFGDLL (libraryRef VIRTEX)))

2. To specify the attribute in Verilog, add the attribute as shown in this example.

```
module test(d, clk, rst, q);
input [1:0] d;
input clk /* synthesis xc_clockbuftype = "BUFGDLL" */, rst;
output [1:0] q;
//other coding
```

3. To specify the attribute in VHDL, add the attribute as shown in this example.

```
entity test_clkbuftype is
   port (d: in std_logic_vector(3 downto 0);
      clk, rst : in std_logic;
      q : out std_logic_vector(3 downto 0)
);
attribute xc_clockbuftype of clk : signal is "BUFGDLL";
end test_clkbuftype
```


Copyright © 2013 Synopsys, Inc.

Working with Clock Skews in Xilinx Virtex-5 Physical Designs

The Synplify Premier software version 9.0.1 and later supports clock skews for Virtex-5 devices , and the SRM clock insertion delay property (Clock input arrival_time). Clock insertion delay models are included for the following components:

- DCM
- BUFG
- BUFR
- BUFIO
- Flip-flops generating clocks

This feature ensures that cross-clock paths are compared correctly. Also, it has a large impact on timing constraints for I/O paths, since any clock delay will be added to the output delay and subtracted from the setup delay. This results in improved timing correlation between the Synplify Premier software and Xilinx timing.

Example1: Calculating Slack with Clock Skew

Clock skew is utilized to calculate the slack in the following Virtex-5 example:

- Source DCM (clock insertion delay = 0.000ns)
- Load IBUFG (clock insertion delay = 4.157ns)

```
                    Requested Period: 5.000
                      - (Setup Time): 0.004
     + (Clock Delay at Ending Point): 4.157
   + (Clock Latency at Ending Point): 0.000

                     = Required Time: 9.153
                - (Propagation Time): 0.746
 - (Clock Latency at Starting Point): 0.000
              = Slack (non-critical): 8.407
```

Example 2: Calculating Slack Using Clock Skew

Clock skew is utilized to calculate the slack in the following Virtex-5 example:

- Source IBUFG (clock insertion delay = 4.157ns)
- Load DCM (clock insertion delay = 0.000ns)

```
                    Requested Period: 5.000
                      - (Setup Time): 0.004
   + (Clock Latency at Ending Point): 0.000

                     = Required Time: 4.996
                - (Propagation Time): 0.745
   - (Clock Delay at Starting Point): 4.157
 - (Clock Latency at Starting Point): 0.000
                  = Slack (critical): 0.094
```

The Synplify Premier software does not automatically forward annotate constraints for derived clocks. Therefore, a clock generated from a set of flip-flops and logic requires you to add a constraint in the UCF file. As a recommendation, derive the clock period the same as the original clock and add a 2-cycle multicycle path from the clock to itself. Better solutions will be provided in the future.

Instantiating Special I/O Standard Buffers for Virtex

The software supports all the I/O Virtex standards, like HSTL_*, CTT, AGP, PC133_*, PC166_*, etc. You can either instantiate these primitives directly, or specify them with the xc_padtype attribute.

1. To instantiate I/O buffers, use code like the following to specify them.

```
module inst_padtype(a, b, clk, rst, en, bidir, q) ;
input [0:0] a, b;
input clk, rst, en;
inout bidir;
output [0:0] q;

reg [0:0] q_int;
wire a_in, q_in;
IBUF_AGP i1 (.O(a_in), .I(a) ) ;
IOBUF_CTT i2 (.O(q_in), .IO(bidir) , .I(q_int), .T(en) ) ;
OBUF_F_12 o1 (.O(q), .I(q_in) ) ;
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge clk or posedge rst)
   if (rst)
      q_int <= 1'b0;
   else
      q_int <= a_in & b;

endmodule
```

2. To specify the I/O buffers with an attribute, add the attribute in the SCOPE window (refer to *Specifying SCOPE Constraints*, on page 135 for details) or in the source code, as the following example illustrates.

```
module inst_padtype(a, b, clk, rst, en, bidir, q) ;
   input [0:0] a /* synthesis xc_padtype = "IBUF_AGP" */, b;
   input clk, rst, en;
   inout bidir /* synthesis xc_padtype = "IOBUF_CTT" */;
   output [0:0] q /* synthesis xc_padtype = "OBUF_F_12" */;

   reg [0:0] q_int;

   assign q = bidir;
   assign bidir = en ? q_int : 1'bz;
   always @(posedge clk or posedge rst)
      if (rst)
         q_int <= 1'b0;
      else
         q_int <= a & b;
endmodule
```

Reoptimizing with EDIF Files

You can add an EDIF file created by Synplify Pro/Synplify Premier as a lower level module or you can resynthesize an EDIF file created by Synplify Pro/Synplify Premier at the top level of the hierarchy to further refine and optimize your design.

To add an EDIF file created by Synplify Pro/Synplify Premier as a lower level module, follow these steps:

1. Disable I/O insertion when creating the EDIF file as the file will not be instantiated at the top level of the hierarchy.
2. Make sure the EDIF file name matches the module name.
3. Create a project and add the EDIF file to the design.

4. Synthesize the design.

To resynthesize an EDIF file created by Synplify Pro/Synplify Premier at the top level of the hierarchy, do the following:

1. Make sure your design does not include any mixed-language files.
2. Check that the EDIF file name matches the module name.
3. Create a project and add the EDIF file to the design.
4. Specify the EDIF file as the top-level design module by doing the following:
   - Click Implementation Options and go to the Verilog or VHDL tab.
   - Enter the module name in the Top Level Module/Entity field. If your module is not in the work library, first specify the library as *libraryName*.*moduleName*.
   - Click OK.
5. Resynthesize your design.

Improving Xilinx Physical Synthesis Performance

The Synplify Premier tool is timing-driven; optimizations depend on timing constraints and are applied until all constraints are met. Therefore, it is very important that you adequately apply timing constraints and not over-constrain the tool. This section includes guidelines for applying constraints.

- Verify the consistency of constraints between synthesis and P&R:
  - Clock constraints
  - Clock-to-clock constraints
  - IO delays
  - IO standard, drive, slew and pull-up/pull-down
  - Multi-cycle and false paths
  - Max-delay paths
  - DCM parameters
  - Register packing into IOB

Copyright © 2013 Synopsys, Inc.

  - LOC/RLOC constraints on macros (BUFG, DCM, RAMB, DSP, MULT, etc.)
  - LOC/RLOC constraints on instances (Register, LUT, SRL, RAMS, RAMD, etc.)
  - AREA_GROUP constraints
  - IDELAYCTRL and IDELAY constraints

- Ensure that the final physical synthesis slack is negative, but no more than 10-15% of the clock constraint.
- Check the log file for Pre-placement timing snapshot.
  If it indicates that a clock has positive slack at this point, but in the final results the clock has negative slack, use the -route constraint for the clock. This lets you to control the amount of early timing optimizations for the clock domain. However, large -route values can degrade performance. Therefore, to determine the correct -route value to use, start with smaller values and increase iteratively. For example, start with half the difference between the estimate and actual slack, or 5% of the clock estimate, whichever is the smallest.
- Experiment with ignoring the relationally-placed macro (RPM) constraints.

  RPMs (also known as RLOCs) can negatively affect results. You can compare placement results using the Synplify Premier tool by setting the global attribute xc_use_rpms to 0. For details on this attribute, see *xc_use_rpms*, on page 1627 in the *Reference Manual*.
- Ensure placement for I/Os, Block Rams, and DSP48 devices.

  This version of the tool uses the Xilinx placer to generate locations for I/Os and block components. To avoid block component placement problems, you need to lock placement. See *Generating a Xilinx Coreloc Placement File*, on page 327 for information.

Running Post-Synthesis Simulation

For post-synthesis simulation with a Xilinx design, do the following:

1. Run synthesis as usual.

   The run generates a `vhm` file, which references the synplify library:

```
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;
```

2. Set up the libraries.
   - Create a library called synplify and compile synplify.vhd into it. The synplify.vhd file is located in *installDirectory*/lib/vhdl_sim.
   - Create a library called UINISIM, and compile the UNISIM simulation library provided by Xilinx into it.
3. Compile the vhm file into work. For example:

```
vcom -work synplify installDirectory/lib/vhdl_sim/synplify.vhd
```

Working with Xilinx Place-and-Route Software

The following procedure shows you how to run the Xilinx place-and-route tool from within the synthesis software.

1. Set the XILINX environment variable to point to your Xilinx software installation directory.
2. Start the synthesis software and open a synthesized design.
3. Start the place-and-route software:
   - To start Xilinx Design Manager, select Options->Xilinx->Start Design Manager.
   - To start Xilinx floorplanner, select Options->Xilinx->Start Floorplanner.
   - To start the ISE tool, select Options->Xilinx->Start ISE Project Navigator.

Limitations Using the Xilinx ISE Tool

Be aware of the following limitations when using the Xilinx ISE tool:

- When you invoke the Xilinx ISE tool in either foreground or background from Options->Xilinx->Start ISE Project Navigator, the synplicity.ucf file is not added to the Project file automatically. You must manually add the synplicity.ucf file to the project.

Copyright © 2013 Synopsys, Inc.

- When you launch Xilinx ISE from the synthesis tool, default settings are used. Any setting changes that you make to the Xilinx project seem to disappear when you subsequently launch ISE.

  Changes made to the Xilinx project are saved in the `npl` file in the working directory. To override normal defaults, open the `npl` file saved previously, before launching ISE.

CHAPTER 16

Working with Synthesis Output

This chapter covers techniques for optimizing your design for various vendors. The information in this chapter is intended to be used together with the information in Chapter 7, *Inferring High-Level Objects*.

This chapter describes the following:

- Passing Information to the P&R Tools, on page 830
- Generating Vendor-Specific Output, on page 834
- Invoking Third-Party Vendor Tools, on page 836

Copyright © 2013 Synopsys, Inc.

Passing Information to the P&R Tools

The following procedures show you how to pass information to the place-and-route tool; this information generally has no impact on synthesis. Typically, you use attributes to pass this information to the place-and-route tools. This section describes the following:

- Specifying Pin Locations, on page 830
- Specifying Locations for Microsemi Bus Ports, on page 831
- Specifying Macro and Register Placement, on page 831
- Passing Technology Properties, on page 832
- Specifying Padtype and Port Information, on page 832

Specifying Pin Locations

In certain technologies you can specify pin locations that are forward-annotated to the corresponding place-and-route tool. The following procedure shows you how to specify the appropriate attributes. For information about other placement properties, see *Specifying Macro and Register Placement*, on page 831.

1. Start with a design using one of the following vendors and technologies: Microsemi, Altera, Xilinx, Lattice, or QuickLogic families.
2. Add the appropriate attribute to the port. For a bus, list all the bus pins, separated by commas. To specify Microsemi bus port locations, see *Specifying Locations for Microsemi Bus Ports*, on page 831.
   - To add the attribute from the SCOPE interface, click the Attributes tab and specify the appropriate attribute and value.
   - To add the attribute in the source files, use the appropriate attribute and syntax. See the *Reference Manual* for syntax details.

Family	Attribute and Value

Microsemi	`syn_loc {pin_number}`
	or
	`alspin {pin_number}`

Altera	`syn_loc {pin_number}`

Lattice	`loc {pin_number}`
QuickLogic	`ql_placement {pin_number}`

Xilinx	`syn_loc {pin_number}`
	or
	`xc_loc {pin_number}`
	See *Specifying RLOCs*, on page 817 for details about relative placement.

Specifying Locations for Microsemi Bus Ports

You can specify pin locations for Microsemi bus ports. To assign pin numbers to a bus port, or to a single- or multiple-bit slice of a bus port, do the following:

1. Open the constraint file an add these attributes to the design.
2. Specify the syn_noarrayports attribute globally to bit blast all bus ports in the design.

   ```
   define_global_attribute syn_noarrayports {1};
   ```

3. Use the alspin attribute to specify pin locations for individual bus bits. This example shows locations specified for individual bits of bus ADDRESS0.

   ```
   define_attribute  {ADDRESS0[4]} alspin {26}
   define_attribute  {ADDRESS0[3]} alspin {30}
   define_attribute  {ADDRESS0[2]} alspin {33}
   define_attribute  {ADDRESS0[1]} alspin {38}
   define_attribute  {ADDRESS0[0]} alspin {40}
   ```

   The software forward-annotates these pin locations to the place-and-route software.

Specifying Macro and Register Placement

You can use attributes to specify macro and register placement in Microsemi and QuickLogic designs. The information here supplements the pin placement information described in *Specifying Pin Locations*, on page 830 and bus pin placement information described in *Specifying Locations for Microsemi Bus Ports*, on page 831.

Copyright © 2013 Synopsys, Inc.

For...	Use...
Relative placement of Microsemi macros and IP blocks	*alsloc* define_attribute {u1} alsloc {R15C6}
Placement of Lattice ORCA input or output registers next to I/O pads	*orca_padtype* define_attribute { load } orca_padtype "IBT"

Passing Technology Properties

The following table summarizes the attributes used to pass technology-specific information for certain vendors. For details about the attributes in the table, see the *Reference Manual*.

Vendor	Attribute for passing properties
Lattice ORCA	orca_props define_attribute {p:data_in} orca_props {LEVELMODE=LVDS}

Xilinx	Specify the Xilinx properties directly in the source code. The software passes them to the place-and-route tool. For example:
	`attribute INIT of RAM1 : label is "0000";`
	or
	`/* synthesis INIT_xx = "value" */`

Specifying Padtype and Port Information

For many vendors, you can use attributes to specify technology-specific port information or padtype.

Information	Vendor	Attribute
Padtype	Lattice ORCA	orca_padtype define_attribute {AIN[3]} orca_padtype {IBT}
	Xilinx	xc_padtype define_attribute {a[3:0]} xc_padtype {IBUF_GTLP}

Copyright © 2013 Synopsys, Inc.

Information	Vendor	Attribute
	Altera	altera_io_powerup define_attribute {seg [31:0]} altera_io_powerup {high}
	Xilinx	xc_isgsr define_attribute {bbgsr.gsrin} xc_isgsr {1}
	Xilinx	xc_pullup/xc_pulldown define_attribute { port_name } xc_pulldown { 1 }

Copyright © 2013 Synopsys, Inc.

Generating Vendor-Specific Output

The following topics describe generating vendor-specific output in the synthesis tools.

- Targeting Output to Your Vendor, on page 834
- Customizing Netlist Formats, on page 835

Targeting Output to Your Vendor

You can generate output targeted to your vendor.

1. To specify the output, click the Implementation Options button.
2. Click the Implementation Results tab, and check the output files you need.

   The following table summarizes the outputs to set for the different vendors, and shows the P&R tools for which the output is intended.

Vendor	Output Netlist	P&R Tool
Altera Stratix, Max-II, Cyclone	Verilog (`.vqm`)	Quartus II
Lattice (newer device families)	EDIF (`.edf`)	Diamond
Lattice Classic (Mach/isp device families)	EDIF (`.edf`) or `.src`	ispExpert
Lattice Classic (Orca)	EDIF (`.edn`)	ispLEVER
Microsemi	EDIF (`.edn`) *_sdc.sdc	Libero SoC or IDE
Xilinx 7 Series	EDIF/VM (`.edf` or `.vm`)	Vivado
Xilinx Virtex and Spartan-3	EDIF (`.edf`)	Design Manager or ISE Project Navigator
Xilinx CoolRunner	EDIF (`.edf`) or `.src`	Web Fitter for EDIF files, Minc for *.src files

Copyright © 2013 Synopsys, Inc.

3. To generate mapped Verilog/VHDL netlists and constraint files, check the appropriate boxes and click OK.

   See *Specifying Result Options*, on page 237 for details about setting the option. For more information about constraint file output formats and how constraints get forward-annotated, see *Generating Constraint Files for Forward Annotation*, on page 125.

Customizing Netlist Formats

The following table lists some attributes for customizing your Microsemi, Altera, and Xilinx output netlists:

For...	Use...
Netlist formatting	*syn_netlist_hierarchy* (Altera, Xilinx, Microsemi) define_global_attribute syn_netlist_hierarchy {0}
EDIF formatting (Xilinx)	*syn_edif_bit_format* (Xilinx) define_global_attribute syn_edif_bit_format {%n<%i>}
	syn_edif_name_length (Xilinx) define_global_attribute syn_edif_name_length { restricted }
	syn_edif_scalar_format (Xilinx) define_global_attribute syn_edif_scalar_format {%u}
Bus specification	*syn_noarrayports* (Altera, Xilinx, Microsemi) define_global_attribute syn_noarrayports {1}

Copyright © 2013 Synopsys, Inc.

Invoking Third-Party Vendor Tools

You can invoke third-party tools from within the Synopsys FPGA synthesis products, and configure the locations and common arguments for the tools. This capability lets you modify source files or libraries or debug problems from within the third-party tool, without leaving the synthesis environment.

You can invoke preconfigured tools or add your own. The process consists of two steps:

- Configuring Tool Tags, on page 836
- Invoking a Third-Party Tool, on page 837

Configuring Tool Tags

A tool tag is a configuration definition for a tool you want to invoke from the synthesis interface. You define a tool tag to set up the third-party tool you want to use. The synthesis software has some popular applications already configured for you to use when the synthesis software starts up: System Designer, Altera Megawizard, Xilinx EDK, and Xilinx Coregen. The following procedure shows you how to define your own tool tags, or add command arguments. Use this to specify other tools, other versions of a tool, or to run a tool with different arguments.

1. Select Options->Configure 3rd Party Tool Options from the Project view.

2. Define the application tag information for the tool you want to invoke.

- Specify the application you want to invoke in Application Tag Name.
- Specify how you want to invoke the application tool. If you want to run the tool directly from the UI, select Direct Execution. If your application is a Tcl procedure, select TCL Mode.
- Specify the location of the application executable or Tcl procedure name in Application Name with Path or Tcl Procedure Name.
- Specify any command arguments you want in the Command Argument if any field. You can use this to define a new tool tag or to add arguments to a tool tag that is already defined.

  For a list of predefined command arguments, click the + button and select them from the list. Otherwise, type the command arguments. For the System Designer and other internal Synopsys tools, you must select $SynCode from the Command Argument if any field.
- Click Apply.
- Click Close.

The tool saves these settings in the FPGA synthesis tool `.ini` file and retrieves them for subsequent invocations. For information about invoking a third-party tool, see *Invoking a Third-Party Tool*, on page 837, next.

Invoking a Third-Party Tool

You can define tool tags globally and then use these tool tags to run the third-party tool from the Project view for the specified tool tag only. Some common tool tags are preconfigured and are read when the application starts up. You can add or modify existing tool tags or define your own Tcl procedures to invoke within the FPGA synthesis tools.

1. Define a tool tag for your application, as described in *Configuring Tool Tags*, on page 836.
2. Right-click in the Project view on a file or folder which is configured to run the vendor tool, and select Launch Tools->Run Vendor Tool from the popup menu.

   This dialog box automatically displays tool tag information associated with the file or folder. If no tool tag information is specified, look for the parent hierarchy and edit or change it, if necessary.

Copyright © 2013 Synopsys, Inc.

3. To associate a file or folder with a particular third-party tool, do the following:
   - Select the file or folder in the Project view. If you select a folder, the third-party tool is associated with all the files in the folder. If you associate a tool with a file, this setting overrides the folder setting.
   - Right-click a file or folder and select Launch Tools->Run Vendor Tool from the popup menu.

   - In the Vendor Tool Invocation dialog box, select the application in Application Tag Name.
   - Include any additional options you want to use with this file when you invoke the vendor tool. You can set command arguments now, if you did not configure them earlier.
   - Verify the command string in the dialog box.
   - Click Save, and Close. The third-party tool is associated with the file or folder and appears in the Launch Tools menu.

4. To invoke an associated third-party tool for a file or folder, do the following:
   - Right-click the file or folder in the Project view.
   - Select Launch Tools-><*Third-Party Tool*> from the popup menu. The synthesis tool automatically runs the tool or Tcl procedure as specified.

5. To invoke the tool at the same time that you associate a third-party tool with a file or folder, or to add additional arguments on the fly, do the following:
   - Right-click a file or folder and select Launch Tools->Run Vendor Tool from the popup menu.
   - In the Vendor Tool Invocation dialog box, select the application in Application Tag Name.
   - Include any additional options you want to use with this file when you invoke the vendor tool. You can set command arguments now, if you did not configure them earlier.
   - Verify the command string in the dialog box.
   - Click Save. The tool and arguments you specified is associated with the file or folder and appears in the Launch Tools menu.

     If you defined a new tool tag, the 3rd Party Tool Configuration dialog box appears. After saving the settings here, go back to the Vendor Tool Invocation dialog box. You are prompted to save this information to the project file before invoking the third-party tool.
   - Click the Run button in the Vendor Tool Invocation dialog box. The synthesis tool launches the third-party tool or runs the Tcl procedure with the arguments you specified.

   These settings are saved in the FPGA synthesis tool `ini` file, from where it can be retrieved for subsequent invocations.

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

CHAPTER 17

Running Post-Synthesis Operations

The following topics describe how to run post-synthesis operations, like place-and-route and verification, with compatible tools:

- Running P&R Automatically after Synthesis, on page 842
- Running Altera Quartus II Incrementally, on page 846
- Running Xilinx Vivado Place-and-Route, on page 852
- Running Vivado Incrementally, on page 864
- Running Xilinx ISE Incrementally, on page 869
- Working with the Identify Tools, on page 880
- Netlist Editing, on page 889
- VIF Formal Verification Flows, on page 893
- Simulating with the VCS Tool, on page 899
- Using the VCD/Identify-Analyst Integration Tool, on page 904

Copyright © 2013 Synopsys, Inc.

Running P&R Automatically after Synthesis

Synplify Pro, Synplify Premier
Altera, Microsemi, Xilinx

You can run place-and-route automatically from within the tool or in batch mode for certain vendor devices. The following diagram illustrates the flow.

For detailed procedures, see the following:

- Integrating Synthesis and Place-and-Route in One Run, on page 842
- Running the Integrated Synthesis and Xilinx ISE Flow, on page 843
- Releasing the Synthesis License During Place and Route, on page 845

Integrating Synthesis and Place-and-Route in One Run

Altera, Xilinx

You can run the place-and-route tool for your target technology automatically after synthesis.

1. Check the Release Notes and make sure that you are using the correct version of the P&R tool.
2. Set the PATH variable to point to the place-and-route tool.
3. To manually launch the P&R tool in Altera and Xilinx technologies, do the following:
   - For Altera designs, select the run option you want from the Options-> *<Altera_tool>* menu. The tool launches and displays the P&R tool interface. You can configure your settings and run P&R.
   - For Xilinx designs, select the run option you want from the Options-> Xilinx menu. The tool launches and displays the place-and-route tool user interface, places the synthesis-generated netlist in a Xilinx project, and names the project. Configure your Xilinx project settings in the place-and-route tool. Depending on the device family and the P&R tool, follow the detailed setup steps in *Running the Vivado Flow*, on page 856, or *Running the Integrated Synthesis and Xilinx ISE Flow*, on page 843.

Copyright © 2013 Synopsys, Inc.

4. To automatically run the P&R tool after synthesis completes, do the following:
   - If necessary, set up a place-and-route implementation as described in . You need a P&R implementation for physical synthesis flows, or if you want to use post-P&R data for backannotation.
   - Click on the Add P&R Implementation button. In the dialog box, select the P&R implementation you want to run and enable Run Place & Route following synthesis.
   - Synthesize the design.

   The tool automatically runs P&R after synthesis.

Running the Integrated Synthesis and Xilinx ISE Flow

Xilinx

This is not a Vivado P&R flow, but you can run this with the ISE P&R tool.

1. In the Implementation Results tab of the Implementation Options dialog box, disable Use Vivado for Place and Route.

   The Result Format is set to `edif` output. For this flow, the synthesis tool maps the output netlist to `edif` format and the timing constraints and netlist properties to the `synplicity.ucf` file. You must run ISE place and route.

Copyright © 2013 Synopsys, Inc.

2. Select the appropriate constraint file for your design.

   The format of timing constraints in the `sdc` or `fdc` file must be the Synopsys standard timing format; if the file contains any legacy timing constraints such as define_clock, the synthesis tool generates an error message. To convert these constraints for Vivado flows, see the *sdc2fdc Tcl Shell Command*, on page 988.

3. Set P&R options as described in *Running P&R Automatically after Synthesis*, on page 842.

4. Run synthesis and place-and-route with ISE.

   See the guidelines below for some tips. You can release the synthesis tool license while place-and-route is running by following the instructions in *Releasing the Synthesis License During Place and Route*, on page 845.

Guidelines for Running Xilinx P&R

The following table lists some tips for running various phases of Xilinx place-and-route:

NGDBuild	Use the user constraint file (`synplify.ucf`) generated after synthesis. From the command line, read the `.ucf` file into Xilinx place-and-route NGDBuild with the -uc command.
MAP	Do not map to 5-input functions. Do not use the -k command line option. For information about using map with compile points, see *Synthesizing Compile Points*, on page 1109.
PAR	Do not use the default effort level of std. Instead, set it to high using the -ol high command line option. For information about using par with the Synplify Pro and Synplify Premier compile-point synthesis flows, see *Synthesizing Compile Points*, on page 1109.

Releasing the Synthesis License During Place and Route

When invoking a third-party place-and-route tool from the FPGA synthesis tool, you can choose to have place and route continue to run even after exiting the synthesis tool so that it does not consume an FPGA license. The software lets you release the license for the synthesis tool and run the place-and-route tool in batch mode.

To release the FPGA license, specify the following command:

```
toolName -batch -license_release
```

Where *toolName* can be any of the following keywords: `synplify`, `synplify_pro`, `synplify_premier`, or `synplify_premier_dp`.

In synthesis batch mode (synbatch), the -license_release option obtains all the synthesis licenses that are checked out for the session and checks them in immediately after the place-and-route job is launched.

When licenses are released, you see the following message is generated:

```
Exiting session due to -license_release option
```


Copyright © 2013 Synopsys, Inc.

Running Altera Quartus II Incrementally

Altera

The Altera place-and-route tool supports incremental place and route. It preserves design implementation data so that it can make incremental place-and-route updates as needed. For complete details on this flow, see the Altera documentation. You can leverage this Quartus functionality by combining it with synthesis compile points to create an integrated flow from RTL partitions to place-and-route. There are two ways to use compile points in Altera flows, based on your design goals:

Design Goal	Altera Flow
Save runtime, or iterate through P&R incrementally.	Incremental P&R with Fast Fit
Stabilize the design and preserve completed parts of the design. Only run incremental P&R on the parts that have changed.	Incremental P&R

- Quartus II Incremental Compilation Flow with Fast Fit

  Use this flow when you are trying to save runtime as you iterate through the flow and make changes. Typically, you use this earlier in the design cycle. In this flow, Quartus II runs with the Fast Fit option. To get even more runtime savings, combine the Fast Fit flow with automatic compile compile points, fast synthesis, and multiprocessing during synthesis.

- Quartus II Incremental Compilation Flow

  Use this flow when you are trying to converge, and design preservation and stability are your primary goals. The same manual RTL compile points defined during synthesis are maintained through place and route. If a compile point is modified, Quartus reruns place and route for only those partitions that have been modified; all others are preserved. To reduce runtime while running this flow, use multiprocessing.

Quartus II Incremental Compilation Flow with Fast Fit

Use this flow in the Synplify Premier tool for incremental changes when you are trying to reduce runtime (see *Running Altera Quartus II Incrementally*, on page 846). For the Synplify Pro tool, you must enable the Fast Fit option

manually. Use the Fast Fit option when you need to make minor changes to a design, like making small HDL changes, moving pin locations, changing attributes, or modifying timing constraints.

The following procedure describes the details of running synthesis and the Fast Fit flow:

1. Set up the synthesis project.
   - Create compile points, as described in *Synthesizing Compile Points,* on page 1109. You can create automatic or manual compile points, but inferring automatic compile points speeds up runtime.
   - In the Synplify Premier tool, enable Fast Synthesis mode. This automatically enables the Quartus Fast Fit option. If you are using the Synplify Pro tool for synthesis, you must manually set the Fast Fit option in the Quartus tool.
   - To speed up runtime, use multiprocessing.
   - Target a supported Altera device.
   - Click the Add P&R Implementation button in the Project view and set up a place-and-route implementation so that it runs automatically after synthesis. See *Running P&R Automatically after Synthesis,* on page 842 for details.
2. Do an initial run of synthesis, placement, and routing. Enable the Quartus Fast Fit option.

   Quartus runs automatically after synthesis completes. It uses low effort to place and route the design.
3. Implement small changes to the design.
4. Rerun synthesis and P&R.

   When you rerun synthesis and P&R on subsequent runs, the tools work incrementally and only rerun the compile points that have been modified. Compile points that were not affected by changes are preserved from the previous run. This results in significant runtime improvements, as the entire design does not need to be rerun.

Copyright © 2013 Synopsys, Inc.

Quartus II Incremental Compilation Flow

In the Altera Quartus II Incremental Compilation flow, the synthesis software automatically generates a Tcl script which creates design partition assignments. The Quartus II software use this Tcl script to determine if partitions should be preserved from previous place and route results when you recompile the design. If a partition is removed, the Tcl script detects this and forward-annotates an updated VQM netlist.

Use this flow when your goal is to stabilize your design. This figure summarizes the flow for working with Quartus incremental compilation, and the steps that follow it provide more detail:

1. Set up the project.
   - Select the target device. Quartus II Incremental Compilation supports most Stratix and newer device families.
   - Set implementation options and constraints.

2. Compile the design, and define manual compile points in the top-level constraint file.
   - Click the Compile Points tab, and set compile points, as described in *Defining Manual Compile Points*, on page 1117.
   - Set the compile point type to hard, locked or locked, partition to run this flow. The following example shows the compile point for v:ff_cp.

   - Create a compile point constraint file and set constraints for each compile point you define. This is especially important if you use a bottom-up flow.
   - Save the constraint file.

3. Click Run to synthesize the design and check the compile point summary in the log file.

   Check the log file for messages like those displayed below.

Copyright © 2013 Synopsys, Inc.

Mapper Messages

```
@N:MF104 : ff_cp.v(1) | Found Compile Point of type locked,partition on View view:work.ff_cp(verilog)


@N:MF105 :  | Performing bottom up mapping of Compile point view:work.ff_cp(verilog)
@N:MF106 : ff_cp.v(1) | Mapping Compile point view:work.ff_cp(verilog)  because
         No database from previous run found.
```

First Run Log Summary

```
 Summary of Compile Points

Name       Status     Reason
--------------------------------
ff_cp      Mapped     No database
test       Mapped     No database
================================
```

The synthesis tool generates a VQM file where the compile points are defined with syn_hier="locked,partition" attributes. For subsequent synthesis runs, the tool also automatically generates a Tcl script which creates design partition assignments.

4. Run the Quartus II place-and-route tool.

   The Quartus II software uses the compile point VQM files, as well as the Tcl script file generated from the synthesis tool to determine if partitions should be preserved from previous place and route results when you recompile the design.

5. Make any required changes, and re-synthesize the modified design.

   The synthesis tool only resynthesizes and optimizes the updated modules. In the VQM file generated for this synthesis run, the tool does not change the timestamp of a compile point if it has not changed since the previous run; it preserves the old timestamp. Updated modules get a new timestamp.

   For an incremental run, the software only resynthesizes compile points whose logic, implementation options, or timing constraints have changed.

   The following figure illustrates incremental synthesis by comparing compile point summaries. After the first run, a logic change in the ff_cp

module. The figure shows that incremental synthesis resynthesizes ff_cp (logic change), but does not resynthesize test because the logic did not change.

6. Rerun the Quartus II tool to place and route the design.

   The Altera Quartus II software checks the compile point file with the corresponding file from the previous run, and incrementally places and routes only those partitions that have changed, using the information from the updated files. It leaves the other partitions untouched, re-using information from the previous run.

Copyright © 2013 Synopsys, Inc.

Running Xilinx Vivado Place-and-Route

Xilinx

Xilinx Vivado Integrated Design environment (IDE) is the recommended place-and-route tool for designs using the Xilinx 7 Series (Virtex-7, Kintex-7, and Artix-7) devices. This place-and-route tool is not backward compatible, so does not support Xilinx Virtex-6 and older devices.

The FPGA synthesis tools can be run with the Vivado place-and-route tool for the following processes: Synplify Pro logic synthesis, Synplify Premier with fast synthesis, and Synplfiy Premier with enhanced optimization.

For details, see the following sections:

- Vivado Place-and-Route Design Flow, on page 853
- Setting Vivado Environment Variables, on page 855
- Running the Vivado Flow, on page 856
- Customizing Vivado Place and Route Options, on page 860

Vivado Place-and-Route Design Flow

There are two Vivado place-and-route (P&R) design flows, Flow 1, and Flow 2. The diagram shows the required inputs and outputs with these flows for the synthesis tools to run Vivado successfully. In both flows, design constraints are forward-annotated to place-and-route through an `xdc` file. The file includes Synopsys standard timing constraints and non-timing design constraints.

- Flow 1 (Default)
  The synthesis tool generates an `edif` netlist for place and route. Constraints are forward-annotated through an `_edif.xdc` file.
- Flow 2
  The synthesis tool generates a `vm` structural Verilog netlist for place and route instead of the edif netlist. Constraints are forward-annotated through an `xdc` file.

Copyright © 2013 Synopsys, Inc.

The `_edif.xdc` and `xdc` constraint files are separate for each flow and include formats that cannot be mixed.

For details about running through these flows, see the following topics:

- Setting Vivado Environment Variables, on page 855
- Running the Vivado Flow, on page 856
- Customizing Vivado Place and Route Options, on page 860
- Limitations to Vivado Support, on page 854

Limitations to Vivado Support

Vivado P&R support includes the following limitations:

- Designs with IP cores
  - For secure and non-secure NGC/NGO cores, you must use the synthesis flow that generates the `edif` netlist and `xdc` constraint file for Vivado P&R.
  - For secure cores that are generated with Vivado software, the encrypted IP cannot be integrated into the Vivado flow so you must black box it in the synthesis tools.
- Input constraints in UCF/NCF file format are not supported in the synthesis tools. You must manually convert the constraints to FDC file format, before adding the constraints for synthesis and that are forward annotated to the Vivado P&R tool.
- Input constraints in XDC file format are not supported in the synthesis tools. You must manually convert these timing constraints to Synopsys timing constraints format (SDC/FDC) before adding the constraints for synthesis and that are forward-annotated to the Vivado P&R tool. Also, other non-timing constraints in the XDC file must be added to the Vivado P&R run.
- On Linux, Vivado P&R only runs on Red Hat Enterprise Linux 5 or later.

Copyright © 2013 Synopsys, Inc.

Setting Vivado Environment Variables

To successfully launch the Vivado tool, do the following:

1. Check the Release Notes and make sure that you are using the recommended versions of the ISE and Vivado tools.

2. Set both the XILINX and XILINX_VIVADO environment variables to point to their respective installation locations.

   – On Linux platforms, set it at as follows:

   ```
   XILINX=/remote/edaTools/14.1/ISE_DS/ISE
   XILINX_VIVADO=/remote/edaToolkits/Vivado/2012.1
   ```

   – For Windows, set the variables from the Control Panel:

Make sure that the ISE and Vivado variables point to the same Xilinx version; for example ISE 14.3 and Vivado 2012.3. This is because the

Copyright © 2013 Synopsys, Inc.

Vivado tool uses some ISE executables, so you must point to compatible versions of these tools.

The XILINX variable is used for ngc2edif conversion as well as to load BRAM primitives (BRAM_SDP_MACRO) automatically from the Xilinx installation. If you do not have a variable set for a valid Xilinx installation at runtime, manually add the BRAM_TDP_MACRO.v file containing the unimacro definition to the Synplify project.

Running the Vivado Flow

To run the integrated synthesis and Vivado P&R flow using either an edif or vm netlist (Flows 1 and 2 in *Vivado Place-and-Route Design Flow*, on page 853), follow the steps below:

1. Create a synthesis project file, and set Vivado environment variables as described in *Setting Vivado Environment Variables*, on page 855.
2. Specify the netlist flow.
   - On the Implementation Results tab of the Implementation Options panel, enable Use Vivado for Place and Route.
   - For the Edif netlist flow, select `edif` from the Result Format pull-down menu. This is the default.
   - For the Structural Verilog netlist flow, select `vm` from the Result Format pull-down menu. The Write Mapped Verilog Netlist option is enabled and greyed out. You cannot change this setting.

3. Set other options and synthesize as usual.

   After synthesis, the tool generates the following files, depending on which output netlist flow you specified:

EDIF Flow	Structural Verilog Flow
edif Netlist	vm Netlist (Can be used for simulation and formal verification, as well as Vivado P&R)
_edif.xdc Constraints	xdc Constraints

4. If you want to generate a netlist in another format without rerunning synthesis, do the following:
   – Change the Result Format on the Implementation Results tab of the Implementation Options panel. The figure below shows it being set to edif after a vm run.

Copyright © 2013 Synopsys, Inc.

– Select the Run->Write Output Netlist Only menu option from the Project view. This generates only the netlist.

  You can also choose to run the integrated synthesis and ISE place-and-route flow instead, but this is not a Vivado P&R flow. For information about this flow, see *Running the Integrated Synthesis and Xilinx ISE Flow*, on page 843.

– Select the appropriate constraint file for your design.

  The `sdc` or `fdc` file can only contain Synopsys standard timing constraints and non-timing design constraints. If the `sdc` file contains any legacy timing constraints such as define_clock, the synthesis tool generates an error message. To convert these constraints for Vivado, see *sdc2fdc Tcl Shell Command*, on page 988.

5. Add a place-and-route implementation to your project by right-clicking the implementation and selecting Add Place & Route or by clicking the Add P&R Implementation button in the Project view.

   For details, see *Creating a Place and Route Implementation*, on page 303. Once you click OK, the P&R implementation directory (pr_1) is created in your project by default. In the following example, the P&R directory

name has been changed to par_1. All P&R revisions are written to this directory.

6. Run Vivado place-and-route.
   - To integrate the P&R step and run the place-and-route tool automatically after synthesis, enable the Run Place & Route following synthesis option.

     After synthesis, the tool writes output files to the par_1 P&R directory by default, but you can specify another directory, if necessary. The synthesis tool also copies the netlist and constraint files to the par_1 P&R directory.

     The synthesis tool also generates the run_vivado.tcl file in the par_1 P&R directory, and then calls and executes run_vivado.tcl in batch mode. You can also use the run_vivado.tcl file to run manually in batch mode. You can edit the place-and-route options before running Vivado,

Copyright © 2013 Synopsys, Inc.

using the procedure described in *Customizing Vivado Place and Route Options*, on page 860.

Check the P&R results in the vivado.log file.

Customizing Vivado Place and Route Options

After synthesis in the Vivado flow, the tool automatically generates a run_vivado.tcl file which contains default options to run Vivado. You can use the default file, or create a new options file or customize an existing options file, as described below.

1. Right-click the the place and route implementation and select P&R Options.

2. To create a new options file for run_vivado.tcl, follow these steps:
   – Click Create New Options File in the Add New Place & Route Task dialog box. The tool creates a new run_vivado.tcl file using the standard options as the template. For subsequent designs, you can select any P&R options file from the list to use as a template.

– Open this file in the Project view, and specify the options you want. See *Vivado P&R Option File*, on page 861 for an example of the file and a description of some of the options.

3. To edit an existing options file, click Existing Options File, open the file in the Project view by double-clicking it, and edit the options.

Vivado P&R Option File

Some of the important Vivado place-and-route options are listed in the following table, but refer to the Xilinx documentation for a definitive list of the Vivado Tcl commands and their syntax. An example of the options file that is generated after synthesis follows the table.

Option	Description
link_design	Creates a design from a netlist. Equivalent to the ISE ngdbuild command. link_design [-name <arg>] [-part <arg>] [-constrset <arg>] [-top <arg>] [-quiet] [-verbose]
opt_design	Performs logic optimizations on the input netlist. Equivalent to the ISE MAP command. All optimizations are on by default. opt_design [-sweep] [-retarget] [-propconst] [-remap] [-resynth <arg>] [-mode <arg>] [-effort_level <arg>] [-quiet] [-verbose]
place_design	Automatically places ports and cells while optimizing for timing, wire length and congestion. Equivalent to the ISE MAP command. place_design [-effort_level <arg>] [-no_timing_driven] [-quiet] [-verbose]
route_design	Routes the placed design. Equivalent to the ISE PAR command. route_design [-unroute] [-re_entrant <arg>] [-nets <args>] [-physical_nets] [-pin <args>] [-effort_level <arg>] [-no_timing_driven] [-preserve] [-delay] [-free_resource_mode] -max_delay <arg> -min_delay <arg> [-quiet] [-verbose]
set_param write_ncd.noDRC 1|0	Use it to turn off checks before writing an ncd and to enable the checks after the ncd is written. Do this to prevent the flow from erroring out because not all the I/O pads and pins are specified.
write_checkpoint	Use this to save intermediate netlists at any point in the P&R run.
report_*	Generates various kinds of reports.
BITSTREAM	Generates a bitstream even if you have errors.

Copyright © 2013 Synopsys, Inc.

```
##################################################
###     SET DESIGN VARIABLES      ###
##################################################
set DesignName  "CameraOpenSource"
set FamilyName  "VIRTEX7"
set DeviceName  "XC7VX980T"
set PackageName   "FFG1930"
set SpeedGrade  "-1"
set TopModule   "CameraOpenSource"
set PartName    "XC7VX980TFFG1930-1"
set InputMode   "EDIF"

##################################################
###     SETUP DESIGN     ###
##################################################
set_property target_part ${PartName} [current_fileset -constrset]
set_property design_mode GateLvl [current_fileset]

if {${InputMode} == "EDIF"} {
   set_property edif_top_file ${DesignName}.edf [current_fileset]
   if {[file exists ${DesignName}.edf]} {read_edif
         ${DesignName}.edf }
   if {[file exists ${DesignName}_edif.xdc]} { read_xdc
         ${DesignName}_edif.xdc }
set TopModule [find_top]
}

if {${InputMode} == "VM"} {
   if {[file exists ${DesignName}.vm]} { read_verilog
         ${DesignName}.vm }
   if {[file exists ${DesignName}.xdc]} { read_xdc
         ${DesignName}.xdc }
   set TopModule [find_top]
   set_property top ${TopModule} [current_fileset]
}

##################################################
###    RUN DESIGN     ###
##################################################
link_design
if {[file exists "clock_groups.tcl"]} {source clock_groups.tcl}
opt_design
place_design
catch {set_param write_ncd.noDrc 1}
write_ncd -force ${DesignName}_place.ncd
catch {set_param write_ncd.noDrc 0}
route_design
```

```
#set_property BITSTREAM.General.UnconstrainedPins {Allow}
      [current_design]
catch {set_param write_ncd.noDrc 1}
write_ncd -force ${DesignName}.ncd
catch {set_param write_ncd.noDrc 0}

##################################################
###     GENERATE REPORTS     ###
##################################################
write_checkpoint -force post_place
report_utilization -file area.txt
report_utilization -slr -file slr.txt
report_timing_summary -nworst 3 -max_paths 3
write_checkpoint -force post_route
write_pcf -force ${DesignName}.pcf
report_io -file pinloc.txt
report_drc -file post_route_drc.txt
write_xdc -no_fixed_only -constraints valid -force
      ${DesignName}_post_par.xdc

##################################################
###     SAVE VIVADO PROJECT     ###
##################################################
save_project_as -force ${DesignName}
save_design -force

##################################################
###     GENERATE BITSTREAM     ###
##################################################
set_property BITSTREAM.CONFIG.OVERTEMPPOWERDOWN {Enable}
      [current_design]
set_property BITSTREAM.GENERAL.COMPRESS {True} [current_design]
      write_bitstream -force ${DesignName}.bit
```


Copyright © 2013 Synopsys, Inc.

Running Vivado Incrementally

The incremental Xilinx Vivado flow saves runtime by preserving and reusing placement and routing information for parts of the design that have not changed, and only rerunning placement and routing for the modified parts of the design. The Vivado incremental flow is the equivalent of the ISE Smart-Guide flow, which is described in *Xilinx SmartGuide Flow*, on page 870.

Use this flow when you are trying to save runtime as you iterate through and make changes. This flow is best suited to designs with long place-and-route runtimes. The initialization overhead of the incremental flow might not gain you a lot in designs with shorter runtimes.

The following procedure provides details of the incremental Vivado flow:

1. Complete an initial synthesis run, followed by place-and-route with Vivado.
   - Set up your design for running with Vivado, making sure that you use version 2012.4 or later. See *Running Xilinx Vivado Place-and-Route*, on page 852 if you need details.
   - In the Implementation Results tab of the Implementation Options dialog box, enable the Use Vivado for Place and Route option.

   - Add a place-and-route job to the synthesis implementation and enable the Incremental Place and Route option in the Add New Place and Route Task dialog box.

Copyright © 2013 Synopsys, Inc.

```
130 Flow is Incremental
131 post_route.dcp does not exist. Running Placer
132 INFO: [Common 17-347] Attempting to get a license for feature 'Implementation' and/or device 'xc7vx485t'
133 WARNING: [Common 17-348] Failed to get the license for feature 'Implementation' and/or device 'xc7vx485t'
134 INFO: [Common 17-347] Attempting to get a license for feature 'Rodin_Implementation' and/or device 'xc7vx485t'
135 INFO: [Common 17-349] Got license for feature 'Rodin_Implementation' and/or device 'xc7vx485t'
136 Running DRC as a precondition to command place_design
137 INFO: [Drc 23-27] Running DRC with 4 threads
138
139 Starting Placer Task
140 Effort Level for place_design : MEDIUM (Default)
141 INFO: [Place 30-611] Multithreading enabled for place_design using a maximum of 4 CPUs
142
143 Phase 1 Mandatory Logic Optimization
144 Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 766.516 ; gain = 0.000
145 Phase 1 Mandatory Logic Optimization | Checksum: f08b97db
146
147 Time (s): cpu = 00:00:00.04 ; elapsed = 00:00:00.05 . Memory (MB): peak = 766.516 ; gain = 2.355
148
149 Phase 2 Build SLR Info
150 Phase 2 Build SLR Info | Checksum: f08b97db
151
152 Time (s): cpu = 00:00:00.14 ; elapsed = 00:00:00.13 . Memory (MB): peak = 770.238 ; gain = 7.078
153
154 Phase 3 Add Constraints
155 Phase 3 Add Constraints | Checksum: f08b97db
156
157 Time (s): cpu = 00:00:00.14 ; elapsed = 00:00:00.16 . Memory (MB): peak = 770.238 ; gain = 7.078
158
```

– Set other options and constraints and run synthesis and place-and-route. The tool generates a vivado.log file under the pr_1 directory. The log reports the runtime, which in the example below is 00:00:00.14.


```
130 Flow is Incremental
131 post_route.dcp does not exist. Running Placer
132 INFO: [Common 17-347] Attempting to get a license for feature 'Implementation' and/or device 'xc7vx485t'
133 WARNING: [Common 17-348] Failed to get the license for feature 'Implementation' and/or device 'xc7vx485t'
134 INFO: [Common 17-347] Attempting to get a license for feature 'Rodin_Implementation' and/or device 'xc7vx485t'
135 INFO: [Common 17-349] Got license for feature 'Rodin_Implementation' and/or device 'xc7vx485t'
136 Running DRC as a precondition to command place_design
137 INFO: [Drc 23-27] Running DRC with 4 threads
138
139 Starting Placer Task
140 Effort Level for place_design : MEDIUM (Default)
141 INFO: [Place 30-611] Multithreading enabled for place_design using a maximum of 4 CPUs
142
143 Phase 1 Mandatory Logic Optimization
144 Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 766.516 ; gain = 0.000
145 Phase 1 Mandatory Logic Optimization | Checksum: f08b97db
146
147 Time (s): cpu = 00:00:00.04 ; elapsed = 00:00:00.05 . Memory (MB): peak = 766.516 ; gain = 2.355
148
149 Phase 2 Build SLR Info
150 Phase 2 Build SLR Info | Checksum: f08b97db
151
152 Time (s): cpu = 00:00:00.14 ; elapsed = 00:00:00.13 . Memory (MB): peak = 770.238 ; gain = 7.078
153
154 Phase 3 Add Constraints
155 Phase 3 Add Constraints | Checksum: f08b97db
156
157 Time (s): cpu = 00:00:00.14 ; elapsed = 00:00:00.16 . Memory (MB): peak = 770.238 ; gain = 7.078
158
```


The tool saves routed design information to a checkpoint file called post_route.dcp.

2. Make changes to the design.

   This flow offers the most runtime savings when the design has only small changes and is very similar to the reference design. It is most effective when design changes amount to less than 10 percent of the design. If many changes are made to critical path placement and routing, this incremental is not as useful, as much of the placement and routing cannot be reused.

3. Rerun synthesis and place-and-route with the Incremental Place and Route option enabled.

   Vivado runs initial placement by matching objects in the current design against objects from the previous run. The post_route.dcp file from the previous run is used as a reference. During routing, Vivado removes routing to objects that are no longer in the design. It uses unchanged routing from the checkpoint database.

   If you check the vivado.log file, it reports that it is running incrementally, and also reports the runtime. In this example, the incremental runtime is 00:00:00.03.

   If you check the run_vivado.tcl file under the pr_1 directory, it too reports that the incremental flow was run:

Copyright © 2013 Synopsys, Inc.

File Edit View Project Import Run Analysis HDL-Analyst Options Window Tech-Support Web Help
2 ##
3 ### SET DESIGN VARIABLES ###
4 ##
5 set DesignName "eight_bit_uc"
6 set FamilyName "VIRTEX7"
7 set DeviceName "XC7VX485T"
8 set PackageName "FFG1157"
9 set SpeedGrade "-1"
10 set TopModule "eight_bit_uc"
11 set PartName "XC7VX485TFFG1157-1"
12 set InputMode "EDIF"
13 set Flow "Incremental" ;#Flow can be set to "Incremental" or "Standard"
14
15
16 ##
17 ### SETUP DESIGN ###
18 ##
19 set_property target_part ${PartName} [current_fileset -constrset]
20 set_property design_mode GateLvl [current_fileset]
21
22 if {${InputMode} == "EDIF"} {
23 set_property edif_top_file ${DesignName}.edf [current_fileset]
24 if {[file exists ${DesignName}.edf]} { read_edif ${DesignName}.edf }
25 if {[file exists ${DesignName}_edif.xdc]} { read_xdc ${DesignName}_edif.xdc }
26 if {[file exists ${DesignName}_floorplan.xdc]} { read_xdc ${DesignName}_floorplan.xdc }
27 set TopModule [find_top]
28 }
29
30 if {${InputMode} == "VM"} {
31 if {[file exists ${DesignName}.vm]} { read_verilog ${DesignName}.vm }
32 if {[file exists ${DesignName}.xdc]} { read_xdc ${DesignName}.xdc }
eight_bit_uc.prj*
vivado.log
prgm_cntr.v
adder.v
run_vivado.tcl
Information
Complete: Place & Route (pr_1) on eight_bit_uc|Incrementalpar_enabled_fast_synth_edif
Complete: Place and Route Logical Synthesis on eight_bit_uc|Doc_Incrementalpar_enabled
Complete: Logic Synthesis on eight_bit_uc|Doc_Incrementalpar_enabled
Complete: Synthesis and Physical Plus on eight_bit_uc|Doc_Incrementalpar_enabled
TCL Script
Messages

Running Xilinx ISE Incrementally

Xilinx

The Xilinx ISE place-and-route tool has two incremental flows, both of which take advantage of the compile points defined during synthesis. Both flows use compile points to run incremental place-and-route, but you use them for different purposes:

Design Goal	Xilinx Flow
Save runtime, or iterate through P&R incrementally.	SmartGuide Flow
Stabilize the design and preserve completed parts of the design. Only run incremental P&R on the parts that have changed.	Partition Flow

- Xilinx SmartGuide Flow

  Use this flow when you are trying to save runtime as you iterate through the flow and make changes. Typically, you use this earlier in the design cycle. In this flow ISE runs with SmartGuide enabled. With this setting, the tool reuses place-and-route information from the previous run for unchanged partitions, and only places and routes the partitions that were modified. To get even more runtime savings, combine the SmartGuide flow with automatic compile compile points, fast synthesis, and multiprocessing during synthesis. For complete details on SmartGuide, see the Xilinx documentation.

- Xilinx Partition Flow

  Use this flow when you are trying to converge, and design preservation and stability are your primary goals. The same manual RTL compile points defined during synthesis are maintained through place and route. If a compile point is modified, the Xilinx tool reruns P&R for only those partitions that have been modified; all others are preserved. To reduce runtime while running this flow, use multiprocessing.

  For versions prior to ISE 12.1, the Partition flow involved manual steps. For a description of this flow, see *Xilinx Partition Flow for Versions Before ISE 12.1*, on page 878.

Copyright © 2013 Synopsys, Inc.

Xilinx SmartGuide Flow

Use the SmartGuide flow for incremental changes when you are trying to reduce runtime, as described in *Running Xilinx ISE Incrementally*, on page 869. Use the SmartGuide option when you need to make minor changes to a design: make HDL changes to a small number of modules (logic changes of approximately 10% or less), move pin locations, change attributes, or modify timing constraints. Do not use this flow if you have changes on the critical path.

The following procedure describes the details of running synthesis and the SmartGuide flow:

1. Make sure to use Xilinx ISE 12.1 or later version of the P&R tool.
2. Set up the synthesis project.
   - Create compile points, as described in *Synthesizing Compile Points*, on page 1109. You can create automatic or manual compile points, but inferring automatic compile points speeds up runtime.
   - To speed up runtime, use multiprocessing. In the Synplify Premier tool you can additionally specify Fast Synthesis mode to reduce runtime even further.
   - Target a supported Xilinx device.
   - Go to the Implementation Results panel of the Implementation Options dialog box, and define the top-level design name for the project in the Result Base Name field. This name must be identical to the top-level module file name.

– In the Device tab of the Implementation Options dialog box, disable Use Xilinx Xflow. SmartGuide uses the xtclsh flow, so this option must be disabled. If you do not do so, you see a popup warning later when you specify SmartGuide.

– Set SmartGuide for the Xilinx P&R tool, by clicking the Add P&R Implementation button in the Project view and selecting the Smart Guide option in the resulting Add New Place and Route Task dialog box.

Copyright © 2013 Synopsys, Inc.

3. Do an initial run of synthesis, placement, and routing.
4. Implement small changes to the design.
5. Rerun synthesis and P&R.

   When you rerun synthesis and P&R with the SmartGuide option enabled, ISE uses the `ncd` file from the previous run as a guide. It preserves any unchanged components and incrementally places and routes those that were modfied. SmartGuide can provide significant runtime improvements, as the entire design does not need to be rerun.

Xilinx Partition Flow

Like the SmartGuide flow, the Partition flow is based on compile points, but it is focused on achieving design stability. It lets you preserve and lock down parts of the design as you iterate through the design, because it only reruns the modified modules. Partitions that have not changed are preserved from the previous implementation. Note that this flow is based on manual compile points, for better control.

The figure below summarizes the process, which is then explained in more detail in the subsequent steps.

1. Make sure to use version 12.1 or later of the Xilinx ISE place-and-route tool.

   If you have an older version of ISE, use the procedure described in *Xilinx Partition Flow for Versions Before ISE 12.1*, on page 878.

2. Set up a project as usual, and set implementation options.

Copyright © 2013 Synopsys, Inc.

– Select a Xilinx target device that the Xilinx Partition flow supports.

– To use the partition flow, enable Use Xilinx Partition Flow from the Device tab. This automatically enables the Use Xilinx Xflow option.

– For the Synplify Premier tool, set these additional options: disable all the netlist prototyping tools options on the GCC & Prototyping Tools tab, and disable the Physical Synthesis option. If you do not disable Physical Synthesis, the tool removes the XML partitions to run physical synthesis.

3. Compile the design, and define manual compile points in the top-level constraint file.

   – Click the Compile Points tab, and set compile points, as described in *Defining Manual Compile Points*, on page 1117.

   – Set the compile point type to hard, locked or locked, partition to run the Xilinx Partition flow. The following example shows three compile points set as locked compile points: ALU, comb_logic, and mult.

– Create a compile point constraint file for each compile point. This is especially important if you use a bottom-up flow.

– Set the clock constraint for the compile point. This can be the same as the top level.

– Save the constraint file.

4. For best results, follow these additional guidelines:

   – Register the compile point I/Os to mitigate QoR loss.

Copyright © 2013 Synopsys, Inc.

- If you are using a bottom-up flow, disable I/O insertion and clock buffer insertion in lower-level compile points.

5. Set up the P&R tool to run automatically after synthesis completes, by doing the following:
   - Click on the Add P&R Implementation button in the Project view to create an initial P&R implementation. When you re-run this P&R implementation, the incremental results are written to this directory.
   - Select the P&R implementation you want to run and enable Run Place & Route following synthesis in the dialog box.

   You can also run place and route manually as a separate process after synthesis is done. To do this, first run the Tcl command for the stand-alone XML converter first, as described in *Using Compile Points in a Standalone Xilinx Partition Flow Run*, on page 877.

6. Run synthesis and place and route.
   - Click Run to start the process.

     The tool first synthesizes the the compile points and then synthesizes the top level. See *Synthesizing Compile Points*, on page 1109 for details about this process. It forward-annotates the compile point information along with a timestamp for each compile point in the xpartition.pxml file. If you have set it up, the ISE tool runs automatically after synthesis is complete, and honors the compile points as partitions.
   - After P&R, check generated reports and the log file for any compile point or Xilinx Partition flow messages.

7. To run the design incrementally, do the following:
   - Make the needed design changes to the source or constraint files.
   - Do not make any changes to the Xilinx ISE P&R environment.
   - Click Run to run incremental synthesis and incremental P&R in the Xilinx tool.

   The synthesis software runs incrementally, only resynthesizing compile points whose logic, implementation options, or timing constraints have changed.

   The ISE tool also runs incrementally. It compares the partition timestamps in the xpartition.pxml file for the previous and current implementations, and preserves partitions that have not changed. It only

reruns P&R on partitions that have been resynthesized. The PAR log file reports the details:

```
Partition Implementation Status
-------------------------------

Preserved Partitions:
   Partition "/top"
   Partition "/top/A"

Implemented Partitions:
   Partition "/top/D"
   Attribute STATE set to IMPLEMENT
```

Using Compile Points in a Standalone Xilinx Partition Flow Run

When you run the P&R tool separately, from outside the synthesis tool, you must first run the sxml2pxml Tcl command for the standalone XML converter.

1. Set up the compile point design and synthesize as usual.
2. After synthesis, run the sxml2pxml Tcl command in one of these ways:
   - Open the Synplify Pro or Synplify Premier GUI, then type the command in the TCL Script window.
   - Write this command into a Tcl script file, then run the Synplify Pro or Synplify Premier tool in batch mode.
   - Run the Synplify Pro or Synplify Premier tool in batch mode, specifying the command with the -tclcmd option.

   This command generates a Xilinx xpartition.pxml file with compile point information for the specified implementation directory and design. For the Tcl command syntax, see *sxml2pxml Tcl Command*, on page 1716.
3. Place and route the design with the Xilinx ISE tool.

Copyright © 2013 Synopsys, Inc.

Xilinx Partition Flow for Versions Before ISE 12.1

For older versions of the software (before ISE12.1), use the following procedure to use compile points with the Xilinx Partition flow. It is similar to the newer Partition flow described in *Xilinx Partition Flow*, on page 873, but you have to execute some of the steps manually.

1. If necessary, set the SYN_XILINX_PR_USE_XTCLSH environment variable to 1.

   This is the default in the latest releases of the synthesis tools, so you do not have to set it. This setting ensures that the tool generates the `run_ise.tcl` script file after synthesis. For subsequent synthesis runs, the tool recognizes the script file and does not overwrite it.

2. Set up the design.
   - Set up the project file and define manual compile points.
   - Target a Xilinx device that the Partition flow supports.
   - Specify implementation options and constraints, as usual.

3. Click Run to synthesize the design.

   The synthesis tool generates an EDIF file where the compile points are defined with "PARTITION" properties. Each compile point also includes a timestamp for when the module was last synthesized. Later, the place and route tool uses the time stamp as the basis for comparison to determine which modules need to be incrementally updated.

4. Run place and route.
   - Make sure you set the system path variables for the Xilinx place-and-route tool
   - Run the P&R project `tcl` script using this command: xtclsh.exe.

5. Perform the following tasks when ISE placement and routing completes.
   - Check generated reports.
   - Make any necessary major design changes to the source or constraint files and place and route the design.

6. Go back to the synthesis tool and re-synthesize the modified design.

   The synthesis tool only resynthesizes and optimizes the updated modules. In the EDIF file generated for this synthesis run, the tool does

Copyright © 2013 Synopsys, Inc.

not change the timestamp of a compile point if it has not changed since the previous run; it preserves the old timestamp. Updated modules get a new timestamp.

7. Rerun the ISE `tcl` script and place and route the design.

   The Xilinx tool compares the compile point timestamps to the corresponding timestamps from the previous run, and incrementally places and routes only those blocks with updated timestamps. It leaves the other blocks untouched. The EDIF file also specifies whether an updated compile point needs to be re-placed and re-routed, or only re-routed. The default specifies incremental placement and routing.

Copyright © 2013 Synopsys, Inc.

Working with the Identify Tools

The Synopsys Identify tool set is a dual-component system that is a valuable part of the HDL design flow process. The system consists of the Identify instrumentor and Identify debugger.

- The Identify instrumentor allows you to select your design instrumentation at the HDL level and then create an on-chip hardware probe.
- The Identify debugger interacts with the on-chip hardware probe and lets you do live debugging of the design.

The combination of these tools allows you to probe your HDL design in the target environment. The combined system allows you to debug your design faster, easier, and more efficiently.

The Synplify, Synplify Pro, and Synplify Premier synthesis tools have integrated the Identify instrumentor into the synthesis user interface. This section describes how to take advantage of this integration and use the Identify instrumentor:

- Launching from the Synplify Pro or Synplify Premier Tool, on page 880
- Launching from the Synplify Tool, on page 882
- Handling Problems with Launching Identify, on page 884
- Using the Identify Tool, on page 885
- Using Compile Points with the Identify Tool, on page 887

Launching from the Synplify Pro or Synplify Premier Tool

Define a Synplify Pro or Synplify Premier project that you can pass to and launch in the Identify instrumentor. For the Synplify Pro and Synplify Premier tools, you must create an Identify implementation in order to run the Identify instrumentor. If you already have an Identify implementation, open it and use the Identify tool as described in *Using the Identify Tool*, on page 885.

Do the following to add an Identify implementation:

1. In the synthesis interface, open the design you want to debug.

2. Do one of the following tasks to add an Identify implementation:
   - With the project implementation selected, right-click and select New Identify Implementation from the pop-up menu.

   - Select Project->New Identify Implementation.

   An Implementation Options dialog box appears where you can set the options for your implementation. Note that the options apply only for logic synthesis and not for physical synthesis. An Identify implementation is created.

3. To run Identify instrumentor, select the Launch Identify Instrumentor icon ( ) in the toolbar or select Run->Identify Instrumentor.

Copyright © 2013 Synopsys, Inc.

The Identify interface opens. You can now use the Identify tool as described in *Using the Identify Tool*, on page 885 For complete details, consult the Identify documentation.

If you run into problems while launching the Identify instrumentor, refer to *Handling Problems with Launching Identify*, on page 884.

Launching from the Synplify Tool

The Synplify synthesis tool does not support multiple implementations. The following procedures describe how to launch an Identify implementation from Synplify, and how to modify an existing implementation.

Creating a New Identify Implementation in Synplify

1. Make sure that your PATH environment variable points to the Identify bin directory.
2. Create or open a project in Synplify.
3. Right-click on the implementation and select New Identify Implementation from the popup menu.
4. Set any implementation options and close the dialog box; dismiss the multiple implementation warning.
5. Select Options->Configure Identify Launch to display this dialog box:

6. Check the Identify installation. If the Use current Identify Installation field entry in the dialog box is not correct, either:
   - click the Locate Identify Installation button and enter the path to the Identify installation directory. Use the browse button if necessary.
   - set the SYN_IDENTIFY_EXE environment variable to point to the Identify installation. This path is the directory path displayed in the Use current Identify Installation field. You must restart the synthesis tool whenever you change the environment variable setting.

   Select the appropriate license option and click OK to launch the Identify instrumentor for the new implementation.

7. Instrument the design as required, save the instrumentation, and exit the Identify instrumentor. See *Using the Identify Tool*, on page 885 for an overview, or the Identify documentation for details.

8. Synthesize the instrumented implementation (rev_*n*_identify) in Synplify (the schematic will show the added IICE circuitry).

   After the design has been synthesized, place and route your design. Program the device, install the device in the target system, and complete the cable interface. You can now run the Identify debugger on the instrumented design (*designName*.prj) to verify correct operation.

Copyright © 2013 Synopsys, Inc.

Modifying or Re-instrumenting an Existing Design

To modify or re-instrument an existing design:

1. Load the project containing the Identify implementation into Synplify.
2. From the Synplify menu bar, select Run->Run TCL Script.
3. Navigate to the Synplify lib directory and run (open) the relaunch_identify.tcl script to launch the Identify instrumentor.
4. Re-instrument the design as required, save the instrumentation, and exit the Identify instrumentor.
5. Re-synthesize the instrumented implementation (rev_*n*_identify) in Synplify.

After the design has been re-synthesized, place and route your design. Program the device and reinstall the device in the target system. You can now rerun the Identify debugger on the instrumented design (*designName*.prj) to verify correct operation.

Handling Problems with Launching Identify

If you have not installed Identify correctly, you might run into problems when you try to launch it from the synthesis tools. The following describe some situations:

- If the Launch Identify Instrumentor icon ( )and the Run->Identify Instrumentor menu command are inaccessible, you are either on an unsupported platform or you are using a technology that does not support this feature.
- If you have the Identify software installed but the synthesis application cannot find it, select Options->Configure Identify Launch.

In the resulting dialog box, either:

– check the Use Current Identify Installation entry. This entry is set by the SYN_IDENTIFY_EXE environment variable to point to the Identify installation. If this path is incorrect, change the environment variable setting and restart the synthesis tool. button and specify the correct location in the Locate Identify Installation field. You can use the Browse button to open the Select Identify Installation Directory dialog box and navigate to your current Identify installation directory.

– click the Locate Identify Installation button and specify the correct location in the corresponding field. Use the browse button to open the Select Identify Installation Directory dialog box and navigate to your current Identify installation directory.

Using the Identify Tool

This procedure provides an overview of how to use the Identify instrumentor. For detailed information about the tool, refer to the Identify RTL debugger documentation.

1. The Identify instrumentor software interface opens, with an Identify project automatically set up for the design to be instrumented and debugged (IICE tab). The following figure shows the main project window.

Copyright © 2013 Synopsys, Inc.

2. Do the following in the Identify instrumentor interface:
   - Instrument the design. For details of using the Identify instrumentor, refer to the Identify RTL debugger documentation.
   - Save the instrumented design.

   The Identify instrumentor tool exports the instrumented design to the synthesis software. It creates an instrumentation subdirectory under your synthesis working directory called *designName_instr*, which contains the following:
   - A synthesis project file
   - An instr_sources subdirectory for the instrumented HDL files
   - Tcl scripts for loading the instrumented design
3. Return to the synthesis interface and view the instrumented design that contains the debugging logic.

- In the synthesis interface, open the project file for the instrumented design, which is in the instr_sources subdirectory listed in the Implementations Results view for your original synthesis project.
- Synthesize the design.
- Open the RTL view to see the inserted debugging logic.

4. Place and route the instrumented design after synthesis.
5. Use the Identify debugger tool to debug the instrumented design.

Using Compile Points with the Identify Tool

You can use compile points to run incrementally. This can reduce runtime while running synthesis, and also while running the Identify flow. The following figure illustrates this:

Copyright © 2013 Synopsys, Inc.

When you use Identify instrumentation, the tool creates extra IICE logic at the top level of the design and the corresponding interface to the signals that need to be debugged. If you define compile points, the tool need only rerun the compile points that have changed because of the insertion of this logic. On subsequent runs, it can incrementally re-instrument only those compile points where there are instrumentation changes or design modifications.The following procedure describes the steps to follow to implement the flow and take advantage of incremental synthesis and instrumentation:

1. Create a synthesis implementation with compile points.
2. Set up the Identify implementation:
   - Generate the Identify implementation by right-clicking the FPGA synthesis implementation and selecting New Identify Implementation from the popup menu.
   - Copy the compile point subdirectories manually to the new Identify implementation directory.
3. Run the tools.
   - Run synthesis.
   - Before running the Identify tool, enable the top-level constraint file and all compile point constraint files in the Identify implementation.
   - Instrument the design. The tool inserts additional logic for instrumentation.
4. Resynthesize the design.

   The tool runs incrementally, only resynthesizing the compile points affected by the inserted instrumentation logic. If you make any other design changes, the tool incrementally synthesizes the affected compile points.
5. Rerun instrumentation.

   The tool runs incrementally, and only re-instruments the affected compile points.

Copyright © 2013 Synopsys, Inc.

Netlist Editing

Netlist editing capabilities are useful for:

- Implementing engineering change orders (ECOs)
- Rerouting a fast clock from an external source to an internal (DCM) source
- Inserting IP blocks in an RTL netlist (`srs` file)

With netlist editing, you do not have to modify the HDL to perform small changes to the netlist. Netlist editing includes the ability to insert or "stitch" IP blocks into the design at a desired level of hierarchy by connecting the blocks to ports and nets within the core design.

In the Synplify Premier tool flow, netlist editing can be used at the RTL level. You can add other RTL views (`srs` files) or primitives to the RTL view of the design to be synthesized.

Netlist editing support can only be used within the top-down, board-level mapping (i.e., pressing the Run button) flow.

The remainder of this section includes the following information:

- RTL-Level Flow
- Netlist Editing Commands, on page 891

RTL-Level Flow

Netlist editing commands are written to a `tcl` file; you must include this file in your project using Add Prototype file on the GCC & Prototyping Tools tab of the Implementation Options dialog box. Nfilter reads these commands and performs the netlist editing. You may need several iterations through this process until the desired results are achieved.

The basic RTL-level flow is:

1. Load the database (i.e., ip.srs or ip.srl file) with the existing netlist from where the new netlist will be created.
2. Create and compile the project with the IP or modules you want to insert into the main design to generate the ip.srs or ip.srl file.

Copyright © 2013 Synopsys, Inc.

3. Apply the netlist editing commands in the `tcl` file.
4. Run nfilter on the project with the source, ip.srs, and `tcl` files to generate the `srs` file.
5. Run the mapper on the generated `srs` file.

Alternatively, you can load the Verilog database; nfilter does not understand the Verilog database, and you must load the *.syn file available in the lib/xilinx or lib/altera folder. Apply the netlist editing commands in the .tcl file. You can use the views directly from the xilinx.syn or altera.syn file to create instances. Run nfilter and then run the mapper.

RTL Flow Examples

The following example adds an RTL (`srs`) IP block and/or library primitives to a project's RTL View (`srs`).

The following lines in the `prj` file add and enable an RTL netlist editing script for the project:

```
add_file -tcl "netlist_edit.tcl"
set_option -nfilter_user_path "netlist_edit.tcl"
```

Sample .tcl file contents:

```
#Set the module to edit
define_current_view top
create_port {out[0:3]} -direction out
create_net {out[0:3]}
create_instance inst_A nle2_add_select
connect_net {out[0:3]} {inst_A.out[0:3]} {p:out[0:3]}
connect_net {d1[0:3]} {inst_A.a[0:3]}
connect_net {d2[0:3]} {inst_A.b[0:3]}
connect_net {d3[0:3]} {inst_A.c[0:3]}
connect_net {d4[0:3]} {inst_A.d[0:3]}
connect_net {d5[0:3]} {inst_A.e[0:3]}
connect_net clk1 inst_A.clk
connect_net sel inst_A.s0
connect_net reset inst_A.s1

#Insert an inv pair from the technology library (unisim.v)
#on the net driven by top_mult.out2[0]
insert_buffer -inverter_pair {top_mult.out2[0]} INV
```

Diagnosing Problems

Problems can occur when adding entries to the .tcl files. Syntax errors are usually reported, but incorrect connections may not be detected until much later in the design flow. The syntax error

```
Could not find pin "pathName.pinName" in module "modName"
```

indicates an incorrect path or pin name. Check that the path/pin names and letter case are correct.

Netlist Editing Commands

Netlist editing commands typically apply to the current view. Note that all path names used in netlist editing commands are relative to the module of the current view and that the current view must be unique (if the view is not unique, a modification in the current view affects all modules of the same view).

The supported netlist commands are listed in the table below. Complete syntax and command description are included in *Netlist Editing Commands*, on page 1733.

Note: Write netlist editing commands to a `tcl` file. You must include this file in your project using Add Prototype file on the GCC & Prototyping Tools tab of the Implementation Options dialog box. Nfilter reads these commands and performs the netlist editing.

connect_net	copy_view	create_cell
create_cgm	create_instance	create_net
create_port	create_process_hierarchy	define_current_view
disconnect_connector	disconnect_net	get_net
insert_buffer	load_database	optimize_netlist
pop_feedthroughs	propagate_constants	remove_buffer
remove_instance	remove_net	remove_port
reset_generate	set_property	swap_instance
tie_net	tie_pin	

Copyright © 2013 Synopsys, Inc.

All arguments used with the netlist editing commands must correspond to an existing object in the design or to a constant. The possible objects are:

- connector (pin or bit port)
- net
- instance
- view
- library file
- data-base file

Each of the above objects can be identified with a simple name or hierarchical path name. Additionally, a name can be prefixed by a qualifier to further define the object type to the command. These qualifiers are:

- **t:** – denotes a pin name
- **p:** – denotes a port name
- **i:** – denotes an instance name
- **v:** – denotes a view name

In some commands (for example, in single-argument commands), the qualifiers can be omitted. In multi-argument commands, the arguments must be separated by spaces.

The following are examples of object names:

i:*instName* – denotes an instance

i:*instPath.instName* – denotes an instance within the netlist of the instance *instPath*

t:*instName.pinName* – denotes an instance pin

p:*portName* – denotes a top-level port

p:*instName.portName* – denotes a port of within the netlist of the instance *instName*

n:*netName* – denotes a net

n:*instName.netName* – denotes a net within the netlist of the instance *instName*

v:*viewName* – denotes a view

VIF Formal Verification Flows

During synthesis, the Synplify Pro tool performs several sequential optimizations and design transformations to improve delay and area. These transformations make it difficult for a formal verification tool to match registers in the result netlist with the corresponding registers in the source HDL (a prerequisite for verifying equivalence). To solve this, the Synplify Pro software provides a Tcl file interface that lets you integrate with verification tools. This proprietary format is called the Verification Interface Format or VIF. This feature is currently available for only Altera technologies.

This section describes the following:

- Overview of the VIF Flow, on page 893
- Generating VIF Files, on page 894
- Using VIF with Cadence Conformal, on page 897
- Handling Equivalency Check Failures, on page 898

Overview of the VIF Flow

The Synplify Pro VIF flow is based on a Tcl file generated during synthesis. This file has a `vif` extension. It contains a vendor-independent list of the design transformations performed during synthesis so that the verification tool can do equivalence checking and match up the post-synthesis registers with the original golden netlist.

Copyright © 2013 Synopsys, Inc.

The following diagram summarizes the two ways in which you can use the .vif file as input. Subsequent sections show you how to use the generated `vif` file to verify your logic with the Conformal tools.

Generating VIF Files

1. In the Synplify Pro interface, select Project->Implementation Options and set the following on the Device tab:

   – Set Technology to an Altera family that supports the VIF flow.

   – Disable optimizations like Retiming and Pipelining on the Options tab and Clock Conversion on the GCC or GCC & Prototyping Tools tab. Disabling these options is recommended because the verification tool requires a

one-to-one correspondence between the design objects in the reference and implementation designs to successfully complete, and the optimizations make it hard to verify this. The downside to disabling the optimizations is performance loss.

– Enable Verification Mode. The equivalent `prj` file syntax is set_option -verification_mode 1. This is an optional step that disables optimizations that can not be easily verified, like the inference of resettable SRLs. Enabling Verification Mode could result in sub-optimal area or performance results, because the optimizations were not run.

  You enable Verification Mode to make it easy for the verification tool to synchronize registers. Sequential optimizations are hard to verify because registers are moved or optimized away. For a list of VIF optimization commands, see step 4, below. Verification Mode is different from the Disable Sequential Optimizations option. The former disables many FPGA optimizations, where the latter only disables sequential optimizations.

  Note: Verification Mode requires that the Write Mapped Verilog Netlist option be enabled as well. See step 2.

2. Go to the Implementation Results tab and do the following:

   Enable Write Mapped Verilog Netlist. This option generates the `vif` file. The equivalent project file syntax is set_option -write_verilog 1. When enabled, this option creates a verif directory under the results/implementation directory. The verif directory contains the *design*.vif file.

3. Synthesize the design as usual.

   The Synplify Pro software generates the `vif` file and stores it in the project/verif directory.

4. Go to the verif directory and check the `vif` file to see how the optimizations were handled.

Copyright © 2013 Synopsys, Inc.

The following table lists the VIF commands used to map some synthesis optimizations. For details of the command syntax, refer to *VIF Commands*, on page 1726 in the *Reference Manual*.

Optimization	VIF Command
FSM register mapping	vif_set_fsmreg
FSM state encoding	vif_set_state_map
Register merging	vif_set_merge
Register replication	vif_set_equiv
Pruning of duplicate registers	vif_set_constant, vif_set_transparent
Black boxes for undefined modules	vif_set_map_point
Port direction changes	vif_set_port_dir

5. Use the `vif` file as input to any formal verification tool that supports a Tcl interface. Use one of the following methods:
   - If you are using the Cadence Conformal tool, run the translation script vif2conformal.tcl which is in the *install dir*/lib directory (see *Using VIF with Cadence Conformal*, on page 897 for details). This translates the .vif file commands to commands for the Conformal tool.
   - If you are using another verification tool that does not directly support VIF commands, create a script that translates the `vif` file commands to native Tcl commands.
   - If you are using a verification tool that supports the VIF commands in its Tcl framework, use the file directly.
6. In the verification tool, use the information from the `vif` file along with the synthesis output when you check logic equivalence against the golden netlist.

Using VIF with Cadence Conformal

The Synplify Pro software includes Tcl scripts for use with the Cadence Conformal tool. You can convert `vif` files manually or automatically.

Converting VIF Manually

The following procedure describes how to convert `vif` files manually.

1. Source the vif2conformal.tcl file by typing one of the following commands in the Synplify Pro Tcl window:

   `source` *synplify_pro_install_dir*`/lib/vif2conformal.tcl`

   or

   `source $LIB/vif2conformal.tcl`

2. In the Tcl window, navigate to the verification folder containing the *design*.vif file, and type the following command:

   `vif2conformal` *design*`.vif`

   The vif2conformal.tcl script runs on the *design*.vif file and translates the information into Conformal side files (*.vtc, *.vsc, *.vmc, and so on). You can now run Conformal using these files.

Automating VIF Conversion with Synhooks

You can create a script using the synhooks.tcl file (see *Automating Flows with synhooks.tcl*, on page 1146) to automate the generation of verification files. An example of this file, synhooks_for_vif2conformal.tcl, is located in the *install_dir*/examples directory.

The synhooks_for_vif2conformal.tcl Tcl script sets your environment to automatically convert the Synplify Pro generated `vif` file to Conformal-specific side files at the end of each synthesis run. Use either of the following methods to convert your files:

- Set the environment variable SYN_TCL_HOOKS to point to the synhooks_for_vif2conformal.tcl file. For example:

  SYN_TCL_HOOKS=*install_dir***examples/synhooks_for_vif2conformal.tcl**

Copyright © 2013 Synopsys, Inc.

- Source the synhooks_for_vif2conformal.tcl file in the Synplify Pro Tcl window to set up automatic conversion. For example:

  % **source** *install_dir***examples/synhooks_for_vif2conformal.tcl**

  For this method, you must source the synhooks_for_vif2conformal.tcl file every time you start a new project; otherwise the tool is reopened. The automatic conversion setup is lost once you close a Synplify Pro project or restart the tool.

Handling Equivalency Check Failures

If your design fails the equivalency check, try the following tips and techniques to debug the results.

- Check the log file report and fix the errors reported.
- Check the optimization mapping in the `vif` file. See step 4 of *Overview of the VIF Flow*, on page 893 for a list of commands.

Simulating with the VCS Tool

The Synopsys VCS® tool is a high-performance, high-capacity Verilog simulator that incorporates advanced, high-level abstraction verification technologies into a single, open, native platform. You can launch this simulation tool from the synthesis tools on Linux and Unix platforms by following the steps below. The VCS tool does not run under the Windows operating system.

1. Set up the tools.
   - Install the VCS software and set up the $VCS_HOME environment variable to define the location of the software.
   - Set up the place-and-route tool.
   - In the synthesis software, either select Run->Configure and Launch VCS Simulator, or click the VCS icon.

   If you did not set up the $VCS_HOME environment variable, you are prompted to define it. The Run VCS Simulator dialog box opens. For descriptions of the options in this dialog box, see *Configure and Launch VCS Simulator Command*, on page 274 of the *Reference Manual*.

2. Choose the category Simulation Type in the dialog box to configure the simulation options.
   - Specify the kind of simulation you want to run.

RTL simulation	Enable Pre-Synthesis
Post-synthesis netlist simulation	Enable Post-Synthesis
Post-P&R netlist simulation	Enable Post P&R

Copyright © 2013 Synopsys, Inc.

– Choose the category VCS Options in the dialog box to set options such as the following VCS commands.

To set...	Type the option in...
VLOGAN command options for compiling and analyzing Verilog, like the -q option	Verilog Compile
VHDLAN options for compiling and analyzing VHDL	VHDL Compile
VCS command options	Elaboration
SIMV command options, like `-debug`	Simulation

The options you set are written out as VCS commands in the script. If you leave the default settings the VCS tool uses the FPGA version of VCS and opens with the debugger (DVE) GUI and the waveform viewer. See the VCS documentation for details of command options.

3. If your project has Verilog files with `include statements, you must use the +incdir+ *fileName* argument when you specify the vlogan command. You enter the +incdir+ in the Verilog Compile field in the VCS Options dialog box, as shown below:

Example Verilog File:

```
`include "component.v"
module Top (input a, output x);
...
endmodule
```

The syntax for the VCS commands must reflect the relative location of the Verilog files:

– If the Verilog files are in the same directory as the top.v file, specify:

```
- vlogan -work work  Top.v  +incdir+ ./
```

– If the Verilog files are in the a directory above the top.v file, specify:

```
- vlogan -work work Top.v +incdir+ ../include1 +incdir+
  ../ include2
```

– If the Verilog files are in directories below and above the top.v file, specify:

```
- vlogan -work work Top.v +incdir+ ./include_dir1
  +incdir../include_dir2
```

4. Specify the libraries and test bench files, if you are using them.

Copyright © 2013 Synopsys, Inc.

- To specify a library, click the green Add button, and specify the library in the dialog box that opens. Use the full path to the libraries. For pre-synthesis simulation, specifying libraries is optional.

- For post-synthesis and post-P&R synthesis, by default the dialog box displays the UNISIM and SIMPRIM libraries in the P&R tool path. You can add and delete libraries or edit them, using the buttons on the side. To restore the defaults, click the Verilog Defaults or VHDL Defaults button, according to the language you are using.
- If you have test bench files, choose the category Test Bench Files in the dialog box to specify them. Use the buttons on the side to add, delete, or edit the files.
- For Xilinx designs, choose the category Vendor Version to specify the ISE version to use, 10.3 or 11.2. The ISE 11.2 directory structure was changed for VHDL simulation libraries.

5. Specify the top-level module and run directory.
   - Choose the category Top Level Module in the dialog box to specify the top-level module or modules for the simulation.
   - If necessary, choose the category Run Directory near the bottom of the dialog box to edit the default run directory listed in the field. The default location is in the implementation results directory.
6. Generate the VCS script.
   - To view the script before generating it, click the View Script button on the top right of the dialog box. A window opens with the specified VCS commands and options.
   - To generate the VCS script, click Save As, or run VCS by clicking the Run button in the upper right. The tool generates the XML script in the directory specified.

7. To run VCS from the synthesis tool interface, do the following:
   - If you do not already have it open, open the Run VCS Simulator dialog box by clicking the VCS icon.
   - To use an existing script, click the Load From button on the lower right and select the script in the dialog box that opens. Then click Run in the Run VCS Simulator dialog box.
   - If you do not have an existing script, specify the VCS options, as described in the previous five steps. Click Run.

   The tool invokes VCS from the synthesis interface, using the commands in the script.

Limitations

If Verilog include paths have been added to your project file, these paths are not automatically added to the VCS script. Add the Verilog include paths manually by using one of the following workarounds:

- From the Run VCS Simulator dialog box, add `+incdir+`*includePath* in the Verilog Compile options field.
- Modify the VCS script file, adding the `+incdir+`*includePath* to all or any relevant vlogan commands.

Copyright © 2013 Synopsys, Inc.

Using the VCD/Identify-Analyst Integration Tool

The Synplify Premier tool includes the VCD/Identify-Analyst Integration feature, which enhances the interaction between the VCD simulator or Identify Debugger integration and HDL Analyst tools by providing a way to watch nets annotated in the HDL Analyst from the Control Panel signal watch list.

See the following topics:

- VCD-Analyst Integration, on page 904
- Identify-Analyst Integration, on page 921
- VCS Test Bench Extraction, on page 927

This feature is available on Linux and Windows. Note the following limitations for Windows:

- A previously generated VCD file is required. The VCD file can also be created as follows:
  - In a test bench
  - If a VPD file is available, you can use the VCD utility vpd2vcd.
  - VCD file generated from an Identify debugging session
- The DVE integration tool is not available on Windows. See *Use DVE (Discovery Virtual Environment)*, on page 916 for information about DVE.

VCD-Analyst Integration

To use VCD-Analyst Integration, perform the following tasks:

- Verify Design Behavior
- Load the VCD File in HDL Analyst
- Set Nets for Watching
- Debug After Loading VCD and Watching Nets
- Display Simulation Values
- Use DVE (Discovery Virtual Environment)

Copyright © 2013 Synopsys, Inc.

Verify Design Behavior

The VCD-Analyst Integration tool should be used for the following design processes.

During the design phase of the project, use the VCD simulator to simulate the design and DVE to verify the output waveform for the overall design. After simulation results are analyzed, use the VCD-Analyst Integration feature to narrow down and target specific signals that you can correlate to on the schematic in the Technology view.

Load the VCD file for the design into the HDL Analyst view. Values from the VCD file can be annotated on the schematic in the Technology view. You can perform pin point analysis for small sections of the design, when the design does not behave according to the design specification.

You can trace specific signals and cone of logic that is suspected of causing a problem through the schematic to identify the faulty behavior. Having the simulation values annotated on the schematic helps to debug the design easily.

Load the VCD File in HDL Analyst

To load the VCD file in the HDL Analyst:

1. Open the HDL Analyst Technology view.
2. To invoke the Control Panel, click the ( ) icon or select HDLAnalyst->VCD->VCD Panel (`Ctrl+R`) from the Project menu.

Copyright © 2013 Synopsys, Inc.

Use the Move this panel to alternate location ( ) icon to change the VCD Panel view to a location beneath the HDL Analyst view.

3. To load the simulation VCD file, click the Open a VCD File icon ( ) or select HDL-Analyst->VCD->Load VCD File from the Project menu.
   - Then, locate the VCD file using the browse capability to fill in the VCD File field of the Load Simulation VCD File dialog box.

   - Locate the top-level scope, which is the path through the test bench to the root module displayed by the HDL Analyst. Use the Locate Top Level Scope to fill in this field on the Load Simulation VCD File dialog box.

     Limit depth option—when enabled, the hierarchy level for the Root Scope is set to three levels of hierarchy.

     Time option—simulation Time is set to Full Range (0 to 1000 ps). The time range shown is determined from the simulation VCD file to be loaded. This value is calculated when you browse the root scope; in this case the VCD file reached 1000ps.

     Limited Range allows you to choose a range and a desired unit of measurement.

Copyright © 2013 Synopsys, Inc.

– When the Simulation VCD File and Top Level Scope are selected, click the Load button at the bottom of the dialog box to load the VCD file.

– To locate the top-level module scope, click on the Locate Top Level Scope button. This brings up the Select Top Level Module Path dialog box, for which you can identify the VCD scope path through the test bench to the top-level module.

Icons next to the hierarchy tree provide information about how the scopes match the netlist. For example:

	User specified top-level module
	Hierarchical module
	Primitive (Scope is considered primitive by the HDL Analyst)
	Scope is within the test bench
	Scope does not match the hierarchy in the HDL Analyst netlist (Some scopes, such as power, do not exist in the netlist or are considered to be primitive in the HDL Analyst)

Copyright © 2013 Synopsys, Inc.

– To prune the scope tree, click on the Prune Scope Tree button. This brings up the Prune Scope Hierarchy dialog box. Check the root scopes to be included in the hierarchy. If none of the scopes are checked, then the scope for the top-level module is loaded. Click the Clear All button to ensure that none of the root scopes are checked. Once you have pruned the scope tree, the paths for the scopes to be loaded are shown in the complimentary box to the right of the Prune Scope Tree button.

  When loading a pruned VCD file, you cannot watch nets originating in hierarchy levels not loaded. If you try to watch such a net, the following message is generated: `No matching signal`.

– You can load VCD files for sub-modules that contain data only for the sub-module hierarchy of the netlist viewed in the Analyst. To select a sub-module, expand the drop-down menu for the Top Level Module and Paths group and chose the desired module.

Copyright © 2013 Synopsys, Inc.

Validate VCD File with Netlist option—when enabled, the tool reports if there are any nets/signals that mismatch in the design root at the bottom of the Load Simulation VCD File dialog box. Click the Show Mismatches button, to display any mismatches.

If this option is disabled, the tool does not validate the VCD file and loads the VCD netlist without any warnings.

Alternatively, you can use the sch_sim_load Tcl command to load the VCD file. For details, see *sch_sim_load*, on page 1677 in the *Reference* manual.

4. To re-load a VCD file, use the re-load icon ( ) on the Control Panel or the HDL Analyst->VCD->Reload VCD File option from the project menu.

   The previously loaded VCD file is re-opened in a new invocation of the HDL Analyst view.

5. To unload the VCD file, use HDL Analyst->VCD->Unload the VCD File option.

   This option frees up memory used by the simulation data without having to close and re-open the Analyst view.

Set Nets for Watching

You can use any of the following methods to set nets (excluding power connections) for watching:

- You can watch nets on a sheet by selecting one of the following options from HDL Analyst->VCD:
  - Automatically Watch Sheets—while navigating to a sheet, all nets on the sheet are added to the watch list.
  - Watch Sheet—watch all nets on the current sheet.
  - Un-Watch Sheet—remove watching all nets on the current sheet.
- Alternatively, right-click in the Control Panel and select Automatically Watch Sheets, Watch Sheet, or Un-Watch Sheet when watching nets in the HDL Analyst View.
- Drag-and-drop a net, instance, or port from the HDL Analyst View to the Control Panel.
- Drag-and-drop a net, instance, or port from the Hierarchy Browser of the HDL Analyst View to the Control Panel.
- Use Tcl commands for watching nets. For example:
  - Use the find -net command to select nets and then use the following syntax to watch the collection of nets:

    sch_sim_watch -add -coll [**-select**] [*names*]
  - Use the following command syntax to watch the nets:

    sch_sim_watch -add *netNames*
  - Use the following command syntax to remove all nets to be watched from the Control Panel signal list:

    sch_sim_watch -clear

  For details, see *sch_sim_watch*, on page 1679 in the *Reference* manual.

The tool does not allow you to watch mismatched nets and issues a warning in the Tcl window. Also, you can click on the alert icon ( ) in the Control Panel to see these error messages.

Copyright © 2013 Synopsys, Inc.

Debug After Loading VCD and Watching Nets

Use the options on the Control Panel to control the different aspects of the debugging process. The following figure shows the Control Panel in the HDL Analyst View.

The Control Panel changes to a gradient blue color and indicates an active session when the simulation file is loaded.

For example, you can perform the following tasks for debugging:

1. Manipulate the time for observation. To do this, you can
   - Specify the direct time in the required time field and click the Click to set time icon ( ).

   - Move the time slider or use the arrows to move to the next transition.

     Move the time slider left or right to decrease or increase the time. Use the arrows to move time forward or backward to the next transition. If no net is selected, the tool specifies the next transition among all nets

in the list. If some nets are selected, the tool selects the next transition among the selected nets. The edit box next to the slider includes the step size. The + or - buttons moves to the next step size.

– Move the magenta bar to select the current time in the Waveform pane. Click the ( ) icon to toggle the Waveform pane on or off. Left-click in the Waveform pane to set the current time.

  For additional options that you can use in the Waveform pane, right-click and select options from the popup menu.

For all the time observations mentioned above, the corresponding values for the watched nets can be observed in the HDL Analyst View, Control Panel value column, and Waveform pane.

– Find time from selected value.

  Select the required signal name from the list in the Control Panel.

  Select the data format from the drop-down menu and enter the appropriate value to find its corresponding time.

  Use the arrows to find the time with respect to the previous/next occurrences of the value.

– Add time markers relative to the current time. You can:

  - Use the Add Marker icon to add a new marker at the current time.

Copyright © 2013 Synopsys, Inc.

- Move to the specified marker, next, or previous marker.

- Delete markers.

2. Observe values for nets

   The values for nets can be observed by the methods below:

   – Observing in the Control Panel

     The net values are displayed under the Prev | Curr | Next column, respectively. The rows under the Name column display its corresponding signal names.

   – Observing in the HDL Analyst View

     Use the Annotate Values option to control whether to display the net values in the HDL Analyst view. To enable this option, select HDL-Analyst->VCD->Annotate Values.

     The values can be displayed for each net/bus net in order of previous, current, and next values. To choose a display option, right-click and select VCD->Annotate from the popup menu.

Choose a radix (for example, Hex, Binary, Octal, or Decimal).

For a bus net, values of the entire bus as well as individual bits are displayed.

A tool tip is provided that displays information such as simulation values and fanout for the nets.

– Observing in the optional Waveform pane

  The values for signals can be observed in waveform format using the Waveform pane.

– Comparisons using multiple HDL Analyst views

  Open multiple Analyst views.

  Synchronize time changes in each view with respect to reference time to observe differences in simulation results.

– Invoking the DVE Waveform viewer

Copyright © 2013 Synopsys, Inc.

Observe values with the information for watched signals using script files. See *Use DVE (Discovery Virtual Environment)*, on page 916 for more information.

– Crossprobing between the Control Panel and the HDL Analyst View

Crossprobing between HDL Analyst nets and signals listed in the Control Panel is possible.

Display Simulation Values

Use the sch_sim_disp Tcl command to display simulation values for signals. For example:

- Display current nets being watched or a list of specified nets not included in the watch list.
- Display simulations value to the Tcl window or write to an output file.
- Create a file format for importing to a spreadsheet.
- Use the -help option for more information.

For details, see *sch_sim_disp*, on page 1676 in the *Reference* manual.

Use DVE (Discovery Virtual Environment)

DVE is an interactive waveform viewer that you can launch and configure within the VCD-Analyst Integration tool. You can also do the following:

- Configure DVE. See *Configuring DVE*, on page 917 for details.
- Import watched signals from DVE to the VCD-Analyst Integration tool. See *Importing Signals from DVE to the VCD-Analyst Integration Tool*, on page 918 for details.
- Export watched signals from the VCD-Analyst Integration tool to DVE. See *Exporting Nets to DVE from the VCD-Analyst Integration Tool*, on page 919 for details.

Note: You must have a VCS license and run the DVE tool on a Linux platform. Also, set the environment variable VCS_HOME in the VCS_HOME/bin installation directory.

Configuring DVE

To configure your DVE environment, do the following:

1. Select HDL Analyst->VCD->DVE->Configure DVE.

2. From the dialog box, specify the signals that you want to view in DVE:
   - None or Analyst Sim File—select no signals or signals from the HDL Analyst simulation VCD file.
   - Other VCD or VPD—specify the path to another simulation VCD file.
   - Session—you can re-load a saved DVE session Tcl file.
3. Use Transfer Signals Group to transfer specified simulation VCD file signals to a user-specified group. Enable Open a new wave window (when not opening session), to automatically open a waveform view in DVE.

Copyright © 2013 Synopsys, Inc.

4. You can specify any arguments to pass on to DVE, if necessary.

   Note, you must set the environment variable VCS_HOME to bring up DVE.

Importing Signals from DVE to the VCD-Analyst Integration Tool

To import watched signals from DVE to the VCD-Analyst Integration tool, do the following:

1. Select HDL Analyst->VCD->DVE->Import Signals from DVE.

2. From the dialog box, specify the session script file saved from DVE.

3. Then, click the Get button, to extract the signal list from DVE.
4. Select the signals to import to the VCD-Analyst Integration tool: All, None, or check selected signals.
5. If you specify a DVE session file group, you can synchronize the signals in this group to match watched signals in the Control Panel.

Exporting Nets to DVE from the VCD-Analyst Integration Tool

To export watched nets from the VCD-Analyst Integration tool to DVE, do the following:

1. Select HDL Analyst->DVE->Export Signals to DVE.

2. Specify the export path DVE script file to which selected nets are to be exported.

Copyright © 2013 Synopsys, Inc.

3. Specify the DVE session file group that contains the nets to be exported. You can enable Sync Group to synchronize the DVE nets to match the watched signals in the Control Panel.

4. Select the nets to export to the VCD-Analyst Integration tool: All, None, or check selected signals.

5. Then, source the script file in a DVE session.

Launching DVE

Click the DVE button on the Control Panel or select HDL Analyst->VCD->DVE->Launch DVE to bring up the DVE interface shown below.

Identify-Analyst Integration

Integration with the Identify Debugger is available for instrumented signals captured in a VCD file and displayed in the HDL Analyst view. Identify integration is enabled for RTL pre-synthesis files; this VCD file can only be opened in Identify mode.

To use Identify-Analyst Integration, perform the following tasks:

Before beginning the Identify-VCD HDL Analyst integration process, as a prerequisite first:

- Create a Synplify Premier project and use Identify to instrument the design.
- Debug the design on an FPGA and generate the VCD file from the Identify Debugger.

  See the Identify documentation for more information.

Follow the steps in this procedure, which describes how to use the Identify-VCD HDL Analyst tool:

1. Load the Synplify Premier project file created above.
2. Open the HDL Analyst RTL view.
3. To invoke the Identify-VCD HDL Analyst, click the ( ) icon or select HDL-Analyst->VCD->VCD Panel (`Ctrl+R`) from the Project menu.

Copyright © 2013 Synopsys, Inc.

4. To load the Identify VCD file, click the Open a VCD File icon ( ) or select HDL-Analyst->VCD->Load VCD File from the Project menu.

Make sure the Identify Debug option at the bottom of the dialog box is checked; this enables the Identify-VCD HDL Analyst flow.

Validate VCD File with Netlist option—when enabled, the tool reports if there are any mismatches between the nets in the instrumented design and VCD file loaded as shown at the bottom of the Load Identify VCD File dialog box. Click the Show Mismatches button, to display any mismatches. Otherwise, you should see a message that the VCD file was loaded successfully.

If this option is disabled, the tool does not validate the VCD file and loads the VCD netlist without any warnings.

5. To watch nets for the instrumented signals on a particular HDL Analyst sheet, select one of the following options from HDL Analyst->VCD:
   - Automatically Watch Sheets—while navigating to a sheet, all nets on the sheet are added to the watch list.
   - Watch Sheet—watch all nets on the current sheet.
   - Un-Watch Sheet—remove watching all nets on the current sheet.

   You can also right-click and select these options on the waveform viewer as displayed in the following figure. The identify trigger position is indicated by the red cursor.

Copyright © 2013 Synopsys, Inc.

6. To view values for the signals, select all the signals in the waveform viewer and go to HDL-Analyst->VCD->VCD Properties. Check the box for the Annotate option.

7. To annotate values on the waveform viewer to their respective HDL Analyst sheet, check the Annotate box on the Control Panel. Select a particular signal on the Control Panel to observe its corresponding signal and values displayed on the sheet. You can display the previous/current/next values for the signal.

8. You can load, re-load, or unload the VCD file. To do this, go to HDL-Analyst and select one of the following options:
   - To load the Identify VCD file, click the Open a VCD File icon ( ) or select HDL-Analyst->VCD->Load VCD File from the Project menu.
   - To re-load an Identify VCD file, use the re-load icon ( ) on the Control Panel or the HDL Analyst->VCD->Reload VCD File option from the project menu.

     In the case where the Identify debugger generates revised VCD files, changes to a VCD file must be handled after it is loaded. The reload policy implemented provides the following options:

     – Auto—automatically reloads the VCD file

     – Ask—asks if the VCD file should be reloaded

     – Never—never reloads the VCD file

     Auto is the default. You can set the reload policy on the VCD Properties dialog box. To do this, right-click in the Control Panel and select VCD Properties or select HDL Analyst->VCD->VCD Properties option from the project menu.

     When an Identify VCD file is reloaded, the tool preserves information as much as possible, such as the current time and watched signals.

Copyright © 2013 Synopsys, Inc.

– To unload the Identify VCD file, use HDL Analyst->VCD->Unload the VCD File option.

  This option frees up memory used by the Identify debug data without having to close and re-open the Analyst view.

9. You can change the format of signals by selecting a signal, then right-click and select the Format option on the Control Panel or use HDL-Analyst->VCD->VCD Properties.

VCS Test Bench Extraction

Test bench extraction automatically extracts RTL test benches for design submodules to run the VCS simulator that is derived from the top-level RTL test bench. This hierarchical approach improves the time for initial board bring-up and allows you to perform internal signoff of a submodule or IP used in the design. The test bench extraction software automatically creates test bench files by launching the VCS `vcat` utility. The full RTL simulation VCD file is selectively read to provide information about design scopes needed to run `vcat` and to determine simulation times and parameters. Test bench extraction also identifies whether simulation data exists for the selected module (xDUT) in the RTL Analyst view. Appropriate naming conventions are applied so that multiple test bench files can co-exist in the same directory.

When debugging, you can often narrow down the faulty behavior to a particular instance or module. For more granular verification at this point, it would be easier to work with a test bench for just that module or instance. Test bench extraction allows you to selectively read the `vcd` file to obtain information for the test bench and automatically generate a test bench for a specific module from the original top-level test bench.

Test bench extraction must be run with the Synplify Premier tools on Linux, since `vcat` and `vcs` are only available on this platform.

Using Test Bench Extraction

This procedure describes the steps required for test bench generation and simulation.

1. Create a Synplify Premier project for the full design.
2. Run simulation to create the full RTL simulation VCD file. For information about VCS simulation, see *Simulating with the VCS Tool*, on page 899.
3. Open the RTL Analyst view for the full design implementation.
4. Locate the module to be extracted (xDUT) from the schematic or hierarchy browser. Select it.

   You must select only one hierarchical instance in the schematic view.
5. Right-click and select VCD->Test Bench Extraction to display the Test Bench Extraction dialog box. Use this to:

Copyright © 2013 Synopsys, Inc.

– Generate the test bench files. For more information, see *Defining the Test Bench Architecture*, on page 929.

– Launch the VCS simulator for the xDUT module.

For a description of the fields used in this dialog box, see the *Test Bench Extraction Command*, on page 410.

6. Debug the module for errors.

   – See the log file produced by the simulation.

– View the simulation VCD file produced by the xDUT simulation in DVE or other waveform viewer.

The main simulation Verilog file generates output indicating the success or failure of the simulation. The file contains:

– Ports that are monitored

– PASS/FAIL status for each port

– Detailed report where mismatches occur (port, time, expected value, actual value)

Defining the Test Bench Architecture

When you click the Gen Test Bench button, the software creates a test bench file that contains the main module, as well as, the stimulus and golden modules generated by `vcat`.

Main Module

The main module includes the following:

- The test bench instantiates the stimulus and golden modules generated by `vcat`. See *Golden Module*, on page 929 and *Stimulus Module*, on page 930.
- A clock signal delay, delayed by the amount specified on the Test Bench Extraction dialog box is used for signal comparison and reporting.
- At the positive edge of the delayed clock, the outputs of the stimulus and golden modules are captured for registers.
- At the negative edge of the clock and for the reset line with the value specified on the Test Bench Extraction dialog box, the registers holding the stimulus and golden outputs are compared. Any differences are reported, which indicate the expected (Golden) and actual (xDUT) values and the time of comparison.
- A final summary of the simulation is displayed. This includes a table of PASS/FAIL status for ports.

Golden Module

The golden module includes the following:

- The golden module has outputs that correspond to the xDUT outputs.

Copyright © 2013 Synopsys, Inc.

- Using blocking assignments that are delayed by the amount specified on the Test Bench Extraction dialog box, the outputs are forced to the expected values based on the RTL full simulation.

Stimulus Module

The stimulus module includes the following:

- The stimulus module instantiates the xDUT module.
- Inputs and outputs connect to the xDUT.
- Using block assignments that are delayed by the amount specified on the Test Bench Extraction dialog box, the inputs for the xDUT are forced to the values that occurred in the RTL full simulation.

Limitations to Consider

Consider the limitations for the following:

- Port Names
- Race Conditions
- Multiple Clocks
- Test Bench File Order

Port Names

The extracted test bench uses the RTL netlist to determine input and output port names of the submodule (xDUT). For simulation to work correctly, these must match the port names of the submodule in the synthesized netlist.

Note: If you get VCS port mismatch errors, check that you are using the correct RTL simulation.

Race Conditions

You might need to correct race conditions by adjusting signal arrival times. Beginning at time 0, `vcat` applies the signals with blocking assignments in the stimulus/golden test bench files. Specify an offset on the Test Bench Extraction dialog box, to offset this time. You can also manually edit the files for more complex modifications.

Multiple Clocks

Test bench extraction does not support multiple clocks. In this case, you are required to manually edit the main test bench module.

Test Bench File Order

The main test bench file and the stimulus file generated by `vcat` contain a timescale directive. However, the module generation (Golden) file does not. Therefore, the best order for passing the files to `vcs` is

- Main test bench file (`.vt`)
- Stimulus file (*`instPath`*`_vcat_stimulus.cfg`)
- Golden file (*`instPath`*`_vcat_golden.cfg`)

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

CHAPTER 18

Working with IP Input

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

- Generating IP with SYNCore, on page 934
- The Synopsys FPGA IP Encryption Flow, on page 970
- Working with Encrypted IP, on page 975
- Using DesignWare IP, on page 988
- Working with Synenc-encrypted IP, on page 992
- Using Hyper Source, on page 993
- Working with Altera IP, on page 998
- Working with SOPC Builder Components, on page 1029
- Importing Projects from Quartus, on page 1033
- Working with Lattice IP, on page 1042
- Importing Vivado IP, on page 1043
- Working with Xilinx IP Cores, on page 1046
- Converting Xilinx Projects with ise2syn, on page 1051

Copyright © 2013 Synopsys, Inc.

Generating IP with SYNCore

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

- Specifying FIFOs with SYNCore, on page 934
- Specifying RAMs with SYNCore, on page 939
- Specifying Byte-Enable RAMs with SYNCore, on page 946
- Specifying ROMs with SYNCore, on page 952
- Specifying Adder/Subtractors with SYNCore, on page 957
- Specifying Counters with SYNCore, on page 964

Specifying FIFOs with SYNCore

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

Note: The SYNCore FIFO model uses Verilog 2001. When adding a FIFO model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a set_option -vlog_std v2001 statement in your project file to prevent a syntax error.

---

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

– In the window that opens, select sfifo_model and click Ok. This opens the first screen of the wizard.

2. Specify the parameters you need in the five pages of the wizard. For details, refer to *Specifying SYNCore FIFO Parameters*, on page 937.

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

You can now close the SYNCore wizard.

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

```
module top (

input Clk,
input [15:0] DataIn,
input WrEn,
input RdEn,

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

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

endmodule
```

template

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

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

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

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

Specifying SYNCore FIFO Parameters

The following elaborates on the parameter settings for SYNCore FIFOs. The status, handshaking, and programmable flags are optional. For descriptions of the parameters, see *SYNCore FIFO Wizard*, on page 247 in the *Reference Manual*. For timing diagrams, see *Synplicity Archive Utility*, on page 1025in the *Reference Manual*.

1. Start the SYNCore wizard, as described in *Specifying FIFOs with SYNCore*, on page 934.

2. Do the following on page 1 of the FIFO wizard:

   – In Component Name, specify a name for the FIFO. Do not use spaces.

Copyright © 2013 Synopsys, Inc.

- In Directory, specify a directory where you want the output files to be written. Do not use spaces.
- In Filename, specify a name for the Verilog output file with the FIFO specifications. Do not use spaces.
- Click Next. The wizard opens another page where you can set parameters.

3. For a FIFO with no status, handshaking, or programmable flags, use the default settings. You can generate the FIFO, as described in *Specifying FIFOs with SYNCore*, on page 934.

4. To set an almost full status flag, do the following on page 2 of the FIFO wizard:
   - Enable Almost Full.
   - Set associated handshaking flags for the signal as desired, with the Overflow Flag and Write Acknowledge options.
   - Click Next when you are done.

5. To set an almost empty status flag, do the following on page 3:
   - Enable Almost Empty.
   - Set associated handshaking flags for the signal as desired, with the Underflow Flag and Read Acknowledge options.
   - Click Next when you are done.

6. To set a programmable full flag, do the following:
   - Make sure you have enabled Full on page 2 of the wizard and set any handshaking flags you require.
   - Go to page 4 and enable Programmable Full.
   - Select one of the four mutually exclusive configurations for Programmable Full on page 4. See *Programmable Full*, on page 1037 in the *Reference Manual* for details.
   - Click Next when you are done.

7. To set a programmable empty flag, do the following:
   - Make sure you have enabled Empty on page 3 of the wizard and set any handshaking flags you require.
   - Go to page 5 and enable Programmable Empty.

Copyright © 2013 Synopsys, Inc.

– Select one of the four mutually exclusive configurations for Programmable Empty on page 5. See *Programmable Empty*, on page 1040 in the *Reference Manual* for details.

You can now generate the FIFO and add it to the design, as described in *Specifying FIFOs with SYNCore*, on page 934.

Specifying RAMs with SYNCore

The SYNCore IP wizard helps you generate Verilog code for your RAM implementation requirements. The following procedure shows you how to generate Verilog code for a RAM using the SYNCore IP wizard.

Note: The SYNCore RAM model uses Verilog 2001. When adding a RAM model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a set_option -vlog_std v2001 statement in your project file to prevent a syntax error.

---

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

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

Copyright © 2013 Synopsys, Inc.

2. Specify the parameters you need in the wizard.
   - For details about the parameters for a single-port RAM, see *Specifying Parameters for Single-Port RAM*, on page 942.
   - For details about the parameters for a dual-port RAM, see *Specifying Parameters for Dual-Port RAM*, on page 943. Note that dual-port implementations are only supported for some technologies.

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

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

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

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

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

You can now close the SYNCore Memory Compiler.

4. Edit the RAM files if necessary.
   - The default RAM has a no_rw_check attribute enabled. If you do not want this, edit syncore_ram.v and comment out the `define SYN_MULTI_PORT_RAM statement, or use `undef SYN_MULTI_PORT_RAM.
   - If you want to use the synchronous RAMs available in the target technology, make sure to register either the read address or the outputs.
5. Add the RAM you generated to your design.
   - Use the Add File command to add the Verilog design file that was generated and the syncore_ram.v file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
   - Use a text editor to open the instantiation_file.vin template file, which is located in the same directory. Copy the lines that define the memory, and paste them into your top-level module. The following figure shows a template file (in red text) inserted into a top-level module.

```
module top (

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

output [15:0] DataOutA

);

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

endmodule
```

template

Copyright © 2013 Synopsys, Inc.

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

```
module top (

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

output [15:0] DataOutA

);


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

endmodule
```

Specifying Parameters for Single-Port RAM

To create a single-port RAM with the SYNCore Memory Compiler, you need to specify a single read/write address (single port) and a single clock. You only need to configure Port A. The following procedure lists what you need to specify. For descriptions of each parameter, refer to *SYNCore RAM Wizard*, on page 257 in the *Reference Manual*.

1. Start the SYNCore RAM wizard, as described in *Specifying RAMs with SYNCore*, on page 939.

2. Do the following on page 1 of the RAM wizard:
   – In Component Name, specify a name for the memory. Do not use spaces.
   – In Directory, specify a directory where you want the output files to be written. Do not use spaces.

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

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

3. Do the following on page 2 of the RAM wizard:
   - Set Use Write Enable to the setting you want.
   - Set Register Read Address to the setting you want.
   - Set Synchronous Reset to the setting you want. Register Outputs is always enabled
   - Specify the read access you require for the RAM.

   You can now generate the RAM by clicking Generate, as described in *Specifying RAMs with SYNCore*, on page 939. You do not need to specify any parameters on page 3, as this is a single-port RAM and you do not need to specify Port B. All output files are in the directory you specified on the first page of the wizard.

   For details about setting dual-port RAM parameters, see *Specifying Parameters for Dual-Port RAM*, on page 943. For read/write timing diagrams, see *Read/Write Timing Sequences*, on page 1050 of the *Reference Manual*.

Specifying Parameters for Dual-Port RAM

The following procedure shows you how to set parameters for dual-port memory in the SYNCore wizard. Dual-port RAMs are only supported for some technologies. For information about generating single-port RAMs, see *Specifying Parameters for Single-Port RAM*, on page 942. It shows you how to generate these common RAM configurations:

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

Copyright © 2013 Synopsys, Inc.

For the corresponding read/write timing diagrams, see *Read/Write Timing Sequences*, on page 1050 of the *Reference Manual*.

1. Start the SYNCore RAM wizard, as described in *Generating IP with SYNCore*, on page 934.

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

For a single clock...	Enable Single Clock.
For separate clocks for each of the ports...	Enable Separate Clocks For Each Port.

   - Click Next. The wizard opens another page where you can set parameters for Port A.

3. Do the following on page 2 of the RAM wizard to specify settings for Port A:
   - Set parameters according to the kind of memory you want to generate:

One read & one write	Enable Read Only Access.
Two reads & one write	Enable Read and Write Access. Specify a setting for Use Write Enable.
Two reads & two writes	Enable Read and Write Access. Specify a setting for Use Write Enable. Specify a read access option for Port A.

   - Specify a setting for Register Read Address.

- Set Synchronous Reset to the setting you want. Register Outputs is always enabled.
- Click Next. The wizard opens another page where you can set parameters for Port B. The page and the parameters are identical to the previous page, except that the settings are for Port B instead of Port A.

4. Specify the settings for Port B on page 3 of the wizard according to the kind of memory you want to generate:

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

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

You can now generate the RAM by clicking Generate, as described in *Generating IP with SYNCore*, on page 934, and add it to your design.

Copyright © 2013 Synopsys, Inc.

Specifying Byte-Enable RAMs with SYNCore

The SYNCore IP wizard helps you generate SystemVerilog code for your byte-enable RAM implementation requirements. The following procedure shows you how to generate SystemVerilog code for a byte-enable RAM using the SYNCore IP wizard.

Note: The SYNCore byte-enable RAM model uses SystemVerilog. When adding a byte-enable RAM to your design, be sure to enable the System Verilog check box on the Verilog tab of the Implementation Options dialog box or include a set_option -vlog_std sysv statement in your project file to prevent a syntax error.

---

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

   - In the window that opens, select byte_en_ram_model and click Ok to open the first page (page1) of the wizard.

2. Specify the parameters you need in the wizard. For details about the parameters, see *Specifying Byte-Enable RAM Parameters*, on page 950. The BYTE ENABLE RAM symbol on the left reflects any parameters you set.

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

   SYNCore also generates a test bench for the byte-enable RAM component. The test bench covers a limited set of vectors. You can now close the SYNCore byte-enable RAM compiler.

4. Edit the generated files for the byte-enable RAM component if necessary.

5. Add the byte-enable RAM that you generated to your design.
   – On the Verilog tab of the Implementation Options dialog box, make sure that SystemVerilog is enabled.

Copyright © 2013 Synopsys, Inc.

- Use the Add File command to add the Verilog design file that was generated (the filename entered on page 1 of the wizard) and the syncore_*.v file to your project. These files are in the directory for output files that you specified on page 1 of the wizard.
- Use a text editor to open the instantiation_file.vin template file. This file is located in the same output files directory. Copy the lines that define the byte-enable RAM and paste them into your top-level module.
- Edit the template port connections so that they agree with the port definitions in the top-level module; also change the instantiation name to agree with the component name entered on page 1. The following figure shows a template file inserted into a top-level module with the updated component name and port connections in red.

```
module top
   (input ClockA,
    input [3:0] AddA
    input [31:0] DataIn
    input WrEnA,
    input Reset
    output [31:0] DataOut
   )

INST_TAG

SP_RAM #
   (.ADD_WIDTH(4),
    .WE_WIDTH(2),
    .RADDR_LTNCY_A(1), // 0 - No Latency , 1 - 1 Cycle Latency
    .RDATA_LTNCY_A(1), // 0 - No Latency , 1 - 1 Cycle Latency
    .RST_TYPE_A(1), // 0 - No Reset , 1 synchronous
    .RST_RDATA_A({32{1'b1}}),
    .DATA_WIDTH(32)
   )

4x32spram
   (// Output Ports
    .RdDataA(DataIn),
    // Input  Ports
    .WrDataA(DataOut),
    .WenA(WeEnA),
    .AddrA(AddA),
    .ResetA(Reset),
    .ClkA(ClockA)
   );
```

Port List

Port A interface signals are applicable for both single-port and dual-port configurations; Port B signals are applicable for dual-port configuration only.

Name	Type	Description
ClkA	Input	Clock input for Port A
WenA	Input	Write enable for Port A; present when Port A is in write mode
AddrA	Input	Memory access address for Port A
ResetA	Input	Reset for memory and all registers in core; present with registered read data when Reset is enabled; active low (cannot be changed)
WrDataA	Input	Write data to memory for Port A; present when Port A is in write mode
RdDataA	Output	Read data output for Port A; present when Port A is in read or read/write mode
ClkB	Input	Clock input for Port B; present in dual-port mode
WenB	Input	Write enable for Port B; present in dual-port mode when Port B is in write mode
AddrB	Input	Memory access address for Port B; present in dual-port mode
ResetB	Input	Reset for memory and all registers in core; present in dual-port mode when read data is registered and Reset is enabled; active low (cannot be changed)
WrDataB	Input	Write data to memory for Port B; present in dual-port mode when Port B is in write mode
RdDataB	Output	Read data output for Port B; present in dual-port mode when Port B is in read or read/write mode

Copyright © 2013 Synopsys, Inc.

Specifying Byte-Enable RAM Parameters

When creating a single-port, byte-enable RAM with the SYNCore IP wizard, you must specify a single read address and a single clock; you only need to configure the Port A parameters on page 2 of the wizard.

When creating a dual-port, byte-enable RAM, you must additionally configure the Port B parameters on page 3 of the wizard.

The following procedure lists the parameters you need to specify. For descriptions of each parameter, refer to *Parameter List*, on page 1057 in the reference manual.

1. Start the SYNCore byte-enable RAM wizard as described in *Specifying Byte-Enable RAMs with SYNCore*, on page 946.

2. Do the following on page 1 of the byte-enable RAM wizard:
   - Specify a name for the memory in the Component Name field; do not use spaces.
   - Specify a directory name in the Directory field where you want the output files to be written; do not use spaces.
   - Specify a name in the File Name field for the SystemVerilog file to be generated with the byte-enable RAM specifications; do not use spaces.
   - Enter a value for the address width of the byte-enable RAM; the maximum depth of memory is limited to 2^256.
   - Enter a value for the data width for the byte-enable RAM; data width values range from 2 to 256.
   - Enter a value for the write enable width; write-enable width values range from 1 to 4.
   - Select Single Port to generate a single-port, byte-enable RAM or select Dual Port to generate a dual-port, byte-enable RAM.
   - Click Next to open page 2 of the wizard.

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

Copyright © 2013 Synopsys, Inc.

3. Do the following on page 2 (configuring Port A) of the wizard:
   - Select the Port A configuration. Only Read and Write Access mode is valid for single-port configurations; this mode is selected by default.
   - Set Pipelining Address Bus and Output Data according to your application. By default, read data is registered; you can register both the address and data registers.
   - Set the Configure Reset Options. Enabling the checkbox enables the synchronous reset for read data. This option is enabled only when the read data is registered. Reset is active low and cannot be changed.
   - Configure output reset data value options under Specify output data on reset; reset data can be set to default value of all '1' s or to a user-defined decimal value. Reset data value options are disabled when the reset is not enabled for Port A.
   - Set Write Enable for Port A value; default for the write-enable level is active high.
4. If you are generating a dual-port, byte-enable RAM, set the Port B parameters on page 3 (note that the Port B parameters are only enabled when Dual Port is selected on page 1).

   The Port B parameters are identical to the Port A parameters on page 2. When using the dual-port configuration, when one port is configured for read access, the other port can only be configured for read/write access or write access.
5. Generate the byte-enable RAM by clicking Generate. Add the file to your project and edit the template file as described in *Specifying Byte-Enable RAMs with SYNCore*, on page 946. For read/write timing diagrams, see *Read/ Write Timing Sequences*, on page 1054 of the reference manual.

Copyright © 2013 Synopsys, Inc.

Specifying ROMs with SYNCore

The SYNCore IP wizard helps you generate Verilog code for your ROM implementation requirements. The following procedure shows you how to generate Verilog code for a ROM using the SYNCore IP wizard.

Note: The SYNCore ROM model uses Verilog 2001. When adding a ROM model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a set_option -vlog_std v2001 statement in your project file to prevent a syntax error.

---

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

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

2. Specify the parameters you need in the wizard. For details about the parameters, see *Specifying ROM Parameters*, on page 956. The ROM symbol on the left reflects any parameters you set.

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

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

   You can now close the SYNCore ROM Compiler.

4. Edit the ROM files if necessary. If you want to use the synchronous ROMs available in the target technology, make sure to register either the read address or the outputs.

5. Add the ROM you generated to your design.
   – Use the Add File command to add the Verilog design file that was generated and the syncore_rom.v file to your project. These files are in

Copyright © 2013 Synopsys, Inc.

the directory for output files that you specified on page 1 of the wizard.

– Use a text editor to open the instantiation_file.vin template file. This file is located in the same output files directory. Copy the lines that define the ROM, and paste them into your top-level module. The following figure shows a template file (in red text) inserted into a top-level module.

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

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

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

template

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

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

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

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

Port List

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

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

Copyright © 2013 Synopsys, Inc.

Specifying ROM Parameters

If you are creating a single-port ROM with the SYNCore IP wizard, you need to specify a single read address and a single clock, and you only need to configure the Port A parameters on page 2. If you are creating a dual-port ROM, you must additionally configure the Port B parameters on page 3. The following procedure lists what you need to specify. For descriptions of each parameter, refer to *SYNCore RAM Wizard*, on page 257 in the *Reference Manual*.

1. Start the SYNCore ROM wizard, as described in *Specifying ROMs with SYNCore*, on page 952.

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

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

3. Do the following on page 2 (Configuring Port A) of the RAM wizard:
   - For synchronous ROMs, select Register address bus AddrA and/or Register output data bus DataA to register the read address and/or the outputs. Selecting either checkbox enables the Enable for Port A checkbox which is used to select the Enable level.
   - Set the Configure Reset Options. Enabling the checkbox enables the type of reset (asynchronous or synchronous) and allows an output data pattern (all 1's or a specified pattern) to be defined on page 4.

4. If you are generating a dual-port ROM, set the port B parameters on page 3 (the page 3 parameters are only enabled when Dual Port Rom is selected on page 1).

Copyright © 2013 Synopsys, Inc.

5. On page 4, specify the location of the ROM initialization file and the data format (Hexadecimal or Binary). ROM initialization is supported using memory-coefficient files. The data format is either binary or hexadecimal with each data entry on a new line in the memory-coefficient file (specified by parameter INIT_FILE). Supported file types are `txt`, `mem`, `dat`, and `init` (recommended).

6. Generate the ROM by clicking Generate, as described in *Specifying ROMs with SYNCore*, on page 952 and add it to your design. All output files are in the directory you specified on page 1 of the wizard.

   For read/write timing diagrams, see *Read/Write Timing Sequences*, on page 1050 of the *Reference Manual*.

Specifying Adder/Subtractors with SYNCore

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

Note: The SYNCore adder/subtractor models use Verilog 2001. When adding an adder/subtractor model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a set_option -vlog_std v2001 statement in your project file to prevent a syntax error.

---

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

Copyright © 2013 Synopsys, Inc.

– In the window that opens, select addnsub_model and click Ok to open page1 of the wizard.

2. Specify the parameters you need in the wizard. For details about the parameters, see *Specifying Adder/Subtractor Parameters*, on page 962. The ADDnSUB symbol on the left reflects any parameters you set.

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

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

endmodule
```

template

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

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

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

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

Port List

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

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

Copyright © 2013 Synopsys, Inc.

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

Specifying Adder/Subtractor Parameters

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

- Adder
- Subtractor
- Dynamic Adder/Subtractor

If you are creating a constant input adder, subtractor, or a dynamic adder/subtractor with the SYNCore IP wizard, you must select Constant Value Input and specify a value for port B in the Constant Value/Port B Width field on page 2 of the parameters. The following procedure lists the parameters you need to define when generating an adder/subtractor. For descriptions of each parameter, see *SYNCore Adder/Subtractor Wizard*, on page 268 in the *Reference Manual*.

1. Start the SYNCore adder/subtractor wizard as described in *Specifying Adder/Subtractors with SYNCore*, on page 957.
2. Enter the following on page 1 of the wizard:
   - In the Component Name field, specify a name for your adder/subtractor. Do not use spaces.
   - In the Directory field, specify a directory where you want the output files to be written. Do not use spaces.
   - In the Filename field, specify a name for the Verilog file that will be generated with the adder/subtractor definitions. Do not use spaces.
   - Select the appropriate configuration in Configure the Mode of Operation.
3. Click Next. The wizard opens page 2 where you set parameters for port A and port B.
4. Configure Port A and B.
   - In the Configure Port A section, enter a value in the Port A Width field.
   - If you are defining a synchronous adder/subtractor, check Register Input A and then check Clock Enable for Register A and/or Reset for Register A.
   - To configure port B as a constant port, go to the Configure Port B section and check Constant Value Input. Enter the constant value in the Constant Value/Port B Width field.
   - To configure port B as a dynamic port, go to the Configure Port B section and check Enable Port B and enter the port width in the Constant Value/Port B Width field.
   - To define a synchronous adder/subtractor, check Register Input B and then check Clock Enable for Register B and/or Reset for Register B.
5. In the Configure Output Port section:
   - Enter a value in the Output port Width field.
   - If you are registering the output port, check Register output Port.
   - If you are defining a synchronous adder/subtractor check Clock Enable for Register PortOut and/or Reset for Register PortOut.
6. In the Configure Reset type for all Reset Signal section, click Synchronous Reset or Asynchronous Reset as appropriate.

Copyright © 2013 Synopsys, Inc.

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

7. Generate the adder/subtractor by clicking the Generate button as described in *Specifying Adder/Subtractors with SYNCore, on page 957* and add it to your design. All output files are in the directory you specified on page 1 of the wizard.

Specifying Counters with SYNCore

The SYNCore IP wizard helps you generate Verilog code for your counter implementation requirements. The following procedure shows you how to generate Verilog code for a counter using the SYNCore IP wizard.

Note: The SYNCore counter model use Verilog 2001. When adding a counter model to a Verilog-95 design, be sure to enable the Verilog 2001 check box on the Verilog tab of the Implementation Options dialog box or include a set_option -vlog_std v2001 statement in your project file to prevent a syntax error.

---

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

   - In the window that opens, select counter_model and click Ok to open page1 of the wizard.

2. Specify the parameters you need in the wizard. For details about the parameters, see *Specifying Counter Parameters, on page 968*. The COUNTER symbol on the left reflects any parameters you set.

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

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

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

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

SynCoreCounter_ins1 (
   .PortCount(PortCount),
   .PortClk(PortClk),
   .PortRST(PortRST),
   .PortUp_nDown(PortUp_nDown),
   .PortLoad(PortLoad),
   .PortLoadValue(PortLoadValue),
   .PortCE(PortCE) );

endmodule
```

template

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

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

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

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

SynCoreCounter_ins1 (
   .PortCount(PortCount),
   .PortClk(Clock),
   .PortRST(Reset),
   .PortUp_nDown(Up_Down),
   .PortLoad(Load),
   .PortLoadValue(LoadValue),
   .PortCE(Enable) );

endmodule
```

Port List

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

Copyright © 2013 Synopsys, Inc.

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

Specifying Counter Parameters

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

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

The counter core can have a constant or variable input load or no load value. If you are creating a constant-load counter, you will need to select Enable Load and Load Constant Value on page 2 of the wizard. If you are creating a variable-load counter, you will need to select Enable Load and Use Variable Port Load on page 2. The following procedure lists the parameters you need to define when generating a counter. For descriptions of each parameter, see *SYNCore Counter Wizard*, on page 272 of the *Reference Manual*.

1. Start the SYNCore counter wizard, as described in *Specifying Counters with SYNCore*, on page 964.
2. Enter the following on page 1 of the wizard:

- In the Component Name field, specify a name for your counter. Do not use spaces.
- In the Directory field, specify a directory where you want the output files to be written. Do not use spaces.
- In the Filename field, specify a name for the Verilog file that will be generated with the counter definitions. Do not use spaces.
- Enter the width and depth of the counter in the Configure the Counter Parameters section.
- Select the appropriate configuration in the Configure the Mode of Counter section.

3. Click Next. The wizard opens page 2 where you set parameters for PortLoad and PortLoadValue.
   - Select Enable Load option and the required load option in Configure Load Value section.
   - Select the required reset type in the Configure Reset type section.

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

4. Generate the counter core by clicking Generate button. All output files are written to the directory you specified on page1 of the wizard.

Copyright © 2013 Synopsys, Inc.

The Synopsys FPGA IP Encryption Flow

The Synopsys FPGA IP encryption flow is a design flow that encourages interoperability while protecting IP implementations using encryption/decryption technologies. This flow offers the following advantages: interoperability, protection of IP, reuse of IP, and a standard flow for IP encryption. Currently, Synopsys FPGA synthesis products support the following encryption technologies:

- P1735 with key-block embedded rights information (Version 1)
- OpenIP

Overview of the Synopsys FPGA IP Flow

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

For further details of the hand-offs between vendors and how encryption and decryption are handled, see *Encryption and Decryption*, on page 971.

Encryption and Decryption

There are two major classes of encryption/decryption algorithms: symmetric, and asymmetric (see *Encryption and Decryption Methodologies*, on page 1081 in the *Reference Manual* for details). Each has its own advantages and disadvantages. The approach for the Synopsys FPGA IP flow is a hybrid scheme that uses both asymmetric and symmetric encryption to leverage the strengths of each scheme. The methodology described here can also be used for other design handoffs. For example, for a handoff from synthesis to place-and-route, the synthesis tool would be in the upstream position occupied by the IP vendor in this flow, and the FPGA vendor would be in the downstream position occupied by the synthesis tool.

Copyright © 2013 Synopsys, Inc.

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

The following describes each of the phases shown in the figure. Note that Synopsys provides the following scripts to simplify and automate the process of encrypting data for the IP vendor.

- P1735
- OpenIP

Data Encryption - Step 1

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

Symmetric Encryption/Decryption with One Key

Data Key Encryption - Step 2

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

Asymmetric Encryption/Decryption with Public and Private Keys

Asymmetric encryption offers the following advantages:

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

Copyright © 2013 Synopsys, Inc.

Bundling of Encrypted Data Block and Data Key - Step 3

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

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

Data Key Decryption - Step 4

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

Data Decryption - Step 5

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

After synthesis, the IP can be re-encrypted if the vendor has adopted one of the Synopsys methodologies. See *Output Methods for encryptIP*, on page 1092 in the *Reference Manual* for a description of the choices available.

Re-Encryption in the Synopsys FPGA IP Flow

Re-encryption of the synthesized IP for FPGA vendors downstream requires that the FPGA vendor supply Synopsys with a public key. When the input file includes a downstream key block, the re-encrypted data is accessible to the downstream tool. If such an agreement is not in place, the IP is treated as a black box. Accordingly, you can have an IP flow that outputs black boxes in the netlists, plaintext netlists, or encrypted netlists.

Copyright © 2013 Synopsys, Inc.

Working with Encrypted IP

The Synopsys FPGA IP encryption schemes available to Synplify Pro and Synplify Premier include:

- P1735
- OpenIP

With either of these approaches, the IP vendor can encrypt and control distribution of the IP from their own website. The synthesis user will have access from the synthesis tool to the IP that the vendor makes available for download and evaluation within a synthesis design.

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

- Encrypting Your IP, on page 975
- Preparing the IP Package, on page 981

Encrypting Your IP

IP vendors can use either of the supported Synopsys FPGA IP schemes to provide IP for synthesis users to evaluate and use. Both schemes uses a two-stage encryption process:

- First, encrypt your IP files using a symmetric encryption algorithm and your own session or data key to create an encrypted data block.
- Next, encrypt the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. All of the Synopsys encryption methodologies support RSA encryption.

Synopsys provides scripts to simplify this process. See the following procedures for details on script usage.

Preparing and Encrypting Your IP

To prepare and encrypt your IP, do the following:

1. Gather your RTL files.

Copyright © 2013 Synopsys, Inc.

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

2. Determine your file setup for each IP.
   - Create a single set of files for the IP (for use with all supported FPGAs), if your IP has no vendor-specific or vendor-optimized content and if the output method is supported by all intended consumers (blackbox or plaintext).
   - Create multiple versions of your protected IP if you have specific FPGA vendors or specific FPGA vendor families; if you are using FPGA device-family specific RTL like architecture-specific instantiations; or if you optimized your RTL or constraints for use with a specific FPGA vendor device family or FPGA vendor.
3. Encrypt the files with the appropriate encryption script as described in one of the following subsections:
   - *Encrypting IP with the encryptP1735.pl Script*, on page 976
   - *Encrypting IP with the encryptIP Script*, on page 978.
4. Package your IP, as described in *Preparing the IP Package*, on page 981.
5. Verify that your IP works with the synthesis tools by going through the procedure that the user would use.
   - Start the synthesis tool and load the IP with the Import IP->Import IP Package command. You can load your IP into an existing Synplify project.
   - For system-level IP, run it through System Designer™ and ensure bus-model compatibility between your IP and any other IP to which it interfaces. See the System Designer documentation for details on using this tool.
   - Run synthesis.

Encrypting IP with the encryptP1735.pl Script

The encryptP1735.pl script supports the P1735 proposed standard with limited interoperability and rights information embedded in the key block. The encryptP1735.pl script accepts inputs from three sources: command line arguments, the RTL input file containing one or more encryption envelopes, and a file containing the public keys.

A keys.txt file, which contains the public key for consumption by Synopsys FPGA tools, is included with the script. Add other public keys to this file when the IP is to be consumed by additional EDA tools.

The following procedure shows you how to encrypt your data with the encryptP1735.pl script. This script automates the two-stage encryption process described in the Synopsys FPGA IP scheme (*The Synopsys FPGA IP Encryption Flow*, on page 970). The encryptP1735.pl script:

- First encrypts your IP files using a symmetric encryption algorithm and your own session or data key to create an encrypted data block.
- Next encrypts the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. Synplify currently supports RSA encryption.

The encryptP1735.pl script is located in the *installDir*/lib directory and requires the installation of Perl on your machine. You cannot run the script if you do not have Perl installed. The following examples show typical script applications. For more information on the script and the command line arguments, see *Syntax for Running encryptP1735*, on page 1082 in the Reference Manual.

Example 1

To encrypt a file using a random key:

```
perl encryptP1735.pl -input plain_ip.v -output protected_ip.v
   -output_method plaintext
```

Example 2

To encrypt a file using a user-specified key:

```
perl encryptP1735.pl -input plain_ip.v -output protected_ip.v
   -output_method plaintext -key mySpecifiedKey
```

In the above examples, the -input (or -i) argument specifies the name of the file to be encrypted, and the -output (or -o) argument specifies the name of the resultant encrypted file. The -output_method (or -om) argument accepts the following values:

- plaintext indicates that the synthesis output netlist for the IP will be in plaintext format
- encrypted indicates that the synthesis output netlist for the IP will be encrypted using the same session key as input

Copyright © 2013 Synopsys, Inc.

- blackbox indicates that the synthesis output netlist for the IP will not be written (that is, the IP will be black-boxed).

The -key (or -k) argument is optional and specifies the session key in text format.

Encrypting IP with the encryptIP Script

The following procedure shows you how to encrypt your data with the encryptIP (OpenIP) script. The encryptIP script automates the two-stage encryption process proposed in the Synopsys FPGA IP scheme (*The Synopsys FPGA IP Encryption Flow*, on page 970).

- First, it encrypts your IP files using a symmetric encryption algorithm and your own session or data key. This creates an encrypted data block.
- Next, it encrypts the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. Synplify currently supports RSA encryption.

1. Install the encryptIP Perl script.
   - You can download the encryptIP Perl script from SolvNet. See the article published at:

     https://solvnet.synopsys.com/retrieve/032343.html
   - Install Perl on your machine. You cannot run the script if you do not have Perl installed.
2. Make sure that the encryptIP script specifies the decryption key and the matching key length:
   - Specify the symmetric data decryption key with the -k option. Optionally, you can also specify a symmetric encryption key in hexadecimal format with the -kx option.
   - Make sure you specify the right key length for the encryption algorithm with the -c option. For example, TEST1234 becomes a 64-bit key, so you specify the des-cbc algorithm.

   See *Syntax for Running encryptIP*, on page 1091 in the *Reference Manual* for full details of the encryptip syntax.
3. Make sure you specify the appropriate output method (-om) when you run the script.

This is important because the output method (-om) determines what is encrypted to the user. When the example above is synthesized, the user can view the output netlist because the output method specified is plaintext, which means that the synthesis output netlist includes the IP netlist in an unencrypted and readable form. See *Specifying the Script Output Method*, on page 979 for more information.

The script encrypts the IP with the standard symmetric encryption algorithm you specified, and produces a data_block. The data key used for encrypting the HDL is then encrypted with an asymmetric algorithm and the Synopsys public key, and produces a key_block. The data_block and the key_block are combined with the appropriate pragmas for the flow being used, and the script creates an encrypted HDL file. For a detailed figure, see *Encryption and Decryption*, on page 971.

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

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

   The following example encrypts the Verilog plain_ip.v file into an encrypted file called protected_ip.v, using AES128-cbc encryption. The session key is MY_AES_SAMPLEKEY. See *Syntax for Running encryptIP*, on page 1091 in the *Reference Manual* for details about the syntax and required parameters.

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

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

Specifying the Script Output Method

You can control access to the IP by setting the appropriate output method. You specify the output method using the -om parameter, as described in *Syntax for Running encryptIP*, on page 1091 *or* *Syntax for Running encryptP1735*, on page 1082 in the *Reference Manual* .

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

Copyright © 2013 Synopsys, Inc.

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

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

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

   Setting the output method to plaintext allows the tool to synthesize, run gate-level simulations, place and route, and implement an FPGA (that includes the IP) on a board. Setting the output method to blackbox does not allow the tool to run gate-level simulations or place and route the IP, because it only uses the port and connectivity information.
4. If you have set -om to plaintext and you want to specify individual cores as white boxes, set the syn_macro directive to 1 on the view for the IP.

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

Copyright © 2013 Synopsys, Inc.

5. During synthesis, the IP contents appear as a black box in the RTL and Physical Analyst views, irrespective of the output method selected. When the output method is set to plaintext, you can push down into the IP from the Technology view.

6. After synthesis, the output method affects the results in the following ways:
   - Output constraints for an IP are in the standard Synopsys format and are not encrypted.
   - The output method affects the contents of the output netlist and its format. This table summarizes the encryptIP or encryptP1735 behavior with different output methods.

Method (-om)	Output Netlist After Synthesis
blackbox	The output netlist contains the IP interface only, and no IP contents. It only includes IP ports and connections. The IPs are treated as black boxes, and there are no nets or instances shown inside the IP. This applies to all the netlist formats generated for different vendors, whether it is HDL (`vm` or `vhm`), EDIF (`edf` or `edn`), or `vqm`.
plaintext	The output netlist contains your unencrypted synthesized IP, which is completely readable (nothing is encrypted).
persistent_key (encryptIP only)	The output netlist includes encrypted versions of the IP. For Lattice designs, the tool generates one output netlist (EDIF or HDL) that includes the encrypted IP blocks. The netlist is readable, except for the IP block sections, which are encrypted.

Preparing the IP Package

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

1. Collect the files for the package.
   - Encrypt the files you need, as described in *Encrypting Your IP*, on page 975.
   - Make sure your package includes the files listed in *IP Package File List*, on page 983.
   - Structure the files as described in *Suggested Directory Structure*, on page 984.

Copyright © 2013 Synopsys, Inc.

2. For IP-XACT models for use with System Designer (see *Providing System-Level Models for the IP*, on page 985), make sure of the following:
   - At a minimum, include a top-level `xml` file that specifies the complete component, vendor, library, name, and component version (VLNV). This file references library components that are described in other files in the directory tree using relative paths.
   - If you want to allow System Designer to generate HDL files for the IP for later synthesis, include the Verilog/VHDL files for the IP in the package. This allows the core to be evaluated using a synthesis flow.
   - If you do not want to allow System Designer to generate HDL files for synthesis, do not include the Verilog/VHDL files in the package. The System Designer tool creates a top-level netlist and corresponding wrappers and generates an error message for the missing files. This method allows the core to be tested for compatibility with the rest of the system, but it will not be an evaluation with complete synthesis.

   The IP-XACT models consist of a library of your system-level components, including bus definitions. When System Designer reads this library, the various components like specific timers, buses, and CPUs, appear in the library window for the user to drag and drop and instantiate, as they assemble a design from the components. See the System Designer documentation for details.

3. If your IP package is intended for synthesis only, without subsystem assembly, create a compressed package for download, using one of these methods:
   - Create a compressed tarball (.tar.gz), which is a tar archive compressed with the gzip utility, using one of these commands:

     ```
     tar cf -fileList | gzip -c > compressed-tarball
     gtar -cf compressed-tarball fileList
     ```

     Preserve the directory structure when you run gzip.
   - Create a zip file (`zip`) by running WinZip. WinZip archives and preserves your directory hierarchy.

4. If your IP package is intended to be a subsystem that will be assembled by System Designer, create a compressed tarball (.tar.gz) using one of these commands:

   ```
   tar cf -fileList | gzip -c > compressed-tarball
   ```

```
gtar -cf compressed-tarball fileList
```

Preserve the directory structure when you run gzip.

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

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

6. Supply Synopsys with the following:
   - The URL for the download package.
   - Vendor and advertising information you wish to display on the Synopsys website. See *Supplying Vendor Information*, on page 986 for details.

IP Package File List

Your IP package should contain the following files:

Files	Description
ipinfo.txt	Text file that lists the name of the IP, the version, restrictions for use, support contact information, and an email alias to request a licence for the full RTL for your IP.
Documentation, preferably a PDF	Documents the IP, and includes detailed information about usage restrictions like vendor, device family, etc.
Readme	An optional text file that contains instructions on use of the IP for assembly and/or synthesis, and hints on how to use it correctly.
Encrypted HDL or EDIF	Protected RTL for the IP, created using the Synopsys encryptIP script. See the documentation for details.
SDC constraints	Unencrypted design constraints for the IP. You need only maintain a single file for both the Synopsys synthesis tools, as the Synplify Pro software ignores any constraints that are specific to Synplify Premier.
SPIRIT IP-XACT v1.4 models	System-level models for your IP. This allows the synthesis tools to include your IP in a system-level design by stitching the IP together using bus architectures.

Copyright © 2013 Synopsys, Inc.

Suggested Directory Structure

Follow these recommendations when you structure the IP package:

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

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

Providing System-Level Models for the IP

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

This tool reads IP-XACT models (an XML schema) as defined by the SPIRIT Consortium (www.spiritconsortium.org). The current release of System Designer is Spirit v1.4 compliant.

Copyright © 2013 Synopsys, Inc.

Supplying Vendor Information

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

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

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

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

IP name	Name of the IP.
IP short description	Sentence describing the IP, which is displayed in the summary view on the Synopsys website.
IP paragraph description	More detailed description of the IP, covering functional description and compatibility with other cores or peripherals.
Notes about usage	Any other information, like licensing requirements

Copyright © 2013 Synopsys, Inc.

Core datasheet (HTML or PDF)	Information about the characteristics, features, functions, and interfaces.
Supported FPGA vendors and devices	List of the targeted vendors and devices that the core supports.
IP-XACT compatibility information	List of the IP-XACT version number supported, the IP-XACT VLNV, and the IP-XACT VLNVs of all the bus definitions required for the core, along with a link to download each of these bus definitions.

Copyright © 2013 Synopsys, Inc.

Using DesignWare IP

The Synplify Premier and Synplify Premier with Design Planner tools can implement DesignWare IP in FPGA designs from the Synopsys DesignWare foundation library building blocks. The Synopsys foundation library is licensed separately from Synopsys.

Using DesignWare Building Blocks

This section describes how to use the building blocks from the DesignWare foundation library with the Synplify Premier tools. For a list of the available building blocks, see http://www.synopsys.com/dw/buildingblock.php.

- Install the Synopsys DesignWare foundation library included in the Synopsys Design Compiler download; a separate license is required.
- Set the path to the DesignWare foundation library by either:
  - specifying the path to the library using the dc_root *installPath* Tcl command
  - selecting the Implementation Options dialog box and using either the Verilog or VHDL tab to enter the path to the library in the Design Compiler Installation Location [$SYNOPSYS=]: field

Note: To access the DesignWare foundation library from a Windows machine, use the map network drive utility to mount the library on an available Windows drive.

- Enable the use of the DesignWare foundation library by:
  - checking the Use DesignWare Foundation Library checkbox on either the Verilog or VHDL tab
  - entering either of the following Tcl commands

```
set_option -dw_foundation 1
set_option -dw_library {dw_foundation}
```

- If low-power building blocks are to be used in place of standard DesignWare building blocks, enable using the DesignWare minPower library by:
  - checking the Use DesignWare MinPower Library on either the Verilog or VHDL tab (checking the Use DesignWare MinPower Library automatically enables the DesignWare foundation library if not enabled).
  - entering either of the following Tcl commands

    ```
    set_option -dw_minpower 1
    set_option -dw_library {dw_minpower}
    ```

    The above commands do not automatically enable the DesignWare foundation library, and that the foundation library must be explicitly enabled to use the minPower library.
- Set the response to being unable to locate a DesignWare foundation library license according to your design criteria. Checking Stop Synthesis if no DesignWare license found or entering the following Tcl command

  ```
  set_option dw_stop_on_nolic 1
  ```

  stops design synthesis when a DesignWare building block is encountered and a DesignWare foundation library feature license is not found. Conversely, not enabling the checkbox or setting the option to 0 (the defaults) allows synthesis to continue by black boxing each building block in the RTL.

Note: The DesignWare feature license can use the multiprocessing capability of the Synplify Premier tools when multiple processor cores are available. To set the number of licenses, open the Options->Configure Compile Point Options dialog box and set the Maximum number of parallel synthesis jobs value to the desired number of licenses. The normal ratio of license use is one DesignWare license for every two synthesis licenses.

Copyright © 2013 Synopsys, Inc.

DW_Foundation_Arith Package

The Synplify Premier and Synplify Premier with Design Planner tools can implement DesignWare function inferencing using the DW_Foundation_Arith package from the dware library. To use these arithmetic functions in your VHDL code, add the DW_Foundation_Arith package from the dware library using the VHDL use clause as shown below:

```
library dware;
use dware.DW_Foundation_arith.all;
```

To enable this feature, check Use DesignWare Foundation Library on the VHDL tab of the Implementation Options dialog box.

Example:

```
library ieee,dware;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use dware.DW_foundation_arith.all;

entity ew_002_sus is
   port (a : in signed(3 downto 0);
         b : in unsigned(3 downto 0);
         c : out signed(3 downto 0));
end entity;

architecture behv of ew_002_sus is
begin
   c <= a / b;
end architecture;
```

The supported DesignWare functions in the DW_Foundation_Arith package are listed in the following table.

Division Functions

function "/" (A : signed; B : unsigned) return signed;
function "/" (A : unsigned; B : signed) return signed;
function "/" (A : unsigned; B : unsigned) return std_logic_vector;
function "/" (A : signed; B : signed) return std_logic_vector;
function "/" (A : signed; B : unsigned) return std_logic_vector;
function "/" (A : unsigned; B : signed) return std_logic_vector;

Remainder Functions

function "rem" (A : signed; B : unsigned) return signed;
function "rem" (A : unsigned; B : signed) return signed;
function "rem" (A : unsigned; B : unsigned) return std_logic_vector;
function "rem" (A : signed; B : signed) return std_logic_vector;
function "rem" (A : signed; B : unsigned) return std_logic_vector;
function "rem" (A : unsigned; B : signed) return std_logic_vector;

Modulo Functions

function "mod" (A : signed; B : unsigned) return signed;
function "mod" (A : unsigned; B : signed) return signed;
function "mod" (A : unsigned; B : unsigned) return std_logic_vector;
function "mod" (A : signed; B : signed) return std_logic_vector;
function "mod" (A : signed; B : unsigned) return std_logic_vector;
function "mod" (A : unsigned; B : signed) return std_logic_vector;

Binary Encoder Functions

function DW_binenc(A: SIGNED; ADDR_width: NATURAL) return SIGNED;
function DW_binenc(A: UNSIGNED; ADDR_width: NATURAL) return UNSIGNED;
function DW_binenc(A: std_logic_vector; ADDR_width: NATURAL) return std_logic_vector;

Decoder Functions

function DW_decode(A: SIGNED) return SIGNED;
function DW_decode(A: UNSIGNED) return UNSIGNED;
function DW_decode(A: std_logic_vector) return std_logic_vector;

Priority Encoder Functions

function DW_prienc(A: SIGNED; INDEX_width: NATURAL) return SIGNED;
function DW_prienc(A: UNSIGNED; INDEX_width: NATURAL) return UNSIGNED;
function DW_prienc(A: std_logic_vector; INDEX_width: NATURAL) return std_logic_vector;

Copyright © 2013 Synopsys, Inc.

Working with Synenc-encrypted IP

Synenc-encrypted IP is accepted as input by Synplify Premier when the tool is running on a Linux platform. This capability allows Synopsys-encrypted RTL cores, configured and generated using the Synopsys coreTools, to be synthesized by Synplify Premier. The capability is extended to include DesignWare library macrocells; a Synopsys DesignWare license is required when the synenc-encrypted cores are generated. Proprietary RTL cores encrypted using the Synopsys coreTools are also supported. Note that the synenc-encrypted data cannot include any "licensed" components.

When importing a Synplify project containing synenc-encrypted DesignWare cores created with the coreConsultant, the files listed in the project are in their correct order. This project file is then inserted into Synplify Premier as a subproject using the project -insert command.

When working with existing synenc-encoded source files without going back to the coreConsultant to create a project file, the core files can be added directly to a Synplify Premier project. When adding files manually, the order of files is critical. To ensure proper file order, either:

- use the original `lst` file from coreConsultant to setup your project. The `lst` file gives the proper order of files. The normal path to the `lst` file is:

  ip_core_name/**src**/*ip_core_name*.**lst**

- if the `lst` file is unavailable, make sure that the params and constants files for each core are listed first and make sure that the undef file for the core appears last.

When the core files are not in the correct order, the Synplify compiler errors out complaining about unknown macros.

Using Hyper Source

Hyper source is a useful feature that lets you prototype ASIC designs that use one or more FPGAs. You can also use it to validate and debug the RTL for IP designs. See the following for more information:

- Using Hyper Source for Prototyping, on page 993
- Using Hyper Source for IP Designs, on page 993
- Threading Signals Through the Design Hierarchy of an IP, on page 994

Using Hyper Source for Prototyping

For prototyping, use hyper source to address the following issues:

- Use it to efficiently thread nets across multiple modules to the top-level design to support Time Domain Multiplexing (TDM).
- Use it to easily replace an ASIC RAM with an FPGA RAM.

Follow these guidelines to replace an ASIC RAM with an FPGA RAM:

1. Change the RTL for the RAM instantiation.
2. Add an extra clock signal to all the module interfaces.

   Hyper source reduces the number of modified RTL modules to two: one for the RAM and one for the top level.

Using Hyper Source for IP Designs

For IP designs, hyper source is useful for validating and debugging the RTL, without directly modifying the RTL. After the RTL has been fully tested with complete QoR results, use hyper source to debug. For example:

- Add some instrumentation logic that is not part of the original design, such as a cache profiler that counts cache misses or bus monitor that might count statistics about bus contention. The cache or bus might be buried deep inside the RTL; accessing the cache or the bus means ports might need to be added through several levels of hierarchy in the RTL. The instrumentation logic can be included anywhere in the design, so you can use hyper source and hyper connect to easily thread the necessary connections during synthesis.

Copyright © 2013 Synopsys, Inc.

- Insert other hyper sourcing inside the IP to probe, monitor, and verify correct operation of known signals within the IP.

Threading Signals Through the Design Hierarchy of an IP

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

However, if you use hyper source to thread signals through an EDIF, you must use the Synplify Premier tool. You can run the software with or without Physical Synthesis, but turn on the Enhanced Optimization switch.

The following procedure describes a method for using hyper source, using the example HDL shown in *Hyper Source Example*, on page 995.

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

- Signal syn_hyper_connect connect_block (probe) module is defined for the connection with a width of 8.
- Tag name of "probe_sig" must match the name used in the hyper source block to thread the signal properly.

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

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

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

Hyper Source Example

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

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

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

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


Copyright © 2013 Synopsys, Inc.

```
defparam my_source.w = 8;

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

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

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

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

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

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

endmodule
```

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

RTL View

Technology View

Copyright © 2013 Synopsys, Inc.

Working with Altera IP

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

- Using Altera LPMs or Megafunctions in Synthesis, on page 998
- Implementing Megafunctions with Clearbox Models, on page 1002
- Implementing Megafunctions with Grey Box Models, on page 1012
- Including Altera MegaCore IP Using an IP Package, on page 1019
- Including Altera Processor Cores Generated in SOPC Builder, on page 1023
- Working with SOPC Builder Components, on page 1029

For information about working with Megacore IPs in imported Quartus designs, see *Importing Quartus Designs with Megacore IPs*, on page 1038.

Using Altera LPMs or Megafunctions in Synthesis

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

- Generate structural Verilog/VHDL for the IP and include it in your design, as described in *Recommended Method for Including Altera LPMs*, on page 999.
- For newer Altera technologies, the synthesis tool can infer the Clearbox or greybox megafunctions as described in *Automatically Inferring Megafunctions with Clearbox Information*, on page 1003 and *Using Clearbox Information for Instantiated Megafunctions*, on page 1007
- For older Altera technologies, if you have a Clearbox netlist generated by the Quartus tool for the IP, include the Clearbox netlist in the project, and synthesize the design. See *Instantiating Clearbox Netlists for Megafunctions*, on page 1010.
- Infer Altera LPMs or megafunctions included in the *installDirectory*/lib/altera directory. This lets the Synplify Pro and Synplify Premier tools access supported LPMs and Megafunctions as required. Note that if you are using a VHDL component from a non-default Quartus library,

you must set the Quartus version and add the library file you want to use to the `prj` file with an -add file command.

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

Recommended Method for Including Altera LPMs

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

1. Use the Altera MegaWizard Plug-in Manager to generate structural Verilog or VHDL files for the LPMs or megafunctions. See *LPM / Megafunction Example*, on page 1001 for an example.

2. Add the structural Verilog or VHDL files in the project. For the example, you would add my_ram.v file to the project.

3. Instantiate the LPMs or megafunctions with a wrapper in your top-level HDL source code.

   For the example in *LPM / Megafunction Example*, on page 1001, instantiate the megafunction wrapper, my_ram, in the top-level HDL source code.

```
module top (
...

);

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

endmodule  // top
```

4. Select the correct Quartus version.

Copyright © 2013 Synopsys, Inc.

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

```
qmegawiz -silent
```

5. Synthesize the design.

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

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

```
stratixii_ram_block altsyncram_component_ram_block1a_0_0_0_Z (
    .portadatain({data_c[0]}),
    .portaaddr({address_c[7], address_c[6], address_c[5], address_c[4],
    address_c[3], address_c[2], address_c[1], address_c[0]}),
    .portawe(wren_c),
    .clk0(clock_c),
    .portadataout({q_c[0]})
);
defparam altsyncram_component_ram_block1a_0_0_0_Z.connectivity_checking =  "OFF";
defparam altsyncram_component_ram_block1a_0_0_0_Z.init_file =
    "C:/public/qinghong/ram_init/rev_1/init_values.mif";
defparam altsyncram_component_ram_block1a_0_0_0_Z.init_file_layout =  "port_a";
defparam altsyncram_component_ram_block1a_0_0_0_Z.logical_ram_name =  "ALTSYNCRAM";
defparam altsyncram_component_ram_block1a_0_0_0_Z.operation_mode =  "single_port";
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_address_width =  8;
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_data_out_clear =  "none";
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_data_out_clock =
    "clock0";
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_data_width =  1;
defparam
    altsyncram_component_ram_block1a_0_0_0_Z.port_a_disable_ce_on_input_registers =
    "on";
defparam
    altsyncram_component_ram_block1a_0_0_0_Z.port_a_disable_ce_on_output_registers
    =  "on";
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_first_address =  0;
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_first_bit_number =  0;
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_last_address =  255;
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_logical_ram_depth
    =  256;
defparam altsyncram_component_ram_block1a_0_0_0_Z.port_a_logical_ram_width =  4;
defparam altsyncram_component_ram_block1a_0_0_0_Z.power_up_uninitialized =
    "false";
defparam altsyncram_component_ram_block1a_0_0_0_Z.ram_block_type =  "M512";
defparam altsyncram_component_ram_block1a_0_0_0_Z.lpm_type
    =  "stratixii_ram_block";
```

LPM / Megafunction Example

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

```
/ megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

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


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


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

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

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

    altsyncram altsyncram_component (
        .wren_a (wren),
```


Copyright © 2013 Synopsys, Inc.

```
        .clock0 (clock),
        .address_a (address),
        .data_a (data),
        .q_a (sub_wire0),
        .aclr0 (1'b0),
        .aclr1 (1'b0),
        .address_b (1'b1),
        .addressstall_a (1'b0),
        .addressstall_b (1'b0),
        .byteena_a (1'b1),
        .byteena_b (1'b1),
        .clock1 (1'b1),
        .clocken0 (1'b1),
        .clocken1 (1'b1),
        .clocken2 (1'b1),
        .clocken3 (1'b1),
        .data_b (1'b1),
        .eccstatus (),
        .q_b (),
        .rden_a (1'b1),
        .rden_b (1'b1),
        .wren_b (1'b0) );
    defparam
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_output_a = "BYPASS",
        altsyncram_component.init_file = "init_values.mif",
        altsyncram_component.intended_device_family = "Stratix II",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 256,
        altsyncram_component.operation_mode = "SINGLE_PORT",
        altsyncram_component.outdata_aclr_a = "NONE",
        altsyncram_component.outdata_reg_a = "CLOCK0",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.ram_block_type = "M512",
        altsyncram_component.widthad_a = 8,
        altsyncram_component.width_a = 4,
        altsyncram_component.width_byteena_a = 1;

endmodule
```

Implementing Megafunctions with Clearbox Models

Generally, user-instantiated Quartus megafunctions do not have timing information and are treated as black boxes, so the synthesis tool cannot optimize timing at the megafunction boundary. For example, the tool does not move the registers of a pipelined LPM_MULT to improve FMAX. Instead of black boxes, you can implement the megafunctions as grey boxes (see *Implementing Megafunctions with Grey Box Models*, on page 1012) or clear boxes, as described here. (This flow is supported in the Synplify Pro and Synplify Premier tools.)

Altera Clearbox netlists provide structural information for modules containing the following primitives lcell, mac_mult, mac_out, and ram_block. The Clearbox netlist is a fully synthesizeable Altera megafunction. When you

synthesize with a clear box model, you get better timing and resource utilization estimates, because the synthesis tool knows the architectural details used in the Quartus II software. For details, see the following:

- Automatically Inferring Megafunctions with Clearbox Information, on page 1003
- Using Clearbox Information for Instantiated Megafunctions, on page 1007
- Instantiating Clearbox Netlists for Megafunctions, on page 1010

Automatically Inferring Megafunctions with Clearbox Information

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

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

   The following table lists some tips for controlling inference:

Multipliers in DSP blocks	Use syn_multstyle to control inference.

Copyright © 2013 Synopsys, Inc.

ROMs	• The address line must be at least two bits wide. • The ROM must be at least half full. • A CASE or IF statement must make 16 or more assignments using constant values of the same width.
Shift registers	Use syn_srlstyle to control inference.
RAMs	• The address line must be at least two bits wide. • Do not have resets on the memory. • Check whether read and write ports must be synchronous for your target family. • Avoid blocking statements when modeling the RAM, because not all Verilog HDL blocking assignments are mapped to RAM blocks. • Use syn_ramstyle to control inference. • Use $readmemb or $readmemh to initialize RAMs.

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

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

Synplify Pro	• Stratix II, Stratix III, Stratix IV, and Stratix V • Arria GX, Arria II, and Arria V • Cyclone V
Synplify Premier (placement)	Stratix II, Stratix III, and Stratix IV

– Set the Altera Models device option.

To generate vqm that contains...	Set it to...
The Altera primitives associated with the inferred or instantiated Megafunction	on
The Altera primitives associated with the inferred Megafunction only	clearbox_only
The instantiated Megafunction only	off

– Click OK.

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

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

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

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

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

endmodule
```

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

Copyright © 2013 Synopsys, Inc.

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

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

Using Clearbox Information for Instantiated Megafunctions

There are two ways of instantiating megafunctions and using Clearbox information in your synthesis design. The following is the recommended method for instantiation in the newer Altera technologies, where you instantiate just the megafunction, and the tool automatically infers the corresponding Clearbox details. For older technologies, instantiate the Clearbox netlist for the megafunction, as described in *Instantiating Clearbox Netlists for Megafunctions*, on page 1010.

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

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

2. Set up the megafunction for synthesis.
   - Instantiate the megafunction in your synthesis design.
   - Add the megafunction wrapper file to your project file.

3. Set up the synthesis tool to use the Clearbox information automatically.
   - Make sure the QUARTUS_ROOTDIR environment variable is set and pointing to the same Quartus version as the library.
   - In the synthesis tool, open the Implementation Options dialog box to the Device tab, and set the synthesis tool target to a supported family:

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

   - Set the Altera Models device option.

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

   - Click OK.

Copyright © 2013 Synopsys, Inc.

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

Copyright © 2013 Synopsys, Inc.

Instantiating Clearbox Netlists for Megafunctions

For older Altera technologies which do not supported the automatic inference of Clearbox information (see *Automatically Inferring Megafunctions with Clearbox Information*, on page 1003 and *Using Clearbox Information for Instantiated Megafunctions*, on page 1007), you can read in a Clearbox netlist. You only need to use the netlist method described here for older Altera target families that do not support the other flows.

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

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

   - Make sure the Clearbox components match the Quartus version. Because of ongoing modifications in Quartus, the component declarations may not match. The component declarations are packaged with the software in the lib/altera/quartus_II*nn* subdirectory. Use the file from the subdirectory that corresponds to the Quartus version that you are using. For example, the stratix.vhd and stratix.v files for use with Quartus 8.1 are in the quartus_II81 subdirectory.
   - If you change from one version of Quartus to another or if you change the target device, regenerate the Clearbox files using the Altera Megafunction wizard before proceeding. Failure to regenerate these files can result in a parameter-mismatch error.
2. Instantiate the megafunction in your design.

Copyright © 2013 Synopsys, Inc.

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

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

   When you run Quartus from the synthesis UI, the Clearbox files must be in the *implementation*/par_1 directory, which is only created after the synthesis run. Specifying the Clearbox files ensures that they are copied to the directory after it is created. You can now synthesize the design, as described in the previous step.
6. Set any other options and synthesize the design.

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

Copyright © 2013 Synopsys, Inc.

Technology views both show the lowest-level primitives. The following figure for example, shows stratixii_ram_blocks.

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

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

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

Implementing Megafunctions with Grey Box Models

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

cannot optimize timing at the megafunction boundary. Instead of using black boxes, you can implement the megafunctions using Clearbox primitives (see *Implementing Megafunctions with Clearbox Models*, on page 1002) or as grey boxes, as described here. Use the grey box methodology when logic is encrypted or when there are no Clearbox models for the megafunction.

There are three ways to use grey boxes:

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

- Automatically Using Grey Box Information for Megafunctions, on page 1013
- Using Grey Box Information for Instantiated Megafunctions, on page 1014
- Instantiating Megafunctions Using Grey Box Netlists, on page 1016

Automatically Using Grey Box Information for Megafunctions

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

Copyright © 2013 Synopsys, Inc.

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

Synplify Pro	Stratix II and later, Arria II, Arria GX, Arria GZ
Synplify Premier (placement)	Stratix II and later

- To use grey box timing information, set Altera Models device option to on.
- Click OK.

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

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

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

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

Using Grey Box Information for Instantiated Megafunctions

There are two ways of instantiating grey box megafunctions in your synthesis design. The following procedure shows you how to instantiate a grey box megafunction without a grey box netlist; to instantiate one with a grey box netlist, refer to the procedure in *Instantiating Megafunctions Using Grey Box Netlists*, on page 1016.

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

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

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

Synplify Pro	Stratix II and later, Arria II, Arria GX, Arria GZ
Synplify Premier (placement)	Stratix II and later

   - To use grey box timing information, set Altera Models device option to on.
   - Click OK.
4. Set any other options you want, and click Run to synthesize the design.

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

   The tool instantiates the megafunction in the output vqm netlist, but does not include the grey box timing information.
5. To place and route in Quartus II, use the following files:
   - The synthesis vqm output netlist, which includes an empty instantiation of the megafunction.
   - The encrypted file for the megafunction.

Copyright © 2013 Synopsys, Inc.

Instantiating Megafunctions Using Grey Box Netlists

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

1. Make sure you are using Quartus II 10.0 or later.
2. Use the Altera MegaWizard tool to generate the files for the IP core and a grey box netlist. The following example shows the files needed for a project:

top.v	Top level design file. See *top.v*, on page 1023.
my_ip_core.v	Variation file (top-level wrapper) generated by the MegaWizard tool, which instantiates the encrypted module encrypted_ip.v. See *my_ip_core.v*, on page 1023. Do not add this file to the project.
encrypted_ip.v	Encrypted IP core that is not readable. Do not add this file to the project.
my_ip_core_syn.v	A timing and resource estimation netlist (grey box file) for use in synthesis. Add this file to the project.
my_ip_core.qip	A file that contains links to IP-related files. You must have the referenced files to complete place and route. This file must reside in the same directory as all other files generated by the MegaWizard tool. Add a reference to this file in altera_par.tcl.

   - To generate the grey box netlist, enable the Generate netlist option in the Megawizard tool when you set up simulation. This is usually under the EDA tab or in the Set Up Simulation section. The grey box netlist provides the logic connectivity of specific mapped instances, but does not represent the true functionality of the MegaCore IP.
   - Parameterize the IP core and generate the IP files. The tool outputs a grey box file (_syn.v) along with the other synthesis files.
3. Set up your design.
   - Instantiate the my_ip_core component in your HDL code.
   - In the synthesis tool, add the grey box netlist file, my_ip_core_syn.v to your synthesis project.

Copyright © 2013 Synopsys, Inc.

4. Set up a reference to the my_ip_core.qip file by doing the following:
   - Create a file called altera_par.tcl and add a reference to the `qip` file. Make sure that the path is relative to your `PAR` directory. The following example shows a path if if your `qip` file is at the same level as the project file:

     ```
     set_global_assignment -name QIP_FILE ../../my_ip_core.qip
     ```

   - Add altera_par.tcl to your project.
   - Right-click the file in the Project view and select File Options. Set File Type to Altera P&R Options and click OK.

Copyright © 2013 Synopsys, Inc.

– Add a new implementation.

– Click Implementation Options. In the Implementation Options dialog box, click P&R Options and enable the altera.par.tcl file.

5. Set other implementation options.

   – Set the target technology on the Device tab.

   – Go to the Implementation Results tab and specify the correct version for the place-and-route tool. This is important because the version determines the format for the `vqm` output file, which varies with different versions.

   – Set any other options or constraints you want.

6. Synthesize the design.

   The synthesis tool uses the grey box netlist file for synthesis and timing analysis. The RTL and Technology views show the internals of the core

IP, because the grey box netlist file contains mapped instances. The log file reports any critical paths found within the core, and the Technology view displays timing numbers and critical paths.

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

7. Place and route the design in Quartus.

   When you run integrated Place & Route, the `qip` file is referenced in the Quartus settings file (`qsf`). This informs Quartus of the location of the IP-related files needed to complete PAR.

Including Altera MegaCore IP Using an IP Package

You can include MegaCore® IP cores generated with the Altera MegaWizard tool using the method described below, but it is preferred that you use a greybox netlist instead (*Instantiating Megafunctions Using Grey Box Netlists*, on page 1016).

For information about including cores generated with SOPC Builder, see *Including Altera Processor Cores Generated in SOPC Builder*, on page 1023.

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

top.v	Top level design file. See *top.v*, on page 1023.
my_ip_core.v	Top-level wrapper generated by the MegaWizard tool, which instantiates the encrypted module encrypted_ip.v. See *my_ip_core.v*, on page 1023.
my_ip_core_enc.v	Encrypted IP core that is not readable.

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

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

Copyright © 2013 Synopsys, Inc.

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

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

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

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

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

6. Instantiate the NIOS core in the top-level file for your synthesis design.

7. Synthesize the design.

Copyright © 2013 Synopsys, Inc.

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

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

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

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

Examples of MegaCore IP Files for Synthesis

top.v

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

```
module top (in1, in2, out1, out2)
input in1, in2;
output out1, out2;
my_ip_core (in1, in2, out1, out2);
endmodule;
```

my_ip_core.v

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

```
module my_ip_core (in1, in2, out1, out2)
input in1, in2;
output out1, out2;
encrypted_ip (in1, in2, out1, out2);
endmodule;
```

Including Altera Processor Cores Generated in SOPC Builder

Altera provides the capability of implementing configurable processor cores. NIOS® II cores are created using the Altera SOPC Builder tool, which uses proprietary RTL code to parameterize, generate, and instantiate NIOS II cores in your design. The following procedure shows you how to use the greybox flow to incorporate these cores in a Synplify Pro or Synplify Premier design targeting certain Altera technologies. For information about including MegaCore cores, see *Instantiating Megafunctions Using Grey Box Netlists*, on page 1016.

1. Make sure the following requirements are in place:
   - Install Quartus II 7.2 or later.
   - Install the MegaCore® IP library.
2. Generate the NIOS II core files as follows:
   - In Altera SOPC Builder, generate Verilog or VHDL output for the NIOS II core files. Note that this flow does not support a block symbol

Copyright © 2013 Synopsys, Inc.

file (`bsf`); you must generate RTL code for the cores. The files may or may not be encrypted.

– Create a top-level wrapper for the core. In most cases, this wrapper instantiates the components used to create the embedded system. For example, if the embedded system consists of a NIOS II processor that uses a PCI bus to interfaces to internal memory, the wrapper would contain instantiations for the processor, memory and the PCI bus.

– Copy all generated IP core output files and the corresponding library files into a single directory. Typically you must include the generated HDL files, as well as any MegaCore IP cores in your design. Depending on your design, the MegaCore IP files can be in the MegaWizard IP tool output directory and in the IP library files in the Quartus install directory (altera/72/ip/pci_compiler/lib).

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

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

Copyright © 2013 Synopsys, Inc.

File	Description
onchip_mem.v	Module instantiated in top-level wrapper (SOPC Builder)
sys_clk_timer.v	Module instantiated in top-level wrapper (SOPC Builder)
sysid.v	Module instantiated in top-level wrapper (SOPC Builder)

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

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

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

Copyright © 2013 Synopsys, Inc.

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

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

5. Instantiate the NIOS core in the top-level file for your synthesis design.

6. Synthesize the design.

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

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

Copyright © 2013 Synopsys, Inc.

```
Worst Path Information
***********************

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

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

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

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

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

Working with SOPC Builder Components

You can incorporate Altera SOPC cores into your FPGA design. The following figure summarizes the process.

See the following for step-by-step details:

- The Synplify-SOPC Builder Design Flow, on page 1029
- Setting up the SOPC Builder Synthesis Project, on page 1030
- Defining SOPC Components as Black Boxes and White Boxes, on page 1031

The Synplify-SOPC Builder Design Flow

The following procedure describes the Synplify-SOPC Builder design flow step by step. The previous figure provides a graphic overview of the flow.

1. Create your embedded design, using the Altera SOPC Builder- Quartus Hardware Development flow. See the Altera documentation for details.
2. Import the SOPC Builder project into the Synplify Pro or Premier tool. See *Setting up the SOPC Builder Synthesis Project*, on page 1030 for details.
3. Synthesize, place, and route the design.
4. Import the bitmap file back into the Quartus environment.

Copyright © 2013 Synopsys, Inc.

- Close the open project in Quartus.
- In the Quartus tool, select File ->Open Project. Browse to the synthesis PAR directory (par_1) and select the `qpf` file: synplify/rev_1/par_1/*system*.qpf.
- Select Processing -> Start Compilation and compile the project.
- If you have a Nios II license, the tool generates a programming file called small_eval_board_time_limited.sof in the par_1 directory. You can now download the file into the FPGA.

Setting up the SOPC Builder Synthesis Project

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

1. Start the Synplify Pro/Premier software, and select Import IP->Import Altera SOPC Project.

   The Import Altera Design dialog box opens.

2. Do the following to import the SOPC project you created with the Altera tools.
   - Specify the Altera `ptf` file you want to import in the PTF File option.
   - Specify a location for the synthesis project to be created in Project Location, and a name for the project in Project Name.
   - Set the options you want in the Import Options section. Define the black boxes and white boxes in your design with the Clear Box and Force White Box Cores options (see *Defining SOPC Components as Black Boxes and White Boxes*, on page 1031 for details).

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

   For descriptions of the import options, see *Import Altera SOPC Project Command*, on page 228 of the *Reference Manual*. The software uses the underlying sopc2syn functionality (see *Altera SOPC2Syn Utility*, on page 998 in the *Reference Manual*) to read the Altera files and include the information from them into the synthesis project.

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

Defining SOPC Components as Black Boxes and White Boxes

Accurately defining SOPC components as black boxes and white boxes is very important, because the synthesis run fails if the design is not correctly specified, and you will have to iterate through another run.

The following table lists definitions for black boxes and white boxes:

Copyright © 2013 Synopsys, Inc.

Black box	A component that does not have any definitions.
White box	A component that has a core definition in `vqm` netlist format.

This procedure describes how to define black boxes and white boxes:

1. In the Synplify Pro/Premier tool, select Import IP->Import Altera Project. This opens the Import Altera Design dialog box.

2. To define a component as a black box, do the following:
   - List the core in Force White Box Cores.
   - Disable the Clear Box option.

   With these settings, the tool copies the core wrapper file to the Synplify folder and edits it to add the black box attribute. The component is

treated as a black box during synthesis. If the tool does not find the named component, it issues a warning message.

3. To define a core as a white box, do the following:
   - List the core in Force White Box Cores.
   - Enable the Clear Box option.

   With these settings, the tool goes through the Clearbox flow, and treats the core as a white box during synthesis. It generates a definition `vqm` file and adds this file to the Synplify project. If it does not find the specified core in the `ptf` file, or the Clearbox file for the core, it issues a warning message.

Importing Projects from Quartus

You can take a Quartus project and automatically translate it into a synthesis project. If you have a SOPC Builder project, use the procedure described in *Working with SOPC Builder Components*, on page 1029. To import projects from Quartus, see the following:

- Importing Quartus Projects, on page 1033
- Importing Quartus Designs with Megacore IPs, on page 1038
- Importing Quartus Designs with Megafunctions/LPMs, on page 1039
- Troubleshooting Imported Quartus Designs, on page 1040

Importing Quartus Projects

This procedure shows you how to translate a Quartus project into a project that you can synthesize in the FPGA synthesis tools. It takes the Quartus project settings and constraints from the `qsf` and `sdc` files and uses the underlying qsf2syn utility to generate a synthesis project. See *Altera qsf2syn Utility*, on page 1003 in the *Reference Manual* for the syntax for the utility.

1. Set up your Quartus design and run Quartus.
   - Check that the Quartus version used matches the one specified in the QUARTUS_ROOTDIR environment variable for synthesis. It is recommended that you use the newer versions of Quartus.

Copyright © 2013 Synopsys, Inc.

- Run Quartus and successfully complete a Quartus run on the design.
- Read through *Troubleshooting Imported Quartus Designs*, on page 1040 and make sure you are following the tips listed there.
- If your design contains Altera Megacore IP files, also read through *Importing Quartus Designs with Megacore IPs*, on page 1038 for additional information on incorporating the IP.
- If you have a SOPC Builder project, use the sopc2syn utility described in *Working with SOPC Builder Components*, on page 1029.

2. In the synthesis tool UI, select Import->Import Altera QSF Project.

   The Import Altera QSF Project dialog box opens.

3. Do the following in this dialog box:
   - In QSF File, specify the `qsf` file of the Quartus project you want to import.
   - In the Synplify Project Location section, specify the name and location of Synplify project file to create.

– Click the Import button at the bottom.

This runs the qsf2syn utility, which does the following:

– Creates a synthesis project file (`prj`) from the referenced source files, synthesis settings, and timing constraints in the Quartus `qsf` and `sdc` files.

– Saves the project file to the specified location.

– Writes the attributes and timing constraints to an `sdc` file.

You can run synthesis using the generated project.

See *Synthesis Files Generated After Importing the Quartus Project*, on page 1036 for a description of the files generated after the Quartus project is successfully imported. For a list of the Quartus settings that are translated, see *Imported Quartus Project Settings and Timing Constraints*, on page 1037.

4. To debug problems that might occur when you first import a Quartus project, especially one with multiple source files, do the following:

   – Check the qsf2syn.log file in the project directory to get details about the errors in the conversion run.

   – If possible, correct the error in the currently loaded project.

   – Check the timing_unapplied.sdc file for improperly translated timing constraints.

   See *Troubleshooting Imported Quartus Designs*, on page 1040 for tips on dealing with improperly translated constraints.

5. To resume the import process after you have fixed the errors that caused translation to fail, select Import->Import Altera QSF Project in the synthesis UI, and enable the Continue Previous Translation option.

   This allows you to continue with the translation after you have fixed the compilation errors.

Copyright © 2013 Synopsys, Inc.

Synthesis Files Generated After Importing the Quartus Project

Once the tool finishes importing the Quartus project, it automatically loads the new synthesis project. The project view contains the following:

Folder	Contains...
VHDL/Verilog	HDL source files referenced in the Quartus `qsf` file
Constraint	Translated constraints referenced in the Quartus `qsf` and `sdc` files. See *Imported Quartus Project Settings and Timing Constraints*, on page 1037 for a list of constraints. • attr_applied.sdc Contains translated attributes from the Quartus `qsf` file • attr_unapplied.sdc Contains translated attributes from the Quartus `qsf` file which are unsupported or failed to match an object in the HDL source. • timing_applied.sdc Contains translated timing constraints from the Quartus `qsf` or `sdc` files. • timing_unapplied.sdc Contains translated timing constraints from the Quartus `qsf` or `sdc` files that failed to match objects in the HDL source. As constraints can be applied at various points throughout the Quartus flow, it may be difficult to match up the original HDL object names during translation. Object names that cannot be traced back to the HDL source name are written to this file. These `sdc` files are physically located in the *projectName*_sdc directory. To see the full path, right-click a file and select File options.
Altera P&R Options	The *projectName*_par_options.tcl file, which contains Quartus project settings that were not translated for the synthesis project. This file is sourced and its contents are written to the Quartus `qsf` file when you run P&R.For IPs, it references the variation (top-level IP wrapper file) and `qip` files.

Imported Quartus Project Settings and Timing Constraints

The following table lists how the synthesis tools handle the import of Quartus constraints and project settings:

Project Settings (Translated in synthesis project file)	FAMILY DEVICE TOP_LEVEL_ENTITY VERILOG_FILE VHDL_FILE SYSTEMVERILOG_FILE SDC_FILE USE_TIMEQUEST_TIMING_ANALYZER EXTRACT_VERILOG_STATE_MACHINES EXTRACT_VHDL_STATE_MACHINES SYNTH_GATED_CLOCK_CONVERSION SOURCE *qsf_fileName* USER_LIBRARIES SEARCH_PATH
Constraint Settings (Translated in attr_applied.sdc file)	IO_STANDARD PIN_<xx> MAX_FANOUT FAST_INPUT_REGISTER FAST_OUTPUT_REGISTER PRESERVE_REGISTER STATE_MACHINE_PROCESSING SAFE_STATE_MACHINE

Copyright © 2013 Synopsys, Inc.

TimeQuest Supported Constraints	create_clock create_generated_clock set_clock_groups set_input_delay set_output_delay set_false_path set_multicycle_path set_max_delay

Untranslated settings are written to the par_options.tcl file and passed to Quartus.

Importing Quartus Designs with Megacore IPs

If your design contains Megacore IPs, use the following procedure:

1. When you generate the Megacore IP in the Altera MegaWizard tool, do the following:
   - Generate each IP in a separate directory.
   - Generate a timing and resource netlist (_syn.v) for each IP.

   It is strongly recommended that you generate this file during IP creation, by going to the EDA tab and enabling the Timing and Resource Netlist option. Although the translation process will open the MegaWizard and prompt you to generate the file if you have not already done so, it is recommended that the file be generated during IP creation instead of later.

   Ensure that this netlist file is in the same directory as the corresponding `qip` file for the IP.

2. Set up the Quartus project so that it follows the IP flow recommended by Altera.
   - When Quartus prompts you, allow the tool to automatically add the associated `qip` file for the IP to the Quartus project. This file contains pointers to all related IP files. Do this for each IP in the design.
   - Do not add the IP-related files directly to the Quartus project. If you do have IP-related files in the Quartus project along with the `qip` file,

the qsf2syn utility attempts to omit the IP-related files when it creates the synthesis project.

Note that the software automatically translates older Quartus projects that do not follow Altera's current flow (where IP-related files are added to the Quartus project and there are no `qip` files) when you import the design as described in step 4.

3. Check the following:
   - Verify that you have a _syn.v timing and resource netlist for each IP.
   - Verify that every IP has an associated `qip` file listed in the `qsf` file. If you are using an older Quartus project, you will not have this file.
   - Verify that the IP-generated files listed in the `qip` file are not referenced in the `qsf` file. If you are working with an older Quartus project, these files may be in the `qsf` (Quartus settings) file.
4. Import the design with the Import Altera QSF command, as described in *Importing Quartus Projects*, on page 1033.

   The process creates a new synthesis project and loads it in the project view.

Importing Quartus Designs with Megafunctions/LPMs

If your design contains megafunctions or LPMs, use the following procedure to create a synthesizable project from the Quartus project:

1. When you generate the megafunction/LPM in the Altera MegaWizard tool, do the following.
   - Generate each megafunction or LPM in a separate directory.
   - Generate IP. You do not need to generate a timing and resource netlist (_syn.v) file.
2. Set up the Quartus project so that it includes the top-level variation file of the megafunction/LPM.

   If you prefer to use the `qip` file instead of the top-level variation file generated by the Megawizard tool, the qsf2syn utility automatically looks for the top-level variation file in the same directory as the `qip` file. It then automatically adds this file to the synthesis project it creates.

Copyright © 2013 Synopsys, Inc.

3. Import the design with the Import Altera QSF command, as described in *Importing Quartus Projects*, on page 1033.

Troubleshooting Imported Quartus Designs

The following table lists some problems and their solutions:

Problem	Solution
Qsf2Syn does not translate SOPC Builder projects	Use the procedure described in *Working with SOPC Builder Components*, on page 1029 to translate these projects.
Timing constraints on RTL objects that infer RAMs are not applied properly	Find the constraints in the timing_unapplied.sdc file. Use the RTL viewer to find proper object names and manually correct the names on these constraints.
Constraints applied to RTL objects in `generate` statements are not applied properly	The tool does not translate constraints applied on `generate` statements. Check the constraints in timing_unapplied.sdc and manually correct the object referenced by the constraint.
Constraints applied on 2-D array objects are not translated	Manually create the constraints, using the RTL viewer to find the proper object names.
Constraints embedded in HDL source code are not translated correctly	Manually add the corresponding constraints found in timing_unapplied.sdc. Use the RTL viewer to find the correct object names.
You get an error message about an undefined macro	Add `define *macroName* to the HDL. You see this message if the HDL contains `*macroName* without the `define *macroName* statement. Although Quartus does not require the define statement, the translation process requires it.
State machine contains blocking and non-blocking assignments in the same variable	Modify the source code so that blocking and non-blocking assignments are not combined in the same variable.
You get a compiler error message about undeclared VHDL libraries	Declare the library in a VHDL `use` clause. Although Quartus does not require the library to be declared, the FPGA synthesis compiler requires this declaration.

Copyright © 2013 Synopsys, Inc.

Problem	Solution
The get_keepers qualifier is translated without a qualifier	Currently, this is the expected behavior. The tool attempts to match an object using all qualifiers. If it finds an object, it puts the translated constraint in _applied.sdc.
The `qip` file generated by Altera Megawizard tags the top-level IP wrapper file as MISC_FILE instead of VERILOG_FILE or VHDL_FILE	Until Quartus fixes this issue, you must manually modify the `qip` file and change the tag to VERILOG_FILE or VHDL_FILE.
Clock constraints like create_generated_clock, that are applied to ALTPLL instantiations in IP megafunctions, are translated to timing_unapplied.sdc	You must modify these constraints with the proper clock object names. You can find the correct names in the constraints checker results or in the `srr` logfile.
Quartus projects with Megacore IPs have IP-related files added directly to the project	Do not add the IP files directly to the Quartus project. Use the flow recommended by Altera and set up the project to include a `qip` file that contains the pointers to the IP files.
Quartus designs with Megacore IPs do not have a timing and resource estimation netlist	When you create the IP with the Megawizard tool, generate the timing and resource estimation netlist. The option to generate the netlist on the EDA tab of the Megawizard.
VHDL file order might cause compiler errors	You might see compiler errors if the architectures file is compiled before the entities, If this happens, edit the Altera `qsf` file to ensure that the VHDL entities are compiled before the architecture files.
VHDL compiler fails because of missing `qsf` file	Edit the Altera `qsf` file and ensure that all library statements are included in this file.

Copyright © 2013 Synopsys, Inc.

Working with Lattice IP

The Lattice IP encryption scheme uses the flow described in *Encryption and Decryption*, on page 971. The following procedure details how to incorporate Lattice encrypted IP in your design.

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

   The Lattice IP is encrypted using the persistent_key method described in *Specifying the Script Output Method*, on page 979.

3. Synthesize your design.

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

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

   See *Encryption and Decryption*, on page 971 for a flow diagram of this process.

Copyright © 2013 Synopsys, Inc.

Importing Vivado IP

Xilinx

If you have Xilinx IP generated from the Vivado IP catalog, you can directly import it into your synthesis design. See the following for details:

- Importing Vivado IP for Synthesis, next
- Debugging Vivado IP Importation, on page 1044

Importing Vivado IP for Synthesis

The following procedure describes how to import Vivado IP into a synthesis project using the import_vivado_ip command. For the command syntax and examples, see *import_vivado_ip Command*, on page 1018 in the *Reference Manual*.

1. Generate IP from the Vivado IP catalog.

   After it has been generated, the IP always has an xml file in the main IP directory. It can also have an xci file, or an xco file, or both.

2. If the generated IP is to be the top-level design, do the following:
   - Open the synthesis tool, but do not load a project.
   - Run the following command from the Tcl command line in the synthesis tool:

     import_vivado_ip *ip_dir* **|-xml_file** *file*.**xml | -xci_file** *file*.**xci | -xco_file** *file*.**xco**

   The command creates a new project and loads the IP files from the specified source into this directory. If there are XDC constraints defined for the IP, it translates them to FDC, and adds the fdc file to the synthesis project. It marks the original XDC files for place-and-route only, and they are not used for synthesis. The command reports the results in the Tcl window.

3. If the generated IP is to be instantiated in a bigger design, do the following:
   - Open the parent project in the synthesis tool.

Copyright © 2013 Synopsys, Inc.

– Run the following command from the Tcl command line in the synthesis tool:

import_vivado_ip [-flat | -hier]
ip_dir **|-xml_file** *file*.**xml | -xci_file** *file*.**xci | -xco_file** *file*.**xco**

If you specify -hier, the command creates a subproject for the IP and loads the IP files from the specified source into the subproject directory. If you specify -flat, the command adds all the IP files directly into the parent project.

If there are XDC constraints defined for the IP, it translates them to FDC, and adds the fdc file to the synthesis project. It marks the original XDC files for place-and-route only, and they are not used for synthesis.

The command reports the import results in the Tcl window and the log file.

4. Check the results.
   – Check the report for warnings and make any changes needed to fix the problem.
   – Check the generated project to make sure it is as you expected.
5. Synthesize the design as usual.

Debugging Vivado IP Importation

The following describe situations you can run into when you import Vivado IP:

- Parsing errors reading IP metadata files

  There are two reasons why the import_vivado_ip command might report an error when reading the xml, xco, or xci files:

  – The file is corrupted. In this case, try regenerating the IP and import it again.
  – The command is not parsing the file correctly. Report the issue to Synopsys support, and include the file with the issue.

- Project loading error after import

  If you have already loaded a project or subproject with the same name as the imported project, the import_vivado_ip command generates an error. You must close the project and re-import the IP.

- Other import errors
  - You can get errors if the HDL and constraint files are not available in the IP. This may happen if IP was not generated correctly from the Vivado IP catalog.
  - You get errors when you import IP into a directory that is not writable.

Copyright © 2013 Synopsys, Inc.

Working with Xilinx IP Cores

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

- Xilinx Cores, on page 1046
- Secure and Non-secure Cores, on page 1047

For information about including EDK cores or IP produced with encryptIP, see *Converting Xilinx Projects with ise2syn*, on page 1051.

Xilinx Cores

The following table describes how the tool handles different types of Xilinx IP cores:

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

Copyright © 2013 Synopsys, Inc.

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

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

Secure and Non-secure Cores

This section describes secure and non-secure IP cores and how the tools handle them during synthesis.

Add the `edn`, `ngo`, and/or `ngc` core files to your project. The synthesis tools can read these formats and incorporate the cores into the design.

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

Copyright © 2013 Synopsys, Inc.

After you synthesize the design, the synthesis process treats the cores as follows:

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

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

EDN NGC, non-secured NGO	• The tool generates one main output netlist that includes all the unencrypted cores. The log file resource usage report includes the resources used by the cores. • All timing constraints are forward-annotated in the synplicity.ucf file. This file includes imported UCF constraints (see *Converting and Using Xilinx UCF Constraints*, on page 200) as well as user-specified timing constraints. • Placement constraints are forward-annotated in the *design*.ncf file.
NGC, secured	• The tool writes out a top-level EDIF file which references individual EDIFs for each instantiation of a secure core. These files are not included in the main netlist.The tool suffixes the original core name with _syn when it names the lower-level files. The log file report of resource usage includes the resources used by the cores. • The synthesis tool puts all timing constraints into one synplicity.ucf file for the P&R tool. This file includes imported UCF constraints (see *Converting and Using Xilinx UCF Constraints*, on page 200) as well as user-specified timing constraints. • Placement constraints, excluding constraints for secure cores, are forward-annotated in a *design*.ncf file. • For each secure core, the tool generates an individual `ncf` file with the constraints for that core.

Limitations

Module names can be changed in the final netlist that is output from logic/physical synthesis, so names may differ from the module names used in your UCF for place and route; this produces an error in the place-and-route tool.

As a workaround, manually update the module names in the UCF.

- Open the technology view (`srm` file).
- Select the module and hit Ctrl-c to copy the module name as it appears in the netlist.

In the UCF, paste the module name (replace the old module name with the updated module name).

Including Xilinx Cores for Logic and Physical Synthesis

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

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

1. Instantiate the Xilinx core in your top-level RTL.
2. Make sure you are targeting a technology that supports this design flow.
3. Add the `edn`, `ngo`, and/or `ngc` core files directly to your project. The synthesis tools can read these formats and incorporate the cores into the design. For more information about theses cores, see *Secure and Non-secure Cores*, on page 1047.
4. Set the syn_macro attribute to determine how you want the cores to be treated during synthesis. For details about this attribute, see *syn_macro*, on page 1392.
5. Synthesize the design.
   - Optionally, set the option to automatically run P&R from the synthesis tool interface after synthesis is complete. If you chose to

Copyright © 2013 Synopsys, Inc.

run P&R from the synthesis tool, the output netlist and constraint files are automatically copied to the P&R directory.

– Set any other options.

– Click the Run button to run synthesis.

The synthesis tools read the timing and resource usage information from the core files. The tool runs synthesis and places the design at the same time. For more information about how synthesis treats these cores and the output core files generated for P&R, see *Secure and Non-secure Cores*, on page 1047.

Converting Xilinx Projects with ise2syn

The ise2syn utility (Import->Import ISE/EDK Project) takes a Xilinx project and converts it to a synthesizable project which you can run in the Synplify Pro or Synplify Premier tools. The ise2syn utility uses ucf2sdc functionality to convert constraints when it translates the Xilinx project. The ucf2sdc utility converts a Synplify Pro logic synthesis project into a project for physical synthesis with the Synplify Premier tool.

The following describe the ise2syn utility and its use in more detail:

- Converting Designs with the ise2syn Utility, on page 1051
- The ise2syn Conversion Process, on page 1056
- Specifying EDK Cores as White Boxes, on page 1059

If you want to convert UCF constraints from a logic synthesis project (Synplify or Synplify Pro) into a physical synthesis project, do not use ise2syn. Instead use ucf2sdc, which is described in *Converting and Using Xilinx UCF Constraints*, on page 200.

Converting Designs with the ise2syn Utility

You can use the ise2syn utility to automatically generate a synthesis project from a Xilinx project. You can use this utility to convert the following Xilinx projects:

- ISE *.ise, *.xise, and *.xmp projects. See *Converting ISE Designs with the ise2syn Utility*, on page 1052.
- EDK standalone projects (*.xmp) or sub-projects used in an ISE project, including EDK white box cores. See *Converting EDK Designs with the ise2syn Utility*, on page 1054.
- Core generator modules that are sub-blocks in an ISE project

The utility translates constraints from `ucf` and `ncf` files. The utility applies to ISE projects created with ISE 11.1 or later.

Copyright © 2013 Synopsys, Inc.

The following figure summarizes the design flow for converting ISE projects. See *Converting ISE Designs with the ise2syn Utility*, on page 1052 and *Converting EDK Designs with the ise2syn Utility*, on page 1054 for detailed procedures.

Converting ISE Designs with the ise2syn Utility

The following procedure describes how to convert ISE designs with the ise2syn utility.

1. Make sure the ISE project was created with ISE 11.1 or later. If you used an earlier version of ISE, update it version 11.1 or later before proceeding.

2. Set the XILINX environment variable to the ISE installation used for running the ISE project.

For example: `XILINX=\\myServer\pctools\Xilinx\ise112\ISE`

3. Check that you have write permission to the synthesis project directory for conversion.

4. Import the ISE project into the Synplify Pro/Synplify Premier tool.

   The following describes how to use the GUI to import the project. For the syntax of the equivalent Tcl command, see *Xilinx ise2syn Utility*, on page 1017 in the *Reference Manual*.

   – Select the Import-> Import Xilinx ISE/EDK Project command. The Import Xilinx Project dialog box opens. You can import .ise, .xise, or .xmp projects.

   – Set other options to import the project, and click Import.

Copyright © 2013 Synopsys, Inc.

After it successfully completes, the ise2syn utility generates the following files. See *Conversion Process*, on page 1057 for details of the process.

design.prj	Synplify Pro/Premier project file for FPGA synthesis
*design*_sdc/*filename*_conv.sdc	Converted `ucf`, `ncf`, or `xcf` constraints for synthesis
*design*_sdc/*filename*_unapplied.ucf	File that contains unconverted constraints. See *Constraint Translation*, on page 1058 for details.

5. Use the converted `prj` and `sdc` files to run synthesis.
6. If you have an _unapplied.ucf file with unconverted constraints, check this file and manually convert these constraints for input to P&R.
7. Run ISE P&R using the files generated after synthesis.

Converting EDK Designs with the ise2syn Utility

The following procedure describes how to convert EDK designs with the ise2syn utility. For additional information about working with EDK cores, see *Specifying EDK Cores as White Boxes*, on page 1059.

1. Make sure the ISE project was created with ISE 11.1 or later. If you used an earlier version of ISE, update it version 11.1 or later before proceeding.
2. Set the following environment variables:

XILINX	Set to the ISE installation used for running the ISE project For example: `XILINX=\\myServer\pctools\Xilinx\ise112\ISE`
XILINX_EDK	Set to the EDK installation used for running the EDK project. For example: `XILINX_EDK=\\myServer\pctools\Xilinx\ise112\EDK`
XIL_MYPERIPHERALS	If required, set to the location of user-specific cores. For example: `XIL_MYPERIPHERALS=e\my_edk_cores`

3. Check that you have write permission to the synthesis project directory for conversion.

4. For an EDK project, first synthesize the project using the ISE tool.

   When you use ISE to run XST synthesis, the tool generates ngc files for the EDK cores. The FPGA synthesis tools use these ngc files instead of the HDL files for the EDK projects. The tool also uses the ngc for encrypted cores, because the ise2syn translator does not read encrypted cores.

5. Import the ISE project into the Synplify Pro/Synplify Premier tool. The following describes how to use the GUI to import the project. For the syntax of the equivalent Tcl command, see *Xilinx ise2syn Utility*, on page 1017 in the *Reference Manual*.

   – Select Import-> Import Xilinx ISE/EDK Project. The Import Xilinx Project dialog box opens. You can import .ise, .xise, or .xmp projects.

Copyright © 2013 Synopsys, Inc.

– If you have an encrypted core or a core that uses non-standard HDL, enable the Force White Box Cores option and specify the names of the cores. See *Force White Box Cores Option (EDK)*, on page 227 in the *Reference Manual* for details of its use.

  If you do not do so, the tool errors out because it cannot read the encrypted core or non-standard HDL. If the synthesis compiler does error out while translating the project, enable the Force White Box Cores option and specify the core names. When you enable this option, the synthesis tool uses `ngc` files instead of the standard HDL files for the specified cores.

– Set other options to import the project, and click Import.

After it successfully completes, the ise2syn utility generates the following files. See *Conversion Process*, on page 1057 for details.

design.prj	Synplify Pro/Premier project file for FPGA synthesis
*design*_sdc/*filename*_conv.sdc	Converted `ucf`, `ncf`, or `xcf` constraints for synthesis
*design*_sdc/*filename*_unapplied.ucf	Unconverted constraints file. See *Constraint Translation*, on page 1058 for details.

6. Use the converted `prj` and `sdc` files to run synthesis.
7. If you have an _unapplied.ucf file with unconverted constraints, check this file and manually convert these constraints for input to P&R.
8. Run ISE P&R using the files generated after synthesis.

The ise2syn Conversion Process

The following describe the ise2syn conversion process:

- Conversion Process, on page 1057
- Troubleshooting ise2syn Conversion Issues, on page 1057
- Constraint Translation, on page 1058

Conversion Process

The ise2syn utility goes through the following phases when it converts the ISE or EDK project to a Synplify Pro/Premier project ready for synthesis.

- The utility translates ISE project options and EDK or Coregen sub-projects, and generates a Synplify Pro/Premier project file (`prj`).
- It compiles the project to get a netlist (`srs`) file.
- For EDK projects, it then updates the memory block names to use the full hierarchical path names.
- It translates the `ucf`, `ncf`, or `xcf` constraints from the Xilinx project to the `sdc` format. It writes all untranslated constraints to the *_unapplied.ucf file.
- It then updates the project file.

Troubleshooting ise2syn Conversion Issues

The following table lists possible reasons for errors that occur at different stages of the translation flow.

Error Occurs When...	Possible Causes
Importing the ISE project	• The XILINX/XILINX_EDK environment variable does not point to a valid ISE/EDK installation. • The XILINX_EDK environment variable points to an EDK version that is not the same as the one with which the imported project was created • You do not have write permission to the specified Synplify project directory.
Compiling the imported project	For EDK projects with cores that use non-standard HDLs, specify those cores as white boxes when you import the project.
Converting `bmm` files	No errors are expected. If you see an error at this stage, report it.

Copyright © 2013 Synopsys, Inc.

Error Occurs When...	Possible Causes
Translating constraints	Currently, there is limited support for translating constraints, as detailed in *Constraint Translation*, on page 1058. All unsupported constraints are written to the *_unapplied.ucf file. Check this file and convert the constraints manually for P&R. No errors are expected at this stage. If you see an error, report it.
Synthesizing the converted project	No errors are expected. If you see an error at this stage, report it.
Running P&R on the converted project	This could be caused by untranslated constraints. Check the *_unapplied.ucf file and convert the constraints manually.

Constraint Translation

The ise2syn utility uses ucf2sdc functionality to translate constraints; this has some limitations when it translates Xilinx constraints.

- The synthesis tools do not currently translate the following UCF constraints. All untranslated Xilinx constraints are written to the *_unapplied.ucf file.
  - `OPEN/CLOSED` in `AREAGROUP` constraints
  - `OFFSET` constraints that use the `VALID` keyword
  - `OFFSET` constraints that use the `HIGH/LOW` clock edges
- If the generated names of objects like nets and instances do not exactly match the XST-generated names, the tool does not convert the constraints associated with these objects. It puts these untranslated constraints in the _unapplied.ucf file. Make sure to check this file and manually edit constraints as needed.
- The tools do not translate constraints applied on inferred objects in an XST-generated netlist.
- Support for constraints applied on a core that uses `ncf` files is not tested.

For additional information about the translation of constraints, refer to *Xilinx ucf2sdc Utility*, on page 1004 in the *Reference Manual*. The ise2syn conversion process uses the ucf2sdc utility to translate constraints.

Specifying EDK Cores as White Boxes

It is very important that you accurately define EDK cores as white boxes because the synthesis run will fail if the design is not correctly specified, and you will have to iterate through another run. For the ise2syn flow, a white box is one that has a core definition in either `edn` or `ngc` format.

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

2. If you want to treat a core as a white box during synthesis, specify the core name in the Force White Box option. For details about settings for this option, refer to *Force White Box Cores Option (EDK)*, on page 227 in the *Reference Manual*.

   The software searches for the core in the mhs file, and for a definition file named implementation/*core*.ngc. It adds this file to the project. The core is treated as a white box during synthesis. If it does not find the named core or the `ngc` file, it issues a warning message.

Copyright © 2013 Synopsys, Inc.

CHAPTER 19

Fast Synthesis

The following describe how to use the Fast Synthesis feature in Synplify Premier:

- About Fast Synthesis, on page 1062
- Using Fast Synthesis, on page 1063

Copyright © 2013 Synopsys, Inc.

About Fast Synthesis

Fast synthesis is a feature available in the Synplify Premier software. It is a logic synthesis design flow that is specific to Synplify Premier, like enhanced optimization.

What fast synthesis does is to significantly reduce synthesis runtimes by a factor of 2 or 3. It accomplishes this by reducing the number of optimizations performed, so there is a trade-off in performance.

You can use fast synthesis with designs targeting Altera Stratix and Xilinx Virtex and Spartan device families.

When to Use Fast Synthesis

Fast synthesis is best used in situations where quality of results (QoR) is not crucial, or quick turnaround times are more important. Do not use this flow if performance is critical. The following list some situations where fast synthsis is effective:

- In the initial design development phase, when you need to quickly evaluate a design or get a baseline result, and performance is secondary.
- When exploring "what if" scenarios when you have different implementations in mind for your design. In such a case, fast synthesis could save you time working through different runs.
- When you need a quick preliminary synthesis result to help get post-synthesis feedback.
- When prototyping a design (Altera Stratix and Xilinx Virtex and Spartan families only). This can speed up the process for ASIC prototype designers who are developing initial board-level implementations to verify the design.
- When you need to have quick RTL-to-board turnaround times for debug iterations.

Copyright © 2013 Synopsys, Inc.

Using Fast Synthesis

This section describes how to run Synplify Premier fast synthesis.

The Fast Synthesis Design Flow

The following figure summarizes the steps in the fast synthesis design flow. The steps are described in *Running Logic Synthesis with Fast Synthesis*, on page 1063.

Running Logic Synthesis with Fast Synthesis

Use the following procedure for quick evaluations or where a faster runtime is more important than the quality of results.

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

Copyright © 2013 Synopsys, Inc.

In general, timing attributes are not honored in Fast Synthesis mode.

4. Set options.
   - Set any options that you want in the Implementation Options dialog box.
   - Make sure that the Auto Constrain option is disabled.
5. Specify logic synthesis with fast synthesis.
   - Disable the Physical Synthesis option, either in the Project window or in the Implementation Options dialog box. This ensures that you are only running logic synthesis. You cannot run fast synthesis if the Physical Synthesis option is enabled.
   - In the Device panel of the Implementation Options dialog box, set Target to one of the supported Altera or Xilinx families.
   - Enable Fast Synthesis either in the Project view or the Options panel of the Implementation Options dialog box. This option is off by default, and you must explicitly enable it.

Project View

Implementation Options->Options

The tool reduces the amount and number of logic synthesis optimizations performed which results in faster runtimes. If you have both Fast Synthesis and Enhanced Optimization enabled (see *Logic Synthesis with Enhanced Optimization*, on page 40), the software ignores the Enhanced Optimization setting and runs fast synthesis.

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

   After you have analyzed the results of this preliminary run, you can do another fast synthesis run, or disable the Fast Synthesis option and repeat synthesis with a full-scale logic or physical synthesis run.

Copyright © 2013 Synopsys, Inc.

If Fast Synthesis is intended for quick synthesis results and not for a fast board implementation, it is recommended that you do not run P&R on the resulting netlist, as you might get sub-optimal QOR and longer P&R runtimes.

Attributes Limitation

The Fast Synthesis flow does not support the following attributes:

- syn_allow_retiming
- syn_maxfan
- syn_pipeline
- syn_replicate

Copyright © 2013 Synopsys, Inc.

Fast Synthesis and Other Synthesis Options

When you run fast synthesis, other optimizations can be affected.

Option	Usage with Fast Synthesis
Auto Constrain	Do not use this option with fast synthesis.
Enhanced Optimization	Do not set this option with fast synthesis, as it will be ignored.
FSM Explorer	You can use this option with fast synthesis.
Physical synthesis	You cannot run fast synthesis with physical synthesis enabled. Fast synthesis only operates in logic synthesis mode in the Synplify Premier tool. Fast synthesis does not generate any placement information.
Pipelining	You can use this option with fast synthesis.
Resource Sharing	You can use this option with fast synthesis.
Retiming	If you enable Fast Synthesis, the tool does not do any retiming optimizations.
Timing constraints	You can set any design constraints as you would normally do. However, if your goal is to shorten runtimes for a fast board implementation for example, it is recommended that you either use loose timing constraints or set the global clock to 1 Mhz.

Copyright © 2013 Synopsys, Inc.

CHAPTER 20

Activity Analysis for Power Optimization

The Synplify Premier Activity Analysis feature can be used for designs targeting Xilinx technologies. This feature captures switching activity data for analyzing and ultimately optimizing design power. Using the Activity Analysis feature, after an initial synthesis run, the design is resynthesized using the netlist and the analysis design constraints (adc) file. This generates the activity report or SAIF file. The file contains logic state and switching transition data for each net in the design. This SAIF file can be read into third-party tools (such as Xilinx ISE) for power-aware routing.

Topics include:

- Activity Analysis Flow
- About the SAIF File
- Specifying Activity Analysis Constraints
- Specifying SAIF Parameters
- Technology View (project_saif.srm)
- Activity Analysis for Power Improvement Flow

Copyright © 2013 Synopsys, Inc.

Activity Analysis Flow

Here is an overview of how to obtain the report:

Activity Analysis Generation

Assign Activity Analysis Constraints

After you synthesize the design, you can assign analysis design constraints. This task consists of assigning the syn_state1_prob and/or syn_trans_prob attributes. You only need to create these constraints if you want to override the tool's defaults for state or switching probabilities. See *syn_state1_prob* and *syn_trans_prob* for default information.

To assign constraints:

1. Define constraints in an analysis design constraint file (`adc`). See *Specifying Activity Analysis Constraints*, on page 1073 for details.

2. Then, add the `adc` files to your project.

Generate the Report (SAIF File)

1. From the GUI, select Analysis->Activity Analysis Generation.

2. Enter the parameters for the report:

   – SAIF File – enter the report file name in the field.

   – Constraint Files – Select the activity design constraint files (`adc`) that you want to include in the analysis data. The `adc` file contains constraints defined by the syn_state1_prob and syn_trans_prob attributes. See *Specifying Activity Analysis Constraints*, on page 1073 for information on specifying these constraints.

   For an overall description of the activity analysis parameters, see *Specifying SAIF Parameters*, on page 1077.

3. Click Generate.

   This generates the `saif` file.

Copyright © 2013 Synopsys, Inc.

4. After the `saif` file is generated, you can optionally:
   - Estimate the power consumption for the design using a third-party power analysis tool and the `saif` file as input. If you are targeting Xilinx technologies, see *Activity Analysis for Power Improvement Flow*, on page 1079 for information.
   - Run place and route for power driven-optimization using the `saif` file as input. If you are targeting Xilinx technologies, see *Xilinx Place and Route using SAIF*, on page 1084 for information.

About the SAIF File

The Synopsys switching activity interchange format file (`saif`) is an ASCII file that stores logic state and transition data for the nets in the design. This information can be used for power optimization and analysis.

Following is a sample report where:

- TC = Toggle count, number of transitions on this net
- T1 = Logic 1 time, simulation time at logic high state
- T0 = Logic 0 time, simulation time at logic low state
- TX = Logic X time, simulation time at logic unknown state
- IG = Number of inertial glitch edges

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN)
(DATE "2009 1 30 14 49 50")
(VENDOR "SYNP")
(PROGRAM_NAME)
(VERSION "1.0")
(DIVIDER /)
(TIMESCALE 1 ns)
(DURATION 8000000000.00)
(INSTANCE tb
  (INSTANCE top_inst
   (NET
     (A
      (T0 4000000000) (T1 4000000000) (TX 0)
      (TC 2) (IG 0)
     )

        (B
      (T0 8000000000) (T1 0) (TX 0)
      (TC 0) (IG 0)
     )

        (C
      (T0 2000000000) (T1 6000000000) (TX 0)
      (TC 3) (IG 0)
     )
```


Copyright © 2013 Synopsys, Inc.

```
            (G
         (T0 8000000000) (T1 0) (TX 0)
         (TC 0) (IG 0)
         )

            (D
         (T0 8000000000) (T1 0) (TX 0)
         (TC 0) (IG 0)
         )

            (...

        (INSTANCE exapme_inst
     (NET
       (d
        (T0 8000000000) (T1 0) (TX 0)
        (TC 0) (IG 0)
        )

       (a
        (T0 4000000000) (T1 4000000000) (TX 0)
        (TC 2) (IG 0)
        )

       (b
        (T0 8000000000) (T1 0) (TX 0)
        (TC 0) (IG 0)
        )

...
       )
     )
```

Specifying Activity Analysis Constraints

You can override the defaults for logic state and state transitioning using the syn_state1_prob and syn_trans_prob attributes, respectively, which are the analysis design constraints (adc) for activity analysis reporting. Specify adc constraints only if you want to override the default probabilities for logic state high (default is 0.5) and logic state transitioning (default is 0.2). You can set these attributes on input ports. (See *syn_state1_prob*, on page 1074 and *syn_trans_prob*, on page 1075 for more details.)

To define analysis design constraints:

1. Select File->New.

   – Click on Analysis Design Constraints.
   – Enter the file name.
   – Specify the path name if you want to change the default location.
   – Click OK.

   This opens the SCOPE editor.

Copyright © 2013 Synopsys, Inc.

2. Click on the Attributes tab and specify the probability values.

   Set the constraints on input ports. See *syn_state1_prob* and *syn_trans_prob* below for more details.

syn_state1_prob

Overrides the default probability for specified input ports to remain at logic state 1 for the duration period. (A duration period default is 1 second.) This attribute assigns the probability for logic state 1. The default value is 0.5 (or 50 percent)—50 percent of the time the signal is high, and the other 50 percent of the time, the signal is low. Use this attribute only if you want to override the default value of 0.5 for a high signal; value must greater than or equal to 0, and less than or equal to 1.

.adc File Syntax and Examples

define_attribute syn_state1_prob {p:*port*} {*probability_value*}

The syntax above sets the logic state probability for *specified input ports*. For example:

```
define_attribute syn_state1_prob {p:x[7:0]} {0.8}
```

See Also

- syn_trans_prob, below.
- *Clock and Reset*, on page 1075.
- *Assign Activity Analysis Constraints*, on page 1068.

syn_trans_prob

Overrides the default probability for logic state transition. The default transition probability is 0.2 (or 20 percent)—20 percent of the time, the signal switches from high to low or from low to high. (The default transition period is 1 second.) Use this attribute only if you want to override the default value of 0.2; value must greater than or equal to 0, and less than or equal to 1.

.adc File Syntax and Examples

define_attribute syn_trans_prob {p:*port*} {*probability_value*}

The syntax above sets the state transition probability for *specified input ports*. For example:

```
define_attribute syn_trans_prob {p:x[7:0]} {0.1}
```

See Also

- syn_state1_prob, above.
- *Clock and Reset*, on page 1075.
- *Assign Activity Analysis Constraints*, on page 1068.

State and Transition Probability Relationship

The relationship between state and transition probabilities is as follows:

```
t is less than or equal to 2 * min (s, 1-s)
```

For example, if syn_state1_prob is 0.7, then syn_trans_prob cannot be larger than 0.6.

Clock and Reset

Clock

For multiple clock domains (clock groups), by default the software uses the fastest clock on the port as the reference clock. For the reference clock: state=1, transition=2.

Copyright © 2013 Synopsys, Inc.

Reset

By default the software detects any input port reset and assigns the following defaults: state=0.001 transition=0. You can override these defaults by specifying the syn_state1_prob or syn_trans_prob in the `adc` file. (See *Specifying Activity Analysis Constraints*, on page 1073.)

Specifying SAIF Parameters

The Analysis->Activity Analysis Generation command generates the power activity analysis report for Xilinx technologies.

The table below shows the options to specify in the Activity Analysis Generation dialog box:

Command	Description
Output Files	
• SAIF File	Contains the activity analysis report. Default file name is *project*.saif. See *About the SAIF File*, on page 1071 for information on the data in this report.
Constraint Files	Select the analysis design constraint files (adc) to use for the report. See *Specifying Activity Analysis Constraints*, on page 1073 for information on specifying analysis constraints.

Copyright © 2013 Synopsys, Inc.

Technology View (*project*_saif.srm)

When you generate an activity analysis report using the Annotated SRM File switch, you can display a Technology view that provides information on the net switching and transitioning data.

The data includes information for transitioning (pwr_trans_formal) and logic state (pwr_state1_formal) values from the simulation tool. It also includes a .pwr_rename value.

Activity Analysis for Power Improvement Flow

You can use the SAIF file from the Synplify Premier tool in Xilinx ISE power analysis using (version 10.1 or later).

Topics include:

- Power Improvement Flow
- Xilinx Power Analysis
- XPower Report
- Xilinx Place and Route using SAIF
- Xilinx Place and Route from Synplify Premier

Power Improvement Flow

SAIF for Power Improvement

Copyright © 2013 Synopsys, Inc.

Here is the flow:

1. Synthesize the design.
2. Generate the `saif` file. See *Assign Activity Analysis Constraints*, on page 1068.
3. Input the `saif` file into Xilinx ISE to obtain an XPower report. See *Xilinx Power Analysis*, on page 1080.
4. Use the `saif` file in the Xilinx ISE place and route tool to enable power-aware placement and routing.
5. Then, recalculate the power using XPower analysis to determine how much improvement is gained by the SAIF and Xilinx ISE power-aware placement and routing.
6. Compare the results in Step 5 to the one obtained in Step 3 above.

Xilinx Power Analysis

To use the `saif` file with Power Analysis in ISE:

1. Right-click on XPower Analyzer, and select Properties.

2. Enter the path name and `saif` file in the Load Simulation File field.

3. Click Run.

   This generates the power report. See *XPower Report*, on page 1081, for an example.

XPower Report

This following figures include sections of a sample Xilinx XPower report.

Copyright © 2013 Synopsys, Inc.

```
-----------------------------------------------------------------
Release 10.1.03 - XPower Analysis Report SoftwareVersion:K.39 (nt)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

# NOTE:  This file is designed to facilitate import into a spreadsheet program
# such as Microsoft Excel for viewing, printing and sorting.  The "|" character
# should be selected as the data field separator.


Design       | C:\testing\hw_loop_open_source.v5\index_inc\index_inc.ncd       |
Preferences  | C:/testing/hw_loop_open_source.v5/index_inc/index_inc.pcf       |
Part         | xc5vlx50ff676-1                                                 |
Data version | PRODUCTION,v1.2,08-28-08                                        |
```

...

```
Power summary                          |    I(mA)   |    P(mW)   |
-------------------------------------------------------------------
Total estimated power consumption      |            |       477  |
                                     ---
                Total Vccint   1.00V   |       277  |       277  |
                Total Vccaux   2.50V   |        58  |       145  |
                Total Vcco25   2.50V   |        22  |        55  |
                                     ---
                           Clocks      |         5  |         5  |
                           Inputs      |         0  |         0  |
                            Logic      |         0  |         0  |
                          Outputs      |
                             Vcco25    |        21  |        52  |
                          Signals      |         0  |         0  |
                                     ---
```

...

```
Thermal summary
-------------------------------------------------------------------
Estimated junction temperature        |            31C            |
             @ airflow of 250 LFM     |            29C            |
                          500 LFM     |            28C            |
                          750 LFM     |            28C            |
                       Ambient temp   |            25C            |
                          Case temp   |            31C            |
                          Theta J-A   |           12C/W           |
                                    ---
Max ambient at junction max of        |        85C  |  79C        |
                          250 LFM     |            81C            |
                          500 LFM     |            82C            |
```

continued...

Power Details:

Outputs

Power details

Outputs:1	Loads	Loading(fF)	C(pF)	F(MHz)	I(mA)	P(mW)
Vcco25						
index_out(0)		5000	19	80.2	13.1	13.1
index_plus_one(0)		5000	19	80.2	13.1	13.1
index_out(1)		5000	19	40.1	6.6	6.6
index_plus_one(1)		5000	19	40.1	6.6	6.6
index_out(2)		5000	19	20.1	3.3	3.3

Clocks

Clocks:1	Loads	Loading(fF)	C(pF)	F(MHz)	I(mA)	P(mW)
clk_ibuf_iso						
Logic:						
clk_ibuf			6	320.5	4.6	4.6

Logic

Logic:	Loads	Loading(fF)	C(pF)	F(MHz)	I(mA)	P(mW)
U_reg_dw/q_RNO[0]			361	80.2	0.0	0.0
index_out_c(0)_rt			361	80.2	0.0	0.0
U_reg_dw/q_rep0_i_se[0]			361	78.7	0.0	0.0
U_reg_dw/q[0]			313	80.2	0.0	0.0
U_reg_dw/q_rep0_i[0]			313	80.2	0.0	0.0
U_adder/U_mux_s_up/mout_cry_3			0	40.1	0.0	0.0

Inputs

Inputs:	Loads	Loading(fF)	C(pF)	F(MHz)	I(mA)	P(mW)
clk			9976	320.5	0.0	0.0
inc_en			9976	80.0	0.0	0.0
reset			9976	33.2	0.0	0.0
Signals:	**Loads**	**Loading(fF)**	**C(pF)**	**F(MHz)**	**I(mA)**	**P(mW)**
U_adder/U_mux_s_up/mout_cry_3/O	1		0	40.1	0.0	~0.0
U_reg_dw.q_rep0_i(0)	2		0	80.2	0.0	~0.0
inc_en_c	4		0	80.0	0.0	~0.0
index_out_c(0)	4		0	80.2	0.0	~0.0
index_out_c(1)	2		0	40.1	0.0	~0.0

Power Improvement Guide

Power improvement guide:

Signal	Power when asserted(mW)	Power when disabled(mW)	Power savings at 50%(mW)
inc_en_c	476.7	476.7	-0.0

See the Xilinx documentation for the details on this report.

Copyright © 2013 Synopsys, Inc.

Xilinx Place and Route using SAIF

You can use the `saif` file in Xilinx ISE place and route tool using (version 10.1 or later). To use the `saif` file with ISE place and route:

1. Right-click on Place and Route, and select Properties.

2. Select the Power Reduction option.
3. Enter the path name and `saif` file in the Power Activity File field.
4. Click Run.

   This generates the power report. See *XPower Report*, on page 1081, for an example.

Xilinx Place and Route from Synplify Premier

In the Synplify Premier tool, you can optionally add the -power and -activityfile options to your Xilinx options file (xf.opt) and run place and route from the synthesis environment. The following example shows where to add these options to the file.

```
FLOWTYPE = FPGA;
#######################
## lib/xilinx/xf.opt ##
## Test file created ##
#######################

       ...

###########################
## Place & Route Options ##
###########################
## Type "par -h" for a detailed list of par command line options
###########################

Program par
-w;                      # Overwrite existing placed and routed
ncd
-power on;
-activityfile X:\testing\s2p.v5\results_2\s2p.saif;
-intstyle xflow;         # Message Reporting Style: ise, xflow,
or silent
-ol high;                 # Overall effort level
-t 1;                     # Placer cost table entry.
#-xe c;
<design>_map.ncd;         # Input mapped NCD file
<inputdir><design>.ncd;   # Output placed and routed NCD
<inputdir><design>.pcf;   # Input physical constraints file
END Program par

   ...
```

This generates the power report. See *XPower Report*, on page 1081, for an example of this report.

Copyright © 2013 Synopsys, Inc.

CHAPTER 21

Working with Compile Points

The following sections describe compile points and how to use them in logic synthesis iterative flows:

- Compile Point Basics, on page 1088
- Compile Point Synthesis Basics, on page 1098
- Synthesizing Compile Points, on page 1109
- Using Compile Points with Other Features, on page 1124
- Resynthesizing Incrementally, on page 1126

Copyright © 2013 Synopsys, Inc.

Compile Point Basics

Compile points are RTL partitions of the design that you define before synthesizing the design. Compile points can be defined manually, or the tool can generate them automatically. The software treats each compile point as a block, and can synthesize, optimize, place, and route the compile points independently. Compile points can be nested.

See the following topics for some details about compile points:

- Advantages of Compile Point Design, on page 1088
- Automatic and Manual Compile Points, on page 1090
- Nested Compile Points, on page 1091
- Compile Point Types, on page 1092

Advantages of Compile Point Design

Designing with compile points makes it more efficient to work with the increasingly larger designs of today and the corresponding team approach to design. They offer several advantages, which are described here:

- Compile Points and Design Flows, next
- Runtime Savings, on page 1089
- Design Preservation, on page 1089

Compile Points and Design Flows

Compile points improve the efficacy of both top-down and bottom-up design flows:

- In a traditional bottom-up design flow, compile points make it possible to easily divide up the design effort between designers or design teams. The compile points can be worked on separately and individually. The compile point synthesis flow eliminates the need to maintain the complex error-prone scripts for stitching, modeling, and ordering required by the traditional bottom-up design flow.
- From a top-down design flow perspective, compile points make it easier to work on the top-level design. You can mark compile points that are still being developed as black boxes, and synthesize the top level with

Copyright © 2013 Synopsys, Inc.

what you have. You can also customize the compile point type settings for individual compile points to take advantage of cross-boundary optimizations.

You can also synthesize incrementally, because the tool does not resynthesize compile points that are unchanged when you resynthesize the design. This saves runtime and also preserves parts of the design that are done while the rest of the design is completed.

See *Compile Point Synthesis*, on page 1104 for a description of the synthesis process with compile points.

Runtime Savings

Compile points are the required foundation for multiprocessing and incremental synthesis, both of which translate directly to runtime savings:

- Multiprocessing runs synthesis as multiple parallel processes, using the compile points as the partitions that are synthesized in parallel on different processors. See *Combining Compile Points with Multiprocessing*, on page 1125.
- Incremental synthesis uses compile points to determine which portions of the design to resynthesize, only resynthesizing the compile points that have been modified. See *Resynthesizing Compile Points Incrementally*, on page 1126.

In addition, you can combine all of them with the Synplify Premier Fast Synthesis mode for even more savings.

Design Preservation

Using compile points addresses the need to maintain the overall stability of a design while portions of the design evolve. When you use compile points to partition the design, you can isolate one part from another. This lets you preserve some compile points, and only resynthesize those that need to be rerun. These scenarios describe some design situations where compile points can be used to isolate parts of the design and run incremental synthesis:

- During the initial design phase, design modules are still being designed. Use compile points to preserve unchanged design modules and evaluate the effects of modifications to parts of the design that are still changing.

Copyright © 2013 Synopsys, Inc.

- During design integration, use compile points to preserve the main design modules and only allow the glue logic to be remapped.
- If your design contains IP, synthesize the IP, and use compile points to preserve them while you run incremental synthesis on the rest of the design.
- In the final stages of the design, use compile points to preserve design modules that do not need to be updated while you work through minor RTL changes in some other part of the design.

Automatic and Manual Compile Points

Compile points can be generated automatically by the tool or you can create them manually. A design can contain a mixture of automatic and manual compile points.

Automatic compile points are simple to use and do not require any setup. Manual compile points require more setup, but provide more control because they let you define the partition boundaries and constraints instead of the tool.

- Automatic compile points (ACP)

  Automatic compile points offer the simplest way to set up compile points and are also the most automated way to leverage multiprocessing. The tool makes the decisions and automatically creates compile points based on various parameters, like the size of the design, the sizes of hierarchical modules, their boundary logic, the number of ports driven by constants, and so on. For details about this process, see *Automatic Compile Point Generation*, on page 1112. You do not need to define boundary constraints for automatic compile points.

  The tool automatically sets automatic compile points as hard compile points. See *Compile Point Types*, on page 1092 for a description. The down side to using automatic compile points is that they might increase area if the partition boundaries prevent many cross-boundary optimizations.

- Manual compile points (MCP)

  Manual compile points provide more control. You can specify boundary constraints for each compile point individually. You can separate completed parts of the design from parts that are still being designed, or

fine-tune the compile points to take advantage of as many cross-boundary optimizations as possible. For example, you can ensure that a critical path does not cross a compile point boundary, thus ensuring synthesis results with optimal performance.

Guidelines for Using Automatic and Manual Compile Points

Determine the kind of compile point to use based on what the design requires. The table lists some guidelines:

Use Automatic Compile Points...	Use Manual Compile Points...
When runtime and quick results are more important than the best QoR	When you know the design in detail. Create manual compile points to get better QoR. Good candidates for manual compile points include the following: • Completed modules with registered interfaces, where you want to preserve the design • Modules created to include an entire critical path, so as to get the best performance. • Modules that are less likely to be affected by cross boundary optimizations like constant propagation and register absorption.
When you expect many updates or cross-boundary optimization changes.	When you do not want further optimizations to a completed compile point. When you want more control to determine cross-boundary optimizations on an individual basis.

Nested Compile Points

A design can have any number of compile points, and compile points can be nested inside other compile points. In the following figure, compile point CP6 is nested inside compile point CP5, which is nested inside compile point CP4.

To simplify things, the term *child* is used to refer to a compile point that is contained inside another compile point; the term *parent* is used to refer to a container compile point that contains a child. These terms are not used in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent

Copyright © 2013 Synopsys, Inc.

of all compile points. In the figure above, both CP5 and CP6 are children of CP4; both CP4 and CP5 are parents of CP6; CP5 is an immediate child of CP4 and an immediate parent of CP6.

Compile Point Types

Compile point designs do not have as good QoR as designs without them because the boundaries limit optimizations. Cross-boundary optimizations typically improve area and timing, at the expense of runtime. The compile point type determines whether boundary optimizations are allowed. The tool marks automatic compile points as hard by default. For manual compile points, you define the type. See *Defining the Compile Point Type*, on page 1118 for details.

These are descriptions of the soft, hard, locked, and black_box compile types:

- Soft

  Compile point boundaries can be reoptimized during top-level mapping. Timing optimizations like sizing, buffering, and DRC logic optimizations can modify boundary instances of the compile point and combine them with functions from the next higher level of the design. The compile point interface can also be modified. Multiple instances are uniquified. Any optimization changes can propagate both ways: into the compile point and from the compile point to its parent.

Using soft mode usually yields the best quality of results, because the software can utilize boundary optimizations. On the other hand, soft compile points can take a longer time to run than the same design with hard or locked compile points.

The following figure shows the soft compile point with a dotted boundary to show that logic can be moved in or out of the compile point.

- Hard

  For hard compile points, the compile point boundary can be reoptimized during top-level mapping and instances on both sides of the boundary can be modified by timing and DRC optimizations using top-level constraints. However, the boundary is not modified. Any changes can propagate in either direction while the compile point boundary (port/interface) remains unchanged. Multiple instances are uniquified. For performance improvements, constant propagation and removal of unused logic optimizations are performed across hard compile points.

  In the following figure, the solid boundary on the hard compile point indicates that no logic can be moved in or out of the compile point.

Copyright © 2013 Synopsys, Inc.

The hard compile point type allows for optimizations on both sides of the boundary without changing the boundary. There is a trade-off in quality of results to keep the boundaries. Using hard also allows for hierarchical equivalence checking for the compile point module.

Note: For automatic compile points, the default compile point is hard. The hard compile point, for automatic compile points, also has the functionality of the locked and locked, partition compile points. See *Locked*, on page 1094 and *Locked, partition*, on page 1096.

- Locked

  This is the default compile point type for manual compile points. With a locked compile point, the tool does not make any interface changes or reoptimize the compile point during top-level mapping. An interface logic model (ILM) of the compile point is created (see *Interface Logic Models*, on page 1100) and included for the top-level mapping. The ILM remains unchanged during top-level mapping.

  The locked value indicates that all instances of the same compile point are identical and unaffected by top-level constraints or critical paths. As a result, multiple instances of the compile point module remain identical even though the compile point is uniquified. The Technology view (srm file) shows unique names for the multiple instances, but in the final Verilog netlist (`vma` file) the original module names for the multiple instances are restored.

  Timing optimization can only modify instances outside the compile point. Although the compile point is used to time the top-level netlist,

changes do not propagate into or out of a locked compile point. The following figure shows a solid boundary for the locked compile point to indicate that no logic is moved in or out of the compile point during top-level mapping.

This mode has the largest trade-off in terms of QoR, because there are no boundary optimizations. So, it is very important to provide accurate constraints for locked compile points. The following table lists some advantages and limitations with the locked compile point:

Advantages	Limitations
Consumes smallest amount of memory. Used for large designs because of this memory advantage.	Interface timing
Provides most runtime advantage compared to other compile point types.	Constant propagation
Allows for obtaining stable results for a completed part of the design.	BUFG insertion
Allows for hierarchical place and route with multiple output netlists for each compile point and the top-level output netlist.	GSR hookup
Allows for hierarchical simulation.	IO pads, like IBUFs and OBUFs, should not be instantiated within compile points

Copyright © 2013 Synopsys, Inc.

Note: For automatic compile points, the default compile point is hard. The hard compile point, for automatic compile points, also has the functionality of the locked compile point.

- Locked, partition

  You can also specify a compile point type to be locked, partition. With this setting, the tool generates a netlist file for the compile points that are defined. Each compile point also includes a timestamp. This mode offers place-and-route runtime advantages and lets you converge on stable results for a completed design. However, this mode has the largest trade-off of quality of results because boundary optimizations are not allowed.

Note: For automatic compile points, the default compile point is hard. The hard compile point, for automatic compile points, also has the functionality of the locked, partition compile point.

- Black Box

  The tool treats black_box compile points as black boxes. It ignores the contents of the compile point and only uses its ports for synthesis. Black box compile point modules only write port definitions to the netlist files. The contents are not written to any of the netlist files (`srm`, `edf`, `vqm`, `edn`, `vm`, or `vhm`). This compile point type supports all black box directives.

  You can change the type of a compile point to black_box at any time during synthesis. The previous compile point results are retained from intermediate mapping `srd` files, but the parent compile point might be remapped. This table shows the results when the RTL is unchanged and a compile point (CP) is changed to black_box for the second synthesis run:

Original CP	Effect of changing to black_box
soft	The top level is resynthesized because of cross-boundary optimizations between the top-level and compile point A during the previous synthesis run. For the second run, since A is a black box, no logic from the top-level can be moved into A.
hard	Same as soft.
locked	Synthesis ignores its contents. When mapping other hierarchical compile points, the tool uses only the black box port information. It only writes port definitions to the output netlist files for these black box compile points.

Compile Point Type Summary

The following table summarizes how the tool handles different compile points during synthesis:

Features	Compile Point Type			
	Soft	**Hard**	**Locked**	**Black Box**
Boundary optimizations	Yes	Limited	No	No
Uniquification of multiple instance modules	Yes	Yes	Limited	No
Compile point interface (port definitions)	Modified	Not modified	Not modified	Not modified
Hierarchical simulation	No	Yes	Yes	Yes*
Hierarchical equivalence checking	No	Yes	Yes	Yes*
Interface Logic Model (created/used)	No	Yes	Yes	No

* If you replace the black box with the original RTL, you can run hierarchical simulation or hierarchical equivalence checking on the rest of the design.

Copyright © 2013 Synopsys, Inc.

Compile Point Synthesis Basics

This section describes the compile point constraint files and timing models, and describes the steps the tool goes through to synthesize compile points. See the following for details:

- Compile Point Constraint Files, on page 1098
- Interface Logic Models, on page 1100
- Interface Timing for Compile Points, on page 1101
- Compile Point Synthesis, on page 1104
- Incremental Compile Point Synthesis, on page 1107
- Forward-annotation of Compile Point Timing Constraints, on page 1108

For step-by-step information about how to use compile points, see *Synthesizing Compile Points*, on page 1109.

Compile Point Constraint Files

A compile point design can contain two levels of constraint files, as described below:

- The constraint file at the top level

  This is a required file, and contains constraints that apply to the entire design. This file also contains the definitions of the compile points in the design. The define_compile_point command is automatically written to the top-level constraint file for each compile point you define.

  The following figure shows that this design has one locked compile point, pgrm_cntr. It uses the following syntax to define the compile point:

  ```
  define_compile_point {v:work.prgm_cntr} -type {locked}
  ```

```
13
14 ###==== BEGIN Collections - (Populated from tab in SCOPE, do not edit)
15 ###==== END Collections
16
17 ###==== BEGIN Clocks - (Populated from tab in SCOPE, do not edit)
18 create_clock  {p:clock} -period {10}
19
20 ###==== END Clocks
21
22 ###==== BEGIN "Generated Clocks" - (Populated from tab in SCOPE, do not edit)
23 ###==== END "Generated Clocks"
24
25 ###==== BEGIN Inputs/Outputs - (Populated from tab in SCOPE, do not edit)
26 ###==== END Inputs/Outputs
27
28 ###==== BEGIN "Delay Paths" - (Populated from tab in SCOPE, do not edit)
29 ###==== END "Delay Paths"
30
31 ###==== BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
32 ###==== END Attributes
33
34 ###==== BEGIN "I/O Standards" - (Populated from tab in SCOPE, do not edit)
35 ###==== END "I/O Standards"
36
37 ###==== BEGIN "Compile Points" - (Populated from tab in SCOPE, do not edit)
38 define_compile_point  {v:work.prgm_cntr} -type {locked}
39 ###==== END "Compile Points"
40
```

- Constraints files at the compile point level

  These constraint files are optional, and are used for better control over manual compile points. If your design consists of automatic compile points only, you do not need any compile point-level files, and the tool uses interface timing to synthesize the individual compile points.

  The compile point constraints are specific to the compile point and only apply within it. If your design has manual compile points, you can define corresponding compile point constraint files for them. See *Setting Constraints at the Compile Point Level*, on page 1119 for a step-by-step procedure. Automatic compile points do not require compile point constraint files, because their constraints come from the top level.

  When compile point constraints are defined, the tool uses them to synthesize the compile point, not automatic interface timing. Note that depending on the compile point type, the tool might further optimize the compile points during top-down synthesis of the top level to improve timing performance and overall design results, but the compile point itself is synthesized with the defined compile point constraints.

Copyright © 2013 Synopsys, Inc.

The first command in a compile point constraint file is define_current_design, and it specifies the compile point module for the contained constraints. This command sets the context for the constraint file. The remainder of the file is similar to the top-level constraint file. For example:

```
define_current_design {work.pgrm_cntr}
```

```
12 define_current_design {work.prgm_cntr}
13 ###==== END Header
14
15 ###==== BEGIN Collections - (Populated from tab in SCOPE, do not edit)
16 ###==== END Collections
17
18 ###==== BEGIN Clocks - (Populated from tab in SCOPE, do not edit)
19 create_clock  {p:clock} -period {11.1111}
20
21 ###==== END Clocks
22
23 ###==== BEGIN "Generated Clocks" - (Populated from tab in SCOPE, do not edit)
24 ###==== END "Generated Clocks"
25
26 ###==== BEGIN Inputs/Outputs - (Populated from tab in SCOPE, do not edit)
27 ###==== END Inputs/Outputs
28
29 ###==== BEGIN "Delay Paths" - (Populated from tab in SCOPE, do not edit)
30 ###==== END "Delay Paths"
31
32 ###==== BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
33 ###==== END Attributes
34
35 ###==== BEGIN "I/O Standards" - (Populated from tab in SCOPE, do not edit)
36 ###==== END "I/O Standards"
37
38 ###==== BEGIN "Compile Points" - (Populated from tab in SCOPE, do not edit)
39 ###==== END "Compile Points"
```

If your design has some compile points with their own constraint files and others without them, the tool uses the defined compile point constraints when it synthesizes those compile points. For the other compile points without defined constraints, it uses automatic interface timing, as described in *Interface Timing for Compile Points*, on page 1101.

Interface Logic Models

The interface logic model (ILM) of a locked or hard compile point is a timing model that contains only the interface logic necessary for accurate timing. An ILM is a partial gate-level netlist that represents the original design

accurately while requiring less memory during mapping. Using ILMs improves the runtime for static timing analysis without compromising timing accuracy.

The tool does not do any timing optimizations on an ILM. The interface logic is preserved with no modifications. All logic required to recreate timing at the top level is included in the ILM. ILM logic includes any paths from an input/inout port to an internal register, an internal register to an output/inout port, and an input/inout port to an output/inout port.

The tool removes internal register-to-register paths, as shown in this example. In this design, and_a is not included in the ILM because the timing path that goes through and_a is an internal register-to-register path.

Interface Timing for Compile Points

By default, the synthesis tool automatically infers timing constraints for all compile points from the top-level constraints. However, if a compile point has its own constraint file, the tool applies those compile point-specific constraints to synthesize the compile point.

- For automatic interface timing, the tool derives constraints from the top level and uses them to synthesize the compile point. The top level is synthesized at the same time as the other compile points.
- When there are compile point constraint files, the tool first synthesizes the compile point using the constraints in the compile point constraints file and then synthesizes the top level using the top-level constraints.

Copyright © 2013 Synopsys, Inc.

When it synthesizes a compile point, the tool considers all other compile points as black boxes and only uses their interface timing information. In the following figure, when the tool is synthesizing compile point A, it applies relevant timing information to the boundary registers of B and C, because it treats them as black boxes.

Interface Timing Example

The design below shows how the interface timing works on compile points.

Contents of level1 Module

Interface Timing Off

Interface timing is off for a compile point when you define constraints for it in a compile point constraints file. In this example, the following frequencies are defined for the level1 compile point shown above:

Clock	Period	Constraints File
Top-level clock	10 ns	Top-level constraint file
Compile point-level clock	20 ns	Compile point constraint file

When interface timing is off, the compile point log file (`srr`) reports the clock period for the compile point as 20 ns, which is the compile point period.

Copyright © 2013 Synopsys, Inc.

Interface Timing On

For automatic interface timing to run on a compile point (interface timing on), there must not be a compile-point level constraints file. When interface timing is on, the compile point log file (`srr`) reports the clock period for the top-level design, which is 10 ns:

Compile Point Synthesis

During synthesis, the tool first synthesizes the compile points and then maps the top level. In the case of automatic compile points, the compile points and the top level are mapped simultaneously. The rest of this section describes the process that the tool goes through to synthesize compile points; for step-by-step information about what you need to do to use compile points, see *Synthesizing Compile Points*, on page 1109.

Automatic Compile Point Synthesis

The tool synthesizes all compile points individually. The top level is also treated as a compile point.

A compile point stands on its own, and is optimized separately from its parent environment (the compile point container or the top level). This means that critical paths from a higher level do not propagate downwards, and they are unaffected by them.

Automatic compile points have constraints automatically assigned from the top level, and you do not need to add any constraints at the compile point level.

By default, synthesis stops if the tool encounters an error while synthesizing a compile point. You can specify that the tool ignore the error and continue synthesizing other compile points. See *Continue on Error Mode*, on page 1106.

After synthesis for all the automatic compile points are done, the software reloads all the automatic compile point results and writes out a single output netlist and one constraint file for the entire design. See *Forward-annotation of Compile Point Timing Constraints*, on page 1108 for a description of the constraints that are forward-annotated.

Manual Compile Point Synthesis

Stage 1: Bottom-up Compile Point Synthesis

The tool synthesizes compile points individually from the bottom up. If you have enabled multiprocessing, it synthesizes the compile points in parallel using multiple processing jobs. For nested compile points, it starts with the compile point at the lowest level of hierarchy and works up the hierarchy.

A compile point stands on its own, and is optimized separately from its parent environment (the compile point container or the top level). This means that critical paths from a higher level do not propagate downwards, and they are unaffected by them.

If you have specified compile point-level constraints, the tool uses them to synthesize the compile point; if not, it uses automatic interface timing propagated from the top level. For compile point synthesis, the tool assumes that all other compile points are black boxes, and only uses the interface information.

Copyright © 2013 Synopsys, Inc.

When defined, compile point constraints apply within the compile point. Automatic compile points have constraints automatically assigned from the top level, and you do not need to add any constraints at the compile point level. For manual compile points, it is recommended that you set constraints on locked compile points, but setting constraints is optional for soft and hard compile points.

By default, synthesis stops if the tool encounters an error while synthesizing a compile point. You can specify that the tool ignore the error and continue synthesizing other compile points. See *Continue on Error Mode*, on page 1106.

Stage 2: Top-Level Synthesis

Once all the compile points have been synthesized, the tool synthesizes the entire design from the top down, using the model information generated for each compile point and constraints defined in the top-level constraints file. You do not need to duplicate compile point constraints at a higher level, because the tool takes the compile point timing models into account when it synthesizes a higher level. Note that if you run standalone timing analysis on a compile point, the timing report reflects the top-level constraints and not the compile point constraints, although the tool used compile point level constraints to synthesize the compile point.

The software writes out a single output netlist and one constraint file for the entire design. See *Forward-annotation of Compile Point Timing Constraints*, on page 1108 for a description of the constraints that are forward-annotated.

Continue on Error Mode

By default, the tool stops the synthesis process if it encounters an error within a compile point. If you want to be able to continue compile-point synthesis with another compile point, do any of the following:

- select Options->Configure Compile Point Process from the top menu and enable the Continue on Error checkbox
- select the Options panel on the Implementation Options dialog box and enable the Continue on Error checkbox
- check the Continue on Error checkbox on the left side of the Project view
- enter a set_option -continue_on_error option with a value of 1 at the Tcl script prompt

Copyright © 2013 Synopsys, Inc.

With this setting, when the tool encounters a mapper error it black boxes the affected compile point and continues to synthesize other compile points. The log file report after synthesis contains warnings like the following for the ignored errors:

```
@W:: m1.v(1) | Mapping of compile point m1 - Unsuccessful
@W:: m1.v(1) | Converting compile point m1 as black_box - as
continue_on_error is set
```

Note the following about the scope of this setting:

- The setting applies to both mapper errors and, in Synplify Premier, to certain, non-syntax-related compiler errors. All compiler errors, however, must be corrected before synthesis can proceed (for information on the impact of using Continue on Error with Synplify Premier compiler errors, see *Continue on Error*, on page 233).
- In Synplify Pro, the setting only applies to compile-point mapper errors. Top-level design errors terminate synthesis, and the tool does not generate an output netlist.

Incremental Compile Point Synthesis

The tool treats compile points as blocks for incremental synthesis. On subsequent synthesis runs, the tool runs incrementally and only resynthesizes those compile points that have changed, and the top level. The synthesis tool automatically detects design changes and resynthesizes compile points only if necessary. For example, it does not resynthesize a compile point if you only add or change a source code comment, because this change does not really affect the design functionality.

Copyright © 2013 Synopsys, Inc.

The tool resynthesizes a compile point that has already been synthesized, in any of these cases:

- The HDL source code defining the compile point is changed in such a way that the design logic is changed.
- The constraints applied to the compile point are changed.
- Any of the options on the Device panel of the Implementation Options dialog box, except Update Compile Point Timing Data, are changed. In this case the entire design is resynthesized, including all compile points.
- You intentionally force the resynthesis of your entire design, including all compile points, with the Run -> Resynthesize All command.
- The Update Compile Point Timing Data device mapping option is enabled and at least one child of the compile point (at any level) has been remapped. The option requires that the parent compile point be resynthesized using the updated timing model of the child. This includes the possibility that the child was remapped earlier, while the option was disabled. The newly enabled option requires that the updated timing model of the child be taken into account, by resynthesizing the parent.

For each compile point, the software creates a subdirectory named for the compile point, in which it stores intermediate files that contain hierarchical interface timing and resource information that is used to synthesize the next level. Once generated, the model file is not updated unless there is an interface design change or you explicitly specify it. If you happen to delete these files, the associated compile point will be resynthesized and the files regenerated.

Forward-annotation of Compile Point Timing Constraints

In addition to a top-level constraint file, each compile point can have its own constraint file. Constraints are forward-annotated to placement and routing from the top-level as well as the compile point-level files. However, not all compile point constraints are forward-annotated, as explained below. For example, constraints on top-level ports are always forward annotated, but compile point port constraints are not forward annotated.

- Top-level constraints are forward-annotated.
- Constraints applied to the interface (ports and bit ports) of the compile point are not forward-annotated.
  These include input_delays, output_delays, and clock definitions on the

ports. Such constraints are only used to map the compile point itself, not its parents. They are not used in the final timing report, and they are not forward-annotated.

- Constraints applied to instances inside the compile point are forward-annotated
  Constraints like timing exceptions and internal clocks are used to map the compile point and its parents. They are used in the final timing report, and they are forward-annotated.

Synthesizing Compile Points

This section describes the synthesis process with automatic compile points, manual compile points, or a combination of both in your design:

- The Automatic Compile Point Flow, next
- The Manual Compile Point Flow, on page 1113
- Creating a Top-Level Constraints File for Compile Points, on page 1116
- Defining Manual Compile Points, on page 1117
- Setting Constraints at the Compile Point Level, on page 1119
- Analyzing Compile Point Results, on page 1121
- Using Automatic and Manual Compile Points Together, on page 1123

The Automatic Compile Point Flow

The following figure shows how to set up and use automatic compile points (ACP) in a synthesis flow. The compile point setup section also shows how to use automatic compile points in a design that includes manual compile points (MCP).

Copyright © 2013 Synopsys, Inc.

1. Create a project and set implementation options as usual.
2. Set constraints.
3. Create a top-level constraints file and set compile point constraints, as described in *Creating a Top-Level Constraints File for Compile Points*, on page 1116.

   If your design is to include both manual and automatic compile points, you can define manual compile points at this stage and set constraints for them. Alternatively, you can generate the automatic compile points first and then specify manual compile points.
4. Specify that you want to generate compile points automatically.
   – Enable the Auto Compile Point option in the Project window, or set it on the Options tab of the Implementation Options dialog box. You can also set

it with the set_option -automatic_compile_point 1 command in the project file.

– If you do not want a module to be made into an automatic compile point, set the syn_no_compile_point attribute on that design module in the top-level constraint file. An automatic compile point is hard and does not allow for many optimizations, so set this attribute if you want the tool to optimize a particular module. If the compile point is a nested compile point, apply the attribute to each level of hierarchy to prevent the tool from creating a compile point from a sub-module.

See *Automatic Compile Point Generation*, on page 1112 for a description of the process the tool goes through to generate automatic compile points.

5. To improve runtime further, use multiprocessing. In the Synplify Premier tool, you can additionally specify Fast Synthesis.

6. Synthesize the design.

   The tool uses automatic interface timing to determine the constraints for the compile points. It first synthesizes individual compile points using interface timing propagated from the top level, and assumes that other compile points are black boxes. It then synthesizes the top level, as described in *Compile Point Synthesis*, on page 1104.

   For automatic compile points, the top level is treated as another compile point and is synthesized along with them.

7. Analyze the design.

Copyright © 2013 Synopsys, Inc.

For details, see *Analyzing Compile Point Results*, on page 1121. If you resynthesize the design, the tool uses incremental synthesis. See *Resynthesizing Compile Points Incrementally*, on page 1126 for details.

At this point, you can choose to create additional manual compile points as needed, by defining them in the top-level constraints file.

If you want to apply constraints to an automatically identified compile point, first define that compile point as a manual compile point and then apply constraints to it.

Automatic Compile Point Generation

The automatic compile point process does not generate new hierarchy. It honors existing hierarchy, so if you have RAMs, ROMs or DSPs that cross hierarchies, it does not disturb them. The automatic compile point process does not generate automatic compile points for IP modules like NGCs and EDIFs.

In a typical design, the tool goes through these stages to generate automatic compile points and their constraints.

- It first identifies compile points based on factors like the size of hierarchical modules, their boundary logic, and the number of hierarchical ports driven by constants.
- It then extracts compile point constraints from the top-level timing constraints, and propagates this interface timing automatically to the automatic compile points.
- If the design has manual compile points that do not have a constraint file at the top level, the tool derives constraints for them from the top level, just as with automatic compile points. If the design has manual compile points with constraints, the tool honors these defined constraints for the manual compile points.
- For black box compile point modules, the tool only writes port definitions to the netlist files. It does not write the contents of the module to any of the netlist files like srm, edf, vqm, edn, vm, or vhm.

Copyright © 2013 Synopsys, Inc.

The Manual Compile Point Flow

Using manual compile points is most advantageous in the following situations, where you

- Have to work with a large design
- Experience long runtimes, or need to reduce synthesis runtime
- Require the maximum QoR from logic synthesis
- Can adjust design methodology to get the best results from the tools

The following figure summarizes the process for using manual compile points in your design.

This procedure describes the steps in more detail:

1. Set up the project.

Copyright © 2013 Synopsys, Inc.

- Create the project and add RTL and IP files to the project, as usual.
- Target a device and technology for which compile points are supported. This includes most of the newer Achronix, Altera, Lattice, Microsemi, and Xilinx device families.
- If you are using the Synplify Premier tool, go to the GCC & Prototyping Tools tab of the Implementation Options dialog box, and enable the Feedthrough Optimization, Constant Propagation, Create Always/Process Level Hierarchy, and Optimize Netlist netlist prototype options. For certain Altera devices, you can also enable Create MAC Hierarchy.
- Set other options as usual.

2. Compile the design (F7) to initialize the constraints file.
3. Do the following in the top-level constraint file:
   - Define compile points in the top-level constraint file. See *Creating a Top-Level Constraints File for Compile Points*, on page 1116. Note that by default, the tool automatically calculates the interface timing for all compile points.
   - Set timing constraints and attributes in the top-level constraint file:

Constraint	Apply to...	Example
Clock	All clocks in the design.	create_clock {p:clk} -name clk -period 100 -clockgroup cg1
I/O constraints	All top-level port constraints. Register the compile point I/O boundaries to improve timing.	set_input_delay {p:a} {1} -clock {clk:r}
Timing exceptions	All timing exceptions that are outside the compile point module, or that might be partially in the compile point modules.	set_false_path -from {i:reg1} -to {i:reg2}
Attributes	All attributes that are applicable to the rest of the design, not within the compile points.	define_attribute {i:statemachine_1} syn_encoding {sequential}

4. Set compile point-specific constraints as needed in a separate, compile point-level constraint file.

See *Setting Constraints at the Compile Point Level*, on page 1119 for a step-by-step procedure. After setting the compile point constraints, add the compile point constraint file to the project.

5. If you do not want to interrupt synthesis for compiler errors, select Options->Configure Compile Point Process and enable the Continue on Error option.

   With this option enabled, the tool black boxes any compile points that have mapper errors and continues to synthesize the rest of the design. See *Continue on Error Mode*, on page 1106 for more information about this mode.

6. Synthesize the design.

   The tool synthesizes the compile points separately and then synthesizes the top level. See *Compile Point Synthesis*, on page 1104 for details about the process.

   – The first time it runs synthesis, the tool maps the entire design.
   – For subsequent synthesis runs, the tool only maps compile points that were modified since the last run. It preserves unchanged compile points.

   You can also run synthesis on individual compile points, without synthesizing the whole design.

7. Analyze the synthesis results using the top-level srr log file.

   See *Analyzing Compile Point Results*, on page 1121 for details.

8. If you do not meet your design goals, make necessary changes to the RTL, constraints, or synthesis controls, and re-synthesize the design.

   The tool runs incremental synthesis on the modified parts of the design, as described in *Incremental Compile Point Synthesis*, on page 1107. See *Resynthesizing Compile Points Incrementally*, on page 1126 for a detailed procedure.

Copyright © 2013 Synopsys, Inc.

Creating a Top-Level Constraints File for Compile Points

All compile points require a top-level constraints file. If you have manual compile points, define them in this file. The top-level file also contains design-level constraints. The following procedure describes how to create a top-level constraints file for a compile point design.

1. Create the top-level constraints file.
   - To define compile points in an existing top-level constraint file, open a SCOPE window by double-clicking the file in the Project view.
   - To define compile points in a new top-level constraint file, click the SCOPE icon. Click on the FPGA Constraints (SCOPE) button.

   The SCOPE window opens. It includes a Current Design field, where you can specify constraints for the top-level design from the drop-down menu and define manual compile points.

2. Set top-level constraints like input/output delays, clock frequencies or multicycle paths.

   You do not have to redefine compile point constraints at the top level as the tool uses them to synthesize the compile points.

3. Define manual compile points if needed.

   See *Defining Manual Compile Points*, on page 1117 for details.

4. Save the top-level constraints file and add it to the project.

Defining Manual Compile Points

Compile points and constraints are both saved in a constraint file, so this step can be combined with the setting of constraints, as convenient. This procedure only describes how to define compile points. You define compile points in a top-level constraint file. You can add the compile point definitions to an existing top-level constraint file or create a new file.

1. From the Current Design field, select the module for which you want to create the compile point.

2. Click the Compile Points tab in the top-level constraints file.

   See *Creating a Top-Level Constraints File for Compile Points*, on page 1116 if you need information about creating this file.

3. Set the module you want as a compile point.

   Do this by either selecting a module from the drop-down list in the View column, or dragging the instance from the HDL Analyst RTL view to the View column. The equivalent Tcl command is define_compile_point, as shown in this example:

```
define_compile_point {v:work.m3} -type {black_box}
```

   You can get a list of all the modules from which you can select and designate compile points with the Tcl find command, as shown here:

```
c_print [find -hier -view {*} -filter ((!@is_black_box) && (@is_verilog == 1 ||
@is_vhdl == 1))] -file view.txt
```


Copyright © 2013 Synopsys, Inc.

4. Set the Type to locked, locked,partition, hard, soft, or black_box, according to your design goals. See *Defining the Compile Point Type*, on page 1118 for details.

   This tags the module as a compile point. The following figure shows the prgm_cntr module set as a locked compile point:

5. Save the top-level constraint file.

You can now open the compile point constraint file and define constraints for the compile point, as needed for manual compile points. See *Setting Constraints at the Compile Point Level*, on page 1119 for details.

Defining the Compile Point Type

The compile point type you select depends on your design goals. For descriptions of the various compile point types, see *Compile Point Types*, on page 1092. This procedure shows you how to set the compile point type in the top-level constraint file when you define the compile points:

1. When runtime is the main objective and QoR is not a primary concern, set the compile point type as follows on the SCOPE Compile Points tab:

Situation	Compile Point Type
RTL is almost ready	locked
RTL is still being built but the module interface is ready	black_box If you define black box compile points, you must update the value and rerun synthesis after you have completed the RTL for the compile point.

The following example shows the Tcl command and the equivalent version in the in the SCOPE GUI:

define_compile_point {v:work.user_top} -type {locked}

	Enable	View	Type
1	☑	v:work.user_top	locked

2. When runtime and QoR are both important, do the following to ensure the best performance while still saving runtime:
   - Register the I/O boundaries for the compile points.
   - As far as possible, put the entire critical path into the same compile point.
   - Set each compile point type individually, using these compile point types:

Situation	Compile Point Type
Need boundary optimizations	soft
Do not need boundary optimizations	locked

3. If your goal is design preservation, set the compile point you want to preserve to locked.

Setting Constraints at the Compile Point Level

You can specify constraints for each compile point in individual constraint files. (See *Compile Point Constraint Files*, on page 1098 for a description of the files.) It is recommended that you specify constraints for each locked manual compile point, but you do not need to set them for soft and hard compile points. You do not need to set constraints for automatic compile points.

When you specify compile point constraints, the tool synthesizes the compile point using the compile point timing models instead of automatic interface timing from the top level. This procedure explains how to create a (compile point constraint file, and set constraints for the compile point:

1. In an open project, click the SCOPE icon ( ). Click on the FPGA Constraints (SCOPE) button. The New Constraints File dialog box opens.

Copyright © 2013 Synopsys, Inc.

2. From the Current Design field, select the module for which you want to create the compile point.

3. Check that you are in the right file.

   A default name for the compile point file appears in the banner of the SCOPE window. Unlike the top-level constraint file, the Compile Point tab in the SCOPE UI is greyed out when the constraint file is for a compile point.

4. Set constraints for the compile point. In particular, do the following:
   - Define clocks for the compile point.
   - Specify I/O delay constraints for non-registered I/O paths that may be critical or near critical.
   - Set port constraints for the compile point that are needed for top-level mapping.

   The tool uses the compile point constraints you define to synthesize the compile point. Compile point port constraints are not used at the parent level, because compile point ports do not exist at that level.

You can specify SCOPE attributes for the compile point as usual. See *Using Attributes with Compile Points*, on page 1121 for some exceptions.

5. Save the file and add it to the project. When prompted, click Yes to add the constraint file to the top-level design project.

   Otherwise, use Save As to write a file such as, *moduleName*.fdc to the current directory. The hierarchical paths for compile point modules in the constraint file are specified at the compile point level; not the top-level design.

Using Attributes with Compile Points

You can use attributes as usual when you set constraints for compile points. The following sections describe some caveats and exceptions:

- syn_hier

  When you use syn_hier on a compile point, the only valid value is flatten. All other values of this attribute are ignored for compile points. The syn_hier attribute behaves normally for all other module boundaries that are not defined as compile points.

- syn_allowed_resources

  Apply the syn_allowed_resources attribute globally or to a compile point to specify its allowed resources. When a compile point is synthesized, the resources of its siblings and parents cannot be taken into account because it stands alone as an independent synthesis unit. This attribute limits dedicated resources such as block RAMs or DSPs that the compile point can use, so that there are adequate resources available during the top-down flow.

Analyzing Compile Point Results

The software writes all timing and area results to a single log file in the implementation directory. You can check this file and the RTL and Technology views to determine if your design has met the goals for area and performance. You can also view and isolate the critical paths, search for and highlight design objects and crossprobe between the schematics and source files.

1. Check that the design meets the target frequency for the design. Use the Watch window or check the log file.

Copyright © 2013 Synopsys, Inc.

2. Open the log file and check the following:
   - Check top-level and compile point boundary timing. You can also check this visually using the RTL and Technology view schematics. If you find negative slack, check the critical path. If the critical path crosses the compile point boundary, you might need to improve the compile point constraints.
   - If the design was resynthesized, check the Summary of Compile Points section to see if compile points were preserved or remapped.

```
Summary of Compile Points :
***************************
Name          Status      Reason                  Start Time                  End Time                    Realtime      CPU Time
-------------------------------------------------------------------------------------------------------------------------------------
user_top      Remapped    Mapping options changed Mon Mar 14 04:43:42 2011    Mon Mar 14 04:44:27 2011    0h:00m:44s    0h:00m:33s
```

   Note that this section reports black box compile points as Not Mapped, and lists the reason as Black Box.
   - Review all warnings and determine which should be addressed and which can be ignored.
   - Review the area report in the log file and determine if the cell usage is acceptable for your design.
   - Check all DRC information.
3. Check other files:
   - Check the individual compile point module log files. The tool creates a separate directory for each compile point module under the implementation directory. Check the compile point log file in this directory for synthesis information about the compile point synthesis run.
   - Check the compile point timing report. This report is located in the compile point results directory of the implementation directory for each compile point.
4. Check the RTL and Technology view schematics for a graphic view of the design logic. Even though instantiations of compile points do not have unique names in the output netlist, they have unique names in the Technology view. This is to facilitate timing analysis and the viewing of critical paths.

Copyright © 2013 Synopsys, Inc.

Note: Compile points of type {hard} and {locked, partition} are easily located in the Technology view with the color green.

5. Fix any errors.

   Remember that the mapper reports an error if synthesis at a parent level requires that interface changes be made to a locked compile point. The software does not change the compile point interface, even if changes are required to fix DRC violations.

Using Automatic and Manual Compile Points Together

There are two ways to use automatic and manual compile points together in the same design:

- Have the tool generate automatic compile points and then define one or more of them as manual compile points. Add constraint files for the manual compile points to the project if needed.
- Start with some manual compile points defined. Then enable the automatic compile points option in the synthesis tool, and let the tool generate automatic compile points in addition to the manual ones that have already been defined.

Copyright © 2013 Synopsys, Inc.

Using Compile Points with Other Features

You can effectively combine compile points with other synthesis features for better runtime. The following sections describe how you can use compile points with fast synthesis and multiprocessing:

- Combining Compile Points with Fast Synthesis, on page 1124
- Combining Compile Points with Multiprocessing, on page 1125

Combining Compile Points with Fast Synthesis

You can use the Synplify Premier Fast Synthesis mode in a compile point design to improve runtime.

Do the following to use compile points with fast synthesis:

1. Set up the project with compile points.

   Automatic compile points are faster to set up than manual compile points.

2. Specify fast synthesis, in one of these ways. Disable the Physical Synthesis option.

Globally	Enable Fast Synthesis from the Project window.
On individual compile points	Open the top-level constraints file, and apply the syn_cp_use_fast_synthesis attribute to the compile point module, as shown in this example:

3. Specify other features. This is a list of combinations that improve runtime, with the fastest runtime results first:
   - Fast synthesis, multiprocessing, and automatic compile points
   - Fast synthesis and automatic compile points
   - Fast synthesis and manual compile points
4. Synthesize the design.

Combining Compile Points with Multiprocessing

To use compile points with multiprocessing, do the following.

1. Set up the project with compile points.
2. Specify the number of parallel jobs to run with the Options->Configure Compile Point Process command.

   Alternatively, you can set this with the `set_option -max_parallel_jobs` Tcl command, or in the ini file.
3. Run synthesis.

   The software synthesizes the compile points as separate processor jobs. Parallel processing reduces runtime. If you are using the Synplify Premier tool, you can use Fast Synthesis mode to further reduce runtime.

Copyright © 2013 Synopsys, Inc.

Resynthesizing Incrementally

Incremental synthesis can significantly reduce runtime on subsequent runs. It can also help with design stabilization and preservation. The following describe the incremental synthesis process, and how compile points are used in incremental synthesis within the tool and with other tools:

- Incremental Compile Point Synthesis, on page 1107
- Resynthesizing Compile Points Incrementally, on page 1126
- Synthesizing Incrementally with Other Tools, on page 1129

Resynthesizing Compile Points Incrementally

The following figure illustrates how compile points (CP) are used in incremental synthesis.

1. To synthesize a design incrementally, make the changes you need to fix errors or improve your design.
   - Define new compile point constraints or modify existing constraints in the existing constraint file or in a new constraint file for the compile point. Save the file.
   - If necessary, reset implementation options. Click Implementation Options and modify the settings (operating conditions, optimization switches, and global frequency).

   To obtain the best results, define any required constraints and set the proper implementation options for the compile point before resynthesizing.

2. Click Run to resynthesize the design.

   When a design is resynthesized, compile points are not resynthesized unless source code logic, implementation options, or constraints have been modified. If there are no compile point interface changes, the software synthesizes the immediate parent using the previously generated model file for the compile point. See *Incremental Compile Point Synthesis*, on page 1107 for details.

3. Check the log file for changes.

   The following figure illustrates incremental synthesis by comparing compile point summaries. After the first run, a syntax change was made in the mult module, and a logic change in the comb_logic module. The figure shows that incremental synthesis resynthesizes comb_logic (logic change), but does not resynthesize mult because the logic did not change even though there was a syntax change. Incremental synthesis re-uses the mapped file generated from the previous run to incrementally synthesize the top level.

Copyright © 2013 Synopsys, Inc.

4. To force the software to generate a new model file for the compile point, click Implementation Options on the Device tab and enable Update Compile Point Timing Data. Click Run.

   The software regenerates the model file for each compile point when it synthesizes the compile points. The new model file is used to synthesize the parent. The option remains in effect until you disable it.

5. To override incremental synthesis and force the software to resynthesize all compile points whether or not there have been changes made, use the Run->Resynthesize All command.

   You might want to force resynthesis to propagate changes from a locked compile point to its environment, or resynthesize compile points one last time before tape out. When you use this option, incremental synthesis is disabled for the current run only. The Resynthesize All command does not regenerate model files for the compile points unless there are interface changes. If you enable Update Compile Point Timing Data and select Resynthesize All, you can resynthesize the entire design and regenerate the compile point model files, but synthesis will take longer than an incremental synthesis run.

Copyright © 2013 Synopsys, Inc.

Synthesizing Incrementally with Other Tools

Just as with incremental synthesis, other downstream tools can also take advantage of the RTL compile points to run incrementally and speed up runtime through their processes. The Xilinx and Altera place-and-route tools can leverage the compile points and use them to run incrementally. Similarly, the Identify tool can run incremental instrumentation if it starts with a synthesis design with compile points.

For details about how these tools use compile points to run incrementally, see the following:

- Running Altera Quartus II Incrementally, on page 846
- Running Xilinx ISE Incrementally, on page 869
- Working with the Identify Tools, on page 880

Copyright © 2013 Synopsys, Inc.

CHAPTER 22

Optimizing Processes for Productivity

This chapter covers topics that can help the advanced user improve productivity and inter operability with other tools. It includes the following:

- Using Batch Mode, on page 1132
- Working with Tcl Scripts and Commands, on page 1139
- Automating Flows with synhooks.tcl, on page 1146
- Using Revision Control Tools, on page 1149

Copyright © 2013 Synopsys, Inc.

Using Batch Mode

Batch mode is a command-line mode in which you run scripts from the command line. You might want to set up multiple synthesis runs with a batch script. You can run in batch mode if you have a floating license, but not with a node-locked license.

Batch scripts are in Tcl format. For more information about Tcl syntax and commands, see *Working with Tcl Scripts and Commands*, on page 1139.

This section describes the following operations:

- Running Batch Mode on a Project File, on page 1132
- Running Batch Mode with a Tcl Script, on page 1133
- License Queuing, on page 1134

Running Batch Mode on a Project File

Use this procedure to run batch mode if you already have a project file set up. You can also run batch mode from a Tcl script, as described in *Running Batch Mode with a Tcl Script*, on page 1133.

1. Make sure you have a project file (`prj`) set up with the implementation options. For more information about creating this Tcl file, see *Creating a Tcl Synthesis Script*, on page 1141.

2. From a command prompt, go to the directory where the project files are located, and type one of the following, depending on which product you are using:

   ```
   synplify -batch project_file_name.prj
   synplify_pro -batch project_file_name.prj
   synplify_premier -batch project_file_name.prj
   synplify_premier_dp -batch project_file_name.prj
   ```

   The software runs synthesis in batch mode. Use absolute path names or a variable instead of a relative path name.

   The software returns the following codes after the batch run:

   0 - OK
   2 - logical error
   3 - startup failure

4 - licensing failure
5 - batch not available
6 - duplicate-user error
7 - project-load error
8 - command-line error
9 - Tcl-script error
20 - graphic-resource error
21 - Tcl-initialization error
22 - job-configuration error
23 - parts error
24 - product-configuration error
25 - multiple top levels

3. If there are errors in the source files, check the standard output for messages. On Linux systems, this is generally the monitor; on Windows systems, it is the stdout.log file.

4. After synthesis, check the *resultFile*.srr log file for error messages about the run.

Running Batch Mode with a Tcl Script

The following procedure shows you how to create a Tcl batch script for running synthesis. If you already have a project file set up, use the procedure described in *Running Batch Mode on a Project File*, on page 1132.

1. Create a Tcl batch script. See *Creating a Tcl Synthesis Script*, on page 1141 for details.

2. Save the file with a `tcl` extension to the directory that contains your source files and other project files.

3. From a command prompt, go to the directory with the files and type one of the following as appropriate:

```
synplify -batch Tcl_script.tcl
synplify_pro -batch Tcl_script.tcl
synplify_premier -batch Tcl_script.tcl
synplify_premier_dp -batch Tcl_script.prj
```

The software runs synthesis in batch mode. The synthesis (compilation and mapping) status results and errors are written to the log file *resultFile*.srr for each implementation. The synthesis tool also reports success and failure return codes.

Copyright © 2013 Synopsys, Inc.

4. Check for errors.
   - For source file or Tcl script errors, check the standard output for messages. On Linux systems, this is generally the monitor in addition to the stdout.log file; on Windows systems, it is the stdout.log file.
   - For synthesis run errors, check the *resultFile*.srr log file. The software uses the following error codes:

     0 - OK
     2 - logical error
     3 - startup failure
     4 - licensing failure
     5 - batch not available
     6 - duplicate-user error
     7 - project-load error
     8 - command-line error
     9 - Tcl-script error
     20 - graphic-resource error
     21 - Tcl-initialization error
     22 - job-configuration error
     23 - parts error
     24 - product-configuration error
     25 - multiple top levels

License Queuing

A common problem when running in batch mode is that the run fails because all of the available licenses are in use. License queuing allows a batch run to wait for the next available license when a license is not immediately available. You can use the following types of license queuing:

- Blocking-style Queuing
- Non Blocking-style Queuing
- Synopsys DesignWare IP License Queuing

Blocking-style Queuing

When blocking-style queuing is enabled and the requested license feature is on the server but not available, the tool waits until the license becomes available.

To enable blocking-style license queuing, either:

- Set environment variable *toolName*_LICENSE_WAIT=1 (*toolName* is the name of the FPGA synthesis tool)
- Include a -license_wait command-line argument when launching batch mode as shown in the following examples:

```
synplify_pro -batch -license_wait Tcl_script.tcl

synplify_premier -batch -license_wait projectFilename.prj
```

For blocking-style license queuing, the following message is generated to stdout.log or the Tcl window:

```
Waiting for license: toolName
```

For example:

```
Waiting for license: synplifypro
```

Queuing Considerations

When using queuing:

- A blocking-style queuing is used; license checkout does not exit until a license becomes available.
- There is no maximum wait time; once initiated, the tool can wait indefinitely for a license.
- If the server shuts down while the tool is waiting, a checkout failure is reported.
- When two licenses are required, queuing waits only until the first license becomes available (and not the second) to avoid holding a license unnecessarily.

Non Blocking-style Queuing

When non blocking-style queuing is enabled and the requested license feature is on the server but not available, the tool waits up to the maximum wait time specified in seconds for the license to become available.

Copyright © 2013 Synopsys, Inc.

To enable non blocking-style license queuing, you can:

- Set environment variable *toolName*_LICENSE_WAIT=*waitTime* (*toolName* is the name of the FPGA synthesis tool and *waitTime* is the maximum wait time in seconds). For example:

```
SYNPLIFYPRO_LICENSE_WAIT=180

SNNPLIFYPREMIER_LICENSE_WAIT=300
```

- Include a -license_wait *waitTime* command-line argument when launching batch mode as shown in the following examples:

```
synplify_pro -batch -license_wait waitTime Tcl_script.tcl

synplify_premier -batch -license_wait waitTime projectFilename.prj
```

For non blocking-style license queuing, the following message is generated to stdout.log or the Tcl window:

```
Waiting up to n seconds for license: toolName
```

For example:

```
Waiting up to 230 seconds for license: synplifypro
```

Non blocking-style license queuing behavior for the environment variable *toolName*_LICENSE_WAIT=*waitTime* or -license_wait *waitTime* command-line argument is shown in the following tables.

Environment Variable	Queuing Behavior
SYNPLIFYPRO_LICENSE_WAIT (undefined)	Queuing off
SYNPLIFYPRO_LICENSE_WAIT=0	Queuing off
SYNPLIFYPRO_LICENSE_WAIT=1	Queuing on; wait indefinitely
SYNPLIFYPRO_LICENSE_WAIT=*n* (n>1)	Queuing on; wait up to *n* seconds

Command Line Argument	Queuing Behavior
synplifypro -license_wait *arguments*	Queuing on; wait indefinitely
synplifypro -license_wait 1 *arguments*	Queuing on; wait indefinitely

Copyright © 2013 Synopsys, Inc.

Command Line Argument	Queuing Behavior
synplifypro license_wait *n* (n>1)	Queuing on; wait up to *n* seconds
synplifypro *arguments*	Queuing off
synplifypro -license_wait 0 *arguments*	Queuing off

- Specify a list of features to wait for using batch mode:

```
synplipfy_pro -batch -license_wait -licensetype
synplifypro:synplifypro_altera name_of_project.prj
```

- Specify feature names to wait for using the *toolName*_LICENSE_TYPE environment variable:

```
SYNPLIFYPRO_LICENSE_TYPE=synplifypro:synplifypro_altera
```

Synopsys DesignWare IP License Queuing

IP license queuing supports IP license requests through a command line parameter. The license queuing mechanism has no precedence over other Synopsys products such as Design Compiler and operates independently from their license queues. With Synopsys DesignWare IP license queuing:

- Requested Synopsys DesignWare IP licenses are queued when the license is not immmediately available.
- In batch mode, the synthesis tool waits for an available IP license. When no license is available, the synthesis tool waits for either a specified time period or indefinitely.
- If the wait time elapses and an IP license is still not available, the synthesis tool continues processing using any available license.
- An IP block without a requested license is processed as either an error or a black box according to the project settings.

To enable IP license queuing in batch mode, the synthesis tool accepts the following command-line parameter:

```
-ip_license_wait n
```

In the parameter, *n* is the number of seconds to wait for a license; when *n* = 1, the synthesis tool waits indefinitely for the license checkout for the requested IP licenses. If all requested licenses are checked-out or if the specified wait

Copyright © 2013 Synopsys, Inc.

time elapses, the tool excludes the IP and continues to process the remaining design. Any IP block without a license is treated either as an error or a black box.

The following examples illustrate command usage:

```
synplify_pro -batch -ip_license_wait waitTime Tcl_script.tcl
synplify_premier -batch -ip_license_wait waitTime projectFilename.prj
```

The following table outlines the queuing behavior:

Command Line Argument	Queuing Behavior
synplifypro -ip_license_wait *arguments*	Queuing on; wait indefinitely
synplifypro -ip_license_wait 1 *arguments*	Queuing on; wait indefinitely
synplifypro -ip_license_wait *n* (n>1)	Queuing on; wait up to *n* seconds
synplifypro -ip_license_wait 0 *arguments*	Queuing off

Working with Tcl Scripts and Commands

The software uses extensions to the popular Tcl (Tool Command Language) scripting language to control synthesis and for constraint files. See the following for more information:

- Using Tcl Commands and Scripts, next
- Generating a Job Script, on page 1140
- Setting Number of Parallel Jobs, on page 1140
- Creating a Tcl Synthesis Script, on page 1141
- Using Tcl Variables to Try Different Clock Frequencies, on page 1143
- Using Tcl Variables to Try Several Target Technologies, on page 1144
- Running Bottom-up Synthesis with a Script, on page 1145

You can also use synhooks Tcl scripts, as described in *Automating Flows with synhooks.tcl*, on page 1146.

Using Tcl Commands and Scripts

1. To get help on Tcl syntax, do any of the following:
   - Refer to the online help (Help->Tcl Help) for general information about Tcl syntax.
   - Refer to the *Reference Manual* for information about the synthesis commands.
   - Enter help * in the Tcl window for a list of all the Tcl synthesis commands. The Tcl window is not available in Synplify.
   - Enter help *commandName* in the Tcl window to see the syntax for an individual command.
2. To run a Tcl script, do the following:
   - Create a Tcl script. Refer to *Generating a Job Script*, on page 1140 and *Creating a Tcl Synthesis Script*, on page 1141.
   - Run the Tcl script by either entering source *Tcl_scriptfile* in the Tcl script window, or by selecting File->Run Tcl Script, selecting the Tcl file, and clicking Open.

Copyright © 2013 Synopsys, Inc.

The software runs the selected script by executing each command in sequence. For more information about Tcl scripts, refer to the following sections.

Generating a Job Script

You can record Tcl commands from the interface and use it to generate job scripts.

1. In the Tcl script window, enter recording -file *logfile* to write out a Tcl log file.
2. Work through a synthesis session.

   The software saves the commands from this session into a Tcl file that you can use as a job script or as a starting point for creating other Tcl files.

For the command syntax, see *recording*, on page 1673 in the *Reference* manual.

Setting Number of Parallel Jobs

You can set the maximum number of parallel jobs by setting a variable in the `ini` file, by defining a Tcl variable, or specifying the maximum number in the GUI.

1. To set the maximum number of parallel jobs in the `ini` file, do the following:
   - Open the `ini` file for the synthesis tool. For example, synplify_premier_dp.ini.
   - Add the MaxParallelJobs variable to the `ini` file, as follows:

     ```
     [JobSetting]
     MaxParallelJobs=<n>
     ```

   The tool uses the MaxParallelJobs value from the `ini` file as the default for both the UI (Project->Options) and batch mode. This value remains in effect until you reset it in the `ini` file or from the GUI, as described in the next step. To locate this configuration and initializaion file (`ini`), see *Input Files*, on page 594.

2. To set or change the maximum number of parallel jobs from the GUI, do the following:
   - Select Project->Options->Configure Compile Point Process.
   - Set the value you want in the Maximum number of parallel synthesis jobs field, and click OK. This field shows the current `ini` value, but you can reset it, and it will remain in effect until you change it again. The value you set is saved to the `ini` file.
3. To set a Tcl variable for the maximum number of parallel jobs, do the following:
   - Determine where you are going to define the variable. You can do this in the project file, or a Tcl file, or you can type it in the Tcl window. If you specify it in a Tcl file, you must source the file. If you specify it in the Tcl window, the tool does not save the value, and it will be lost when you end the current session.
   - Specify the max_parallel_jobs variable with the set_option Tcl command:

     `set_option -max_parallel_jobs` *value*

   The tool applies the max_parallel_jobs *value* specified to all project files and their respective implementations. This is a global option. The maximum number of parallel jobs remains in effect until you specify a new value. This new value takes effect immediately, going forward. However, when you set this option from the Tcl command window, the max_parallel_jobs value is not saved and will be lost when you exit the application.

Creating a Tcl Synthesis Script

Tcl scripts are text files with a `tcl` extension. You can use the graphic user interface to help you create a Tcl script. Interactive commands that you use actually execute Tcl commands, which are displayed in the Tcl window as they are run. You can copy the command text and paste it into a text file that you build to run as a Tcl script. For example:

```
add_file prep2.v
set_option -technology STRATIX
set_option -part EP1SGX40D
set_option -package FC1020

project -run
```


Copyright © 2013 Synopsys, Inc.

The following procedure covers general guidelines for creating a synthesis script.

1. Use a text file editor or select File->New, click the Tcl Script option, and type a name for your Tcl script.

2. Start the script by specifying the project with the project -new command. For an existing project, use project -load *project*.prj.

3. Add files using the add_file command. The files are added to their appropriate directories based on their file name extensions (see *add_file*, on page 1639 in the *Reference Manual*). Make sure the top-level file is last in the file list:

   ```
   add_file statemach.vhd
   add_file rotate.vhd
   add_file memory.vhd
   add_file top_level.vhd
   add_file design.fdc
   ```

   For information on constraints and vendor-specific attributes, see *Using a Text Editor for Constraint Files (Legacy)*, on page 122 for details about constraint files.

4. Set the design synthesis controls and the output:
   - Use the set_option command for setting implementation options and vendor-specific controls as needed. See the appropriate vendor chapter in the *Synplify Reference Manual* for details.
   - Set the output file information with project -result_file and project -log_file.

5. Set the file and run options:
   - Save the project with a project -save command
   - Run the project with a project -run command
   - Open the RTL and Technology views:

     ```
     open_file -rtl_view
     open_file -technology_view
     ```

6. Check the syntax.
   - Check case (Tcl commands are case-sensitive).
   - Start all comments with a hash mark (#).

– Always use a forward slash (/) in directory and pathnames, even on the Windows platform.

Using Tcl Variables to Try Different Clock Frequencies

To create a single script for multiple synthesis runs with different clock frequencies, you need to create a Tcl variable for the different settings you want to try. For example, you might want to try different target technologies.

1. To create a variable, use this syntax:

```
set variable_name {
   first_option_to_try
   second_option_to_try
   ...}
```

2. Create a foreach loop that runs through each option in the list, using the appropriate Tcl commands. The following example shows a variable set up to synthesize a design with different frequencies. It also creates a separate log file for each run.

The following code shows the complete script:

Copyright © 2013 Synopsys, Inc.

```
project -load design.prj
set try_these {
   20.0
   24.0
   28.0
   32.0
   36.0
   40.0
}

foreach frequency $try_these {
   set_option -frequency $frequency
   project -log_file $frequency.srr
   project -run
   open_file -edit_file $frequency.srr
}
```

Using Tcl Variables to Try Several Target Technologies

This technique used here to run multiple synthesis implementations with different target technologies is similar to the one described in *Using Tcl Variables to Try Different Clock Frequencies*, on page 1143. As in that section, you use a variable to define the target technologies you want to try.

1. Create a variable called try_these with a list of the technologies.

```
set try_these {

      STRATIXII CYCLONEII VIRTEX2 # list of technologies
}
```

2. Add a foreach loop that creates a new implementation for each technology and opens the RTL view for each implementation.

```
foreach technology $try_these {
   impl -add
   set_option -technology $technology
   project -run -fg
   open_file -rtl_view
}
```

The following code example shows the script:

```
# Open a new project, set frequency, and add files.
   project -new
   set_option -frequency 33.3
   add_file -verilog D:/test/simpletest/prep2_2.v

# Create the Tcl variable to try different target technologies.
   set try_these
      STRATIXII CYCLONEII VIRTEX2 # list of technologies
   }

# Loop through synthesis for each target technology.
   foreach technology $try_these {
      impl -add
      set_option -technology $technology
      project -run -fg
      open_file -rtl_view
   }
```

Running Bottom-up Synthesis with a Script

To run bottom-up synthesis, you create Tcl scripts for individual logic blocks, and a script for the top level that reads the other Tcl scripts.

1. Create a Tcl script for each logic block. The Tcl script must synthesize the block. See *Creating a Tcl Synthesis Script*, on page 1141 for details.
2. Create a top-level script that reads the block scripts. Create the script with the with the project -new command.
3. Add the top-level data:
   - Add source and constraint files with the add_file command.
   - Set the top-level options with the set_option command.
   - Set the output file information with project -result_file and project -log_file.
   - Save the project with a project -save command.
   - Run the project with a project -run command.
4. Save the top-level script, and then run it using this syntax:

   source *block_script*.**tcl**

   When you run this command, the entire design is synthesized, beginning with the lower-level logic blocks specified in the sourced files, and then the top level.

Copyright © 2013 Synopsys, Inc.

Automating Flows with synhooks.tcl

This procedure provides the advanced user with callbacks that let you customize your design flow or integrate with other products. For example, you might use the callbacks to send yourself email when a job is done (see *Automating Message Filtering with a Tcl Script*, on page 526), or to automatically copy files to another location after mapping. You can use the callback functions to integrate with a version control system, or generate the files needed to run formal verification with the Cadence Conformal tool. The procedure is based on a file called synhooks.tcl, which contains the Tcl callbacks.

1. Copy the synhooks.tcl file from the *installDirectory*/examples directory to a new location.

   You must copy the file to a new location so that it does not get overwritten by subsequent product installations and you can maintain your customizations from version to version. For example, copy it to C:/work/synhooks.tcl.

2. Define an environment variable called SYN_TCL_HOOKS, and point it to the location of the synhooks.tcl file.

3. Open the synhooks.tcl file in a text editor, and edit the file so that the commands reflect what you want to do. The default file contains examples of the callbacks, which provide you with hooks at various points of the design process.

   – Customize the file by deleting the ones you do not need and by adding your customized code to the callbacks you want to use. The following table summarizes the various design phases where you can use the callbacks and lists the corresponding functions. For details of the syntax, refer to *synhooks File Syntax*, on page 1719 in the *Reference Manual*.

Design Phase	Tcl Callback Function
Project Setup Callbacks	
Settings defaults for projects	proc syn_on_set_project_template
Creating projects	proc syn_on_new_project
Opening projects	proc syn_on_open_project

Design Phase	Tcl Callback Function
Closing projects	proc syn_on_close_project
Application Callbacks	
Starting the application after opening a project	proc syn_on_start_application
Exiting the application	proc syn_on_exit_application
Run Callbacks	
Starting a run. See *Example: proc syn_on_start_run*, on page 1147.	proc syn_on_start_run
Ending a run	proc syn_on_end_run
Key Assignment Callbacks	
Setting an operation for Ctrl-F8. See *Example: proc syn_on_press_ctrl_f8*, on page 1148.	proc syn_on_press_ctrl_f8
Setting an operation for Ctrl-F9	proc syn_on_press_ctrl_f9
Setting an operation for Ctrl-F11	proc syn_on_press_ctrl_f11

– Save the file.

As you synthesize your design, the software automatically executes the function callbacks you defined at the appropriate points in the design flow.

Example: proc syn_on_start_run

The following code example gets selected files from the project browser at the start of a run:

Copyright © 2013 Synopsys, Inc.

```
proc syn_on_start_run {compile c:/work/prep2.prj rev_1} {
   set sel_files [get_selected_files -browser]
      while {[expr [llength $sel_files] > 0]} {
         set file_name [lindex $sel_files 0]
         puts $file_name
         set sel_files [lrange $sel_files 1 end]
      }
}
```

Example: proc syn_on_press_ctrl_f8

The following code example gets all the selected files from the project browser and project directory when the Ctrl-F8 key combination is pressed:

```
proc syn_on_press_ctrl_f8 {} {
   set sel_files [get_selected_files]
      while {[expr [llength $sel_files] > 0]} {
         set file_name [lindex $sel_files 0]
         puts $file_name
         set sel_files [lrange $sel_files 1 end]
      }
}
```

Copyright © 2013 Synopsys, Inc.

Using Revision Control Tools

Synplify Premier (Beta)

You integrate a third-party revision control tool and use its functionality to manage design files for large and complex designs in your project. Use the Configure Revision Control Tools process to help you manage your design files.

Note: Third-Party Revision Control Tool Prerequisite

The following procedure shows you how to configure and use a third-party revision control tool to manage your design files.

1. Set up the third-party revision control tool, and set all required environment variables.
2. Select Project->Configure Revision Control (Beta).

   By default, the third-party revision control tools supported are Perforce and CVS. However, you are free to choose any revision control tool of your choice.

Copyright © 2013 Synopsys, Inc.

Configure Revision Control Tools (Beta)
Enable Revision Control Tool Integration
Include timestamps in messages submitted to server
Current Configuration: CVS Default
Edit Tcl...
Standard Commands
Action
Confirmation
Command TCL
Add Files to Repository
scm_add_files $FileList
Check In
scm_check_in $FileList $Message
Get Latest Revision
scm_get_latest_revision $FileList
Refresh
scm_file_status $FileList
Custom Command
Action
Add
New Configuration.
Configure Revision Control Tools (Beta)
Enable Revision Control Tool Integration
Include timestamps in messages submitted to server
Current Configuration: Perforce Default
Edit Tcl...
Standard Commands
Action
Confirmation
Command TCL
Add Files to Repository
scm_add_files $FileList
Check In
scm_check_in $FileList $Message
Check Out
scm_check_out $FileList
Get Latest Revision
scm_get_latest_revision $FileList
Custom Commands
Action
Confirmation
Command TCL
Add
Delete
New Configuration...
OK
Cancel

3. Do the following on the dialog box to configure the revision control tool:
   - Select the Enable Revision Control Tool Integration option. This enables the other fields and makes the revision control tool commands available in the Project view.
   - Select the revision control tool you want to use from the Current Configuration menu. The synthesis tool automatically includes the standard default commands for the revision control tool you selected. You can only edit the Confirmation field; if enabled, you will be asked to confirm the revision control operation.

     In the following example, Perforce is specified as the revision control tool below.

   - The tool automatically includes the standard default commands for the revision control tool specified in a Tcl script file. Click on the Edit Tcl button, to look over these standard default commands and edit them in the Tcl script file if needed. For examples of the Tcl script file, see *SCM Tcl Script File*, on page 193.

     For command syntax, refer to the documentation for the specified revision control tool.
   - To create custom commands, click on the Add button. Specify the command name, if you would like confirmation of this revision

Copyright © 2013 Synopsys, Inc.

control operation, and its Tcl command string. The first example below is a simple command that can be used as defined. However, the second example is more complex. Click on the Edit Tcl button to add code for this new revision control command to the Tcl script file as needed.

Custom Commands

Action	Confirmation	Command TCL
Delete from Repository	☑	exec "p4 delete $FileList"
Perform diff	☑	mydiff $FileList

Add Delete

– To configure a revision control tool of your choice, see *Create New SCM Configuration*, on page 187 in the *Reference* manual.

– Click OK.

For more information about options on the Configure Revision Control Tools dialog box, see *Configure Revision Control Tools Command*, on page 184 in the Reference manual.

4. Do the following in the Project Files view to specify the files for revision control:

   – For revision control on the project file, right-click the file and select Project.

   – For revision control support on all the files in the project, right-click the project file and select All Input Files.

   Alternatively you can also set revision control options from the Project results view by going to the Implementation Directory tab, and selecting File(s).

5. Right-click, go to the Revision Control (Beta) menu, and select the revision control tool command you want to run. The commands vary, according to the tool you are using:

SCM Function	SCM Command
To check in the design files...	Check In
To check out the design files...	Check Out (Perforce only)
To add files to the SCM database...	Add Files to Repository
To retrieve the latest version of files...	Get Latest Revision
To discard changes and reverts...	Revert Changes (Perforce) Discard Changes (CVS)
To provide status for the design files with respect to the SCM database...	Refresh File Info

The synthesis tool uses the revision control tool you configured earlier and runs the commands with the syntax specified in step 3. The Project Files view displays icons next to file to indicate their status. The icons vary with the tool used: :

Status	Perforce Icon	CVS Icon
• For Perforce, file is checked in and locked. • For CVS, file is up-to-date		
File is up-to-date.		N/A
File has been added to the repository, but is not checked in.		
File is modified.	N/A	
Indicates a file merge conflict.	N/A	
File is out-of-sync and requires an update.	N/A	

Copyright © 2013 Synopsys, Inc.

CHAPTER 23

Using Multiprocessing

The following sections describe how to use multiprocessing to run parallel synthesis jobs and improve runtime:

- Multiprocessing With Compile Points, on page 1156
  - Setting Maximum Parallel Jobs, on page 1156
  - License Utilization, on page 1157

Copyright © 2013 Synopsys, Inc.

Multiprocessing With Compile Points

Use the Configure Compile Point Process command to run multiprocessing with compile points. This option allows the synthesis software to run multiple, independent compile point jobs simultaneously, providing additional runtime improvements for the logical compile point synthesis flows. On the Configure Compile Point Process dialog box, specify the maximum number of synthesis jobs you can run in parallel. Note, one license is used for each job. For a description of how to set the maximum number of parallel synthesis jobs, see *Setting Maximum Parallel Jobs*, on page 1156.

To use multiprocessing in the Logical Compile Point Synthesis flows for the Synplify Pro and Synplify Premier tools, see Chapter 21, *Working with Compile Points*. For the Synplify Premier tool, the Physical Synthesis switch must be turned off when you run compile points.

Setting Maximum Parallel Jobs

You can set maximum number of parallel jobs in the following ways:

- INI Variable — MaxParallelJobs
- Tcl Variable — max_parallel_jobs

INI Variable — MaxParallelJobs

The maximum number of parallel jobs is set in the product `ini` file. The following commands are set in the *product*.ini file (for example, synplify_premier_dp.ini):

```
[JobSetting]

MaxParallelJobs=<n>
```

The MaxParallelJobs value is used by the UI as well as in batch mode. This value is effective until you specify a new value. To change the number of parallel jobs you can run, use the Options->Configure Compile Point Process command from the Project view menu. On the Configure Compile Point Process dialog box, in the Maximum number of parallel synthesis jobs field you will see the current `ini` value. You can specify a new MaxParallelJobs value which is effective until you change it again. Once you click OK, the new value is saved in the `ini` file. For a description of the dialog box, see *Configure Compile Point Process Command*, on page 332.

Copyright © 2013 Synopsys, Inc.

Tcl Variable — max_parallel_jobs

You can also manually set an override value for the maximum number of parallel jobs. To do this, use the Tcl command:

set_option -max_parallel_jobs *numberJobs*

You can choose to:

- Source the Tcl file containing this option.
- Add this option to the Project file.
- Set this option from the Tcl command window.

This max_parallel_jobs value is applied to all project files and their respective implementations. This is a global option. The maximum number of parallel jobs remains in effect until you specify a new value. This new value takes affect immediately going forward. However, when you set this option from the Tcl command window, the max_parallel_jobs value is not saved and will be lost when you exit the application.

License Utilization

When you decide to run parallel synthesis jobs, a license is used for each compile point job that runs. For example, if you set the Maximum number of parallel synthesis jobs to 4, then the synthesis tool consumes one license and three additional licenses are utilized to run the parallel jobs if they are available for your computing environment. Licenses are released as jobs complete, and then consumed by new jobs which need to run.

The actual number of licenses utilized depends on the:

1. Synthesis software scheme for the compile point requirements used to determine the maximum number of parallel jobs or licenses a particular design tries to use.
2. Value set on the Configure Compile Point Process dialog box.
3. Number of licenses actually available. You can use Help->Preferred License Selection to check the number of available license. If you need to increase the number of available licenses, you can specify multiple license types. For more information, see *Specifying License Types*, on page 1158.

Copyright © 2013 Synopsys, Inc.

Factors 1 and 3 above can change during a single synthesis run. The number of jobs equals the number of licenses; which then equates the lowest value of these three factors.

Specifying License Types

You can specify multiple license types to increase the total number of licenses available for multiprocessing. To do this, you can either:

- Use the -licensetype command line option when you execute your tool.

  For example, suppose you have two synplifypremier licenses, two synplifypremier_allvendor licenses, and three synplifypremier_xilinx licenses. Type the following at the command line:

  ```
  synplify_premier.exe -licensetype
  "synplifypremier:synplifypremier_allvendor:synplifypremier_xilinx"
  ```

- Use one of the following environment variables specified with the license type:
  - SYNPLIFYPRO_LICENSE_TYPE (Synplify Pro tool)
  - SYNPLIFYPREMIER_LICENSE_TYPE (Synplify Premier and Synplify Premier with Design Planner tools)

  ```
  setenv SYNPLIFYPREMIER_LICENSE_TYPE=
  "synplifypremier:synplifypremier_allvendor:synplifypremier_xilinx"
  ```

Multiprocessing can access any of these license types for additional licenses.

Copyright © 2013 Synopsys, Inc.

CHAPTER 24

Verifying Results with Formality

The increase in the size and complexity of FPGA devices has caused design verification to become more critical in the design process. While simulation and prototyping are still the norm for verifying design functionality, formal verification methods have become invaluable in ironing out implementation issues.

Formal verification uses exhaustive mathematical techniques to verify design functionality. The advantage of formal verification over traditional verification using test vectors is a significant savings in verification time. The accuracy and speed of formal verification has increased its acceptance among designers working with large designs. There are two types of formal verification: equivalence checking and model checking. Equivalence checking uses mathematical techniques to determine if a modified or revised design is logically equivalent to the original design.

This document describes how to use the Synopsys® Formality® equivalence checker to verify results from the Synplify® Premier FPGA tool.

- Overview of the Formality Application, on page 1160
- The Design Flow, on page 1162
- Finite State Machine Design Example, on page 1169
- Tips and Guidelines, on page 1174

Note: The Synplify Premier tool supports this Formality flow with Synplify Premier compiler output only.

Copyright © 2013 Synopsys, Inc.

Overview of the Formality Application

The Formality equivalence checker uses formal techniques to prove and verify the functional equivalence of two designs or two technology libraries. You can use this tool to compare a gate-level netlist to its register transfer level (RTL) source, or to a modified version of that gate-level netlist.

Formality equivalence checking techniques are static and do not require simulation vectors. Consequently, for design verification you only need to provide a functionally correct, or *golden*, design (the reference design) and a modified version of the design (the implementation design). The tool uses mathematical techniques to compare the two and determine whether the modified version is functionally equivalent to the golden design.

The tool methodology fragments the design into compare points. A compare point is a design object used as a combinational logic endpoint during verification. It could be an output port, register, latch, black box input pin, or net driven by multiple drivers. The Formality tool uses the following design objects to create compare points automatically:

- Primary outputs
- Sequential elements
- Black box input pins
- Nets driven by multiple drivers, where at least one driver is a port or black box

The Formality checker verifies a compare point by comparing the logic cone from a compare point in the implementation design against a logic cone for a matching compare point from the reference design, as shown in the following figure:

It tries to match each primary output, sequential element, black box input pin, and qualified net in the implementation design with a comparable design object in the reference design. For the tool to perform a complete verification, all compare points must be verifiable. There must be a one-to-one correspondence between the design objects in the reference and implementation designs.

When functions defining the cones of logic for a matched pair of compare points (one from the reference design and one from the implementation design) are functionally equivalent, the tool reports that they have been successfully verified. If all compare points in the reference design pass verification, the result for the entire design is a successful verification.

Copyright © 2013 Synopsys, Inc.

The Design Flow

The following figure summarizes the design flow between the synthesis tools and the Formality equivalence checker.

The following sections explain the details of the flow and the files:

- Verifying the Design with Formality Equivalence Checking, on page 1162
- Generating VIF Files, on page 1165
- The svf File, on page 1168

Verifying the Design with Formality Equivalence Checking

The following procedure describes the details of the process summarized in the previous figure. For an actual example, see *Finite State Machine Design Example*, on page 1169.

1. Read through the guidelines and set up your design for formal verification.

   See *Tips and Guidelines*, on page 1174 for details.

2. Compile the design by entering project -run compile_vm in the Synplify Premier tool, and generate `vm` and `vif` files as described in *Generating VIF Files*, on page 1165.

3. Convert the `vif` file into an `svf` file (setup verification file for Formality guidance) by doing the following:

   – In the Synplify Premier Tcl window, source the vif2formality Tcl script by typing either of the following commands:

   ```
   source synplify_install_dir/lib/vif2formality.tcl
   ```

   ```
   source $LIB/vif2formality.tcl
   ```

   – Go to the *implementation*/verif/post_compile directory, and type the following command:

   ```
   v2f design.vif
   ```

   The vif2formalityl.tcl script runs on the *design*.vif file and translates the information into an `svf` file for Formality verification. See *The svf File*, on page 1168 for a description of this file.

4. Provide the following inputs for Formality:

RTL	Gate-level post-synthesis netlist (.vs, .vm)
Xilinx library files	For Xilinx designs, do the following: • Download the Xilinx Formality library from the Xilinx website: http://www.xilinx.com/support/download/caesol.htm • Set $XILINX to point to the Xilinx library. For example: setenv XILINX *xilinx_install*/m14_4/ISE • Untar the library to $XILINX/verilog/xeclib.
design.svf	Formality guidance file. See *The svf File*, on page 1168 for more information.
design.tcl	Formality setup file that is generated by the synthesis tool. See *FSM Tcl File Example*, on page 1173 for an example.

Copyright © 2013 Synopsys, Inc.

5. Run the Formality equivalence checker by typing the following in a csh window:

```
formality -file design_compile.tcl | tee result.log
```

The tool fragments the design into compare points and then verifies them by comparing the logic cone from a compare point in the implementation design against a logic cone for the corresponding compare point from the reference design.

If there are syntax errors in the `svf` file, the Formality tool discards the entire file. If there is an interpretation error, it discards that operation and continues the equivalence check.

After it completes, the Formality tool reports one of the following:

Succeeded	Implementation is equivalent to the reference (golden) design.
Failed	Implementation is not equivalent to the reference. This could be because of a logic difference or a setup problem. See *Tips and Guidelines*, on page 1174 for information.
Inconclusive	No compare points failed, but the analysis is incomplete. This could be because of a timeout or the complexity of the design.
Not run	A problem earlier in the flow prevented verification from running.

Copyright © 2013 Synopsys, Inc.

Generating VIF Files

As part of synthesis, the FPGA synthesis tools perform various optimizations and design transformations to improve timing and area. These transformations make it difficult for a formal verification tool to match registers in the output netlist with the corresponding registers in the source HDL. To address the undesired tradeoff between QoR and verifiability, the Synplify Premier tool provides a Tcl file interface that lets you integrate with verification tools. This proprietary format is called the Verification Interface Format or VIF. This feature is currently available for only Xilinx technologies. The following procedure shows you how to specify and generate this file:

1. In the synthesis tool, select Project->Implementation Options and do the following on the Device tab:
   - Set Technology to a Xilinx family that supports the verification flow.
   - Disable optimizations like Retiming and Pipelining on the Options tab and Clock Conversion on the GCC or GCC & Prototyping Tools tab. Disabling these options is recommended because the verification tool requires a one-to-one correspondence between the design objects in the reference and implementation designs to successfully complete, and the optimizations make it hard to verify this. The downside to disabling the optimizations is performance loss.
   - Open the `prj` file and add the Verification Mode (set_option -verification_mode 1) command.

Copyright © 2013 Synopsys, Inc.

This setting makes it easy for the verification tool to synchronize registers. In this mode, synthesis does not perform various optimizations that move registers or optimize them away, because this makes the design hard to verify. The scope of this setting is broader than the Disable Sequential Optimizations option, because Disable Sequential Optimizations only disables sequential optimizations. For a list of the verification mode optimizations, refer to the *FPGA Synthesis Reference Manual*.

2. Go to the Implementation Results tab and do the following:
   - Ensure that Write Mapped Verilog Netlist is enabled. This setting generates the `vif` file. The equivalent project file syntax is set_option -write_verilog 1. When enabled, this option creates a verif directory under the results/implementation directory. The verif directory contains the *design*.vif file.

     Note that this option is automatically set when Verification Mode is enabled.

3. Generate a `vm` file after compilation by adding the following command to the project file:

   ```
   project -run compile_vm
   ```

   The synthesis software generates a `vm` file that reflects the design after compilation but before mapping and stores it in the project/verif/post_compile directory. It also generates a `vif` file in this directory.

4. Go to the verif/post_compile directory and check the `vif` file to see how the optimizations were handled.

   For details of the VIF commands refer to the *FPGA Synthesis Reference* manual.

   You can now convert the `vif` file into an `svf` (setup verification for Formality) guidance file and use it to run equivalence checking, as described in *Verifying the Design with Formality Equivalence Checking*, on page 1162.

Copyright © 2013 Synopsys, Inc.

The svf File

The Setup Verification for Formality (svf) script is a file that contains Formality commands. Use the vif2formality Tcl script to convert a FPGA synthesis vif file into an svf file.

The svf commands provide guidance so that the Formality tool can account for the transformations made by the synthesis optimizations. For example, the synthesis tool might duplicate a register to improve drive strength, and this file records the register duplication. When the Formality tool reads the svf, it can account for the extra register during compare point matching and verification. See *FSM svf File Example*, on page 1173 for an example of this file. The following example shows some file syntax that informs the Formality tool that X-reg and Y_reg are tied to 1'b0 and 1'b1 in the netlist.

```
guide_reg_constant -design vcr {X_reg} 0
guide_reg_constant -design vcr {Y_reg[0]} 1
```

The svf file supports the following:

- Guide_reg_merging
- Guide_reg_duplication
- Guide_reg_constant
- Guide_FSM_reencoding

Finite State Machine Design Example

This section runs through the flow using a finite state machine (FSM) design as an example.

- Verifying a Finite State Machine Design Example, on page 1169
- RTL Code Example for a Finite State Machine, on page 1170
- FSM Tcl File Example, on page 1173
- FSM svf File Example, on page 1173

Verifying a Finite State Machine Design Example

The following procedure takes a specific example, a finite state machine, through the steps for equivalence checking. See *RTL Code Example for a Finite State Machine*, on page 1170 for the example code.

1. Compile the code shown in *RTL Code Example for a Finite State Machine*, on page 1170 in Synplify Premier tool, going through the steps described in *Verifying the Design with Formality Equivalence Checking*, on page 1162 and generating the `vif` file as described.

2. In the Tcl window of the synthesis tool, run the vif2formality.tcl script as described in *Verifying the Design with Formality Equivalence Checking*, on page 1162.

   This generates the *design*.tcl and *design*.svf files in the verif directory. These files are also reproduced in *FSM Tcl File Example*, on page 1173 and *FSM svf File Example*, on page 1173.

3. Verify the design by doing the following:
   - Start the Formality tool.
   - Select File->Run Script and run the `tcl` file generated in the previous step.

   For successful verification, the Formality report should not have any unmatched points. With this example, you see that there are two registers in the reference design that are unmatched.

Copyright © 2013 Synopsys, Inc.

4. Set these registers as constants by setting the following commands in the `svf` file:

```
guide_reg_constant -design vcr {forward_tape_reg} 0
guide_reg_constant -design vcr {current_state_reg[0]} 1
```

5. Select File->Run Script and run the `tcl` file again to verify the design.

   The design no longer contains any unverified points or aborted points, and passes the equivalence check successfully.

RTL Code Example for a Finite State Machine

This is the RTL code for the FSM example used in *Verifying a Finite State Machine Design Example*, on page 1169.

```
// Copyright (c) 2010 by Synopsys, Inc.
// You may distribute freely, as long as this header
// remains attached.
// VCR tape player and recorder state machine 10 States

`define stop            4'h0
`define will_forward    4'h1
`define forward         4'h2x
`define will_rewind     4'h3
`define rewind          4'h4
```

```
`define pause           4'h5
`define will_play       4'h6
`define play            4'h7
`define will_record     4'h8
`define record          4'h9

module vcr(stop_tape, pause_tape,
   forward_tape, rewind_tape,
   play_tape, record_tape,
   clk,
   stop_button, pause_button,
   forward_button, rewind_button,
   play_button,record_button,
   is_stopped, reset);

output stop_tape, pause_tape, forward_tape, rewind_tape,
   play_tape, record_tape;
input clk, stop_button, pause_button, forward_button,
rewind_button,
   play_button, record_button, is_stopped, reset;

reg stop_tape, pause_tape, forward_tape, rewind_tape,
   play_tape, record_tape;
reg [3:0] current_state, next_state; reg
[4:0] get_next_state;

always @ (posedge clk)
begin:  this_always
   reg state_independent;
   stop_tape = 0;
   pause_tape = 0;
   play_tape = 0;
   record_tape = 0;
   forward_tape = 0;
   rewind_tape = 0;

// SET OUTPUTS
case(current_state)
   `stop,`will_play,`will_record,
   `will_forward,`will_rewind: stop_tape = 1;
   `pause: pause_tape = 1;
   `play: play_tape = 1;
   `record: record_tape = 1;
   `forward: forward_tape = 1;
   `rewind: rewind_tape = 1;
endcase
```


Copyright © 2013 Synopsys, Inc.

```
// STATE TRANSITIONS
if (!reset)   // synchronous reset

next_state = `stop;
else begin
   // go to next state
   // state independent transitions
   get_next_state = {1'b1, current_state};
   if (stop_button)
      get_next_state = `stop;
   else if (record_button && play_button)
      get_next_state  = `will_record;
   else if (play_button)
      get_next_state  = `will_play;
   else if (forward_button)
      get_next_state  = `will_forward;
   else if (rewind_button)
      get_next_state  = `will_rewind;
   else
   // next state is state-dependent and we are not setting it here
      get_next_state  = {1'b0, current_state};
      {state_independent, next_state} =get_next_state;
   // do state-dependent transitions
      if (!state_independent)
         if (is_stopped)
            case(current_state)
               `will_forward:  next_state = `forward;
               `will_rewind:   next_state = `rewind;
               `will_play:     next_state = `play;
               `will_record:   next_state = `record;

         endcase
      else if (current_state == `play  && pause_button)
         next_state = `pause;
      else if (current_state == `pause && pause_button)
      next_state = `play;
end // outer if

current_state = next_state;

end  // always block
endmodule
```

FSM Tcl File Example

This is the Tcl file generated as a result of running the FSM design described in *Verifying a Finite State Machine Design Example*, on page 1169.

```
set synopsys_auto_setup true
set_svf statmch2.svf
set XILINX [get_unix_variable "XILINX"]
set hdlin_library_directory $XILINX/verilog/xeclib/unisims

read_verilog -r -vcs "-y $XILINX/verilog/xeclib/unisims -y
$XILINX/verilog/xeclib/simprims +libext+.v" -01 "../statmch2.v "
set_top vcr
read_verilog -i -vcs "-y $XILINX/verilog/xeclib/unisims -y
$XILINX/verilog/xeclib/simprims +libext+.v" -01 " ../statmch2.vm "
set_top vcr

current_design $ref
ungroup -all -flatten
#set_parameters -retimed $ref
set signature_analysis_match_compare_points false
match
set signature_analysis_match_compare_points true
match
verify
```

FSM svf File Example

This is the Tcl file generated as a result of synthesizing the FSM design described in *Verifying a Finite State Machine Design Example*, on page 1169.

```
# Active SVF file statmch2.svf
guide
guide_environment \
{ { bus_dimension_separator_style ][ } \
{ bus_extraction_style %s\[%d:%d\] } \
{ bus_naming_style %s[%d] } \
}

guide_fsm_reencoding -design { vcr } \
-previous_state_vector {current_state_reg[3] \
current_state_reg[2] \
   current_state_reg[1] current_state_reg[0] } \
-current_state_vector { current_state[8] current_state[7] \
current_state[6]| \
   current_state[5] current_state[4] current_state[3] \
```


Copyright © 2013 Synopsys, Inc.

```
current_state[2]\
   current_state[1] current_state[0] } \
-state_reencoding { \
{ S0 2#0000 2#000000001 } \
{ S1 2#0001 2#000000010 } \
{ S2 2#0011 2#000000100 } \
{ S3 2#0100 2#000001000 } \
{ S4 2#0101 2#000010000 } \
{ S5 2#0110 2#000100000 } \
{ S6 2#0111 2#001000000 } \
{ S7 2#1000 2#010000000 } \
{ S8 2#1001 2#100000000 } \ }

guide_reg_constant -design vcr {forward_tape_reg} 0
guide_reg_constant -design vcr {current_state_reg[0]} 1
setup
```

Tips and Guidelines

See the following for additional information on getting the best out of the Formality equivalence checking flow:

- Guidelines for Successful Verification, on page 1174
- Writing RTL for Verification used in ASIC Prototyping, on page 1175
- Limitations to Verification with Formality, on page 1178
- Verification Mode Dependencies, on page 1179

Guidelines for Successful Verification

The following tips provide some guidelines for successfully checking your design with the Formality tool:

- Map all key points correctly and completely before beginning formal verification. Unmapped points result in non-equivalences during the comparison phase of formal verification.
- Generally, non-equivalent points are the result of missing or incorrect mapping rules. Evaluate any non-equivalent points carefully to determine the cause. In most cases, fixing annotations resolves the problem.

Search for errors and warnings in the formal verification tool log file and resolve them.

- Although the final goal is a top-down verification run with no non-equivalent points, it is recommended that you begin verification at the sub-module level and work up to the top level. Small issues at the sub-module level can translate into a large number of unmapped or non-equivalent points at the top level. There is a smaller amount of design logic at the sub-module level, so it is much easier to deal with and resolve annotation issues at this level.
- When extracting a sub-module from an entire design for verification, preserve the sub-module interface as it is in the RTL. Synthesis tools perform optimizations across hierarchies and can alter module level interfaces. For module-level verification, you can either apply synthesis attributes to the module to preserve the interface, or synthesize the module by itself.

Writing RTL for Verification used in ASIC Prototyping

If you are targeting an FPGA to prototype an ASIC design, follow these guidelines to help the verification process run smoothly:

- The Synplify synthesis tools ignore simulation-specific constructs like $DISPLAY and $SETUPHOLD but the Formality checker does not. Enclose these constructs in the //synthesis translate_off/on pragmas.
- Always size datatypes. For example, use 1`b0 instead of`b0.
- Do not use duplicate modules or entities in the same project. The synthesis tools permits the allow multiple modules option, which allows the tool to select the last read definition when it encounters multiple contents for the same module or entity. This option causes ambiguity in the Formality checker and is not permitted. Make sure each project has only one definition of each module or entity.
- When using custom packages, for operators like + and -, provide the function definition as part of the project. For example, include the package definition file in the project if you use a non-standard package. Assume you use the following non-standard library:

```
library std_developerskit;
User std_developerskit.synth_regpak.all;
```


Copyright © 2013 Synopsys, Inc.

You must then add the following file to the prj: vhdl_src/std_developerskit/synthreg.vhd.

- Use synopsys full_case and synopsys parallel_case with caution. If you specify full_case and do not define all the cases, it can cause simulation issues because simulators do not honor this directive. The board behavior is not affected.

  Use pragmas and synthesis directives with caution, because they direct the synthesis tools to remove logic based on these directives. As these directives are typically specified by the designer, it is assumed that the board behavior will be correct. However, simulators do not read these directives and do not do any reachability analysis.

- Avoid using multiple drivers, where a net is driven both by a constant and an active signal. The synthesis tools now resolve this to a constant, but a simulator might not do the same. The Formality tool treats this as a wired AND.

  Up until the 2010.09 release, the synthesis tools errored out by default in these situations. Although it is not recommended, you can enable the Resolve Mixed Drivers option to revert to the old behavior. See the *FPGA Reference Manual* for details about this option.

- Gated clock structures

  By default, the synthesis tools convert gated clock structures for optimal performance. The conversion is needed for the prototype to work on the board but poses issues for verification and cannot be easily verified. As gated clocks are used extensively in ASICs that are implemented in FPGAs, you can help the verification process by doing the following:

  – Identify and constrain the correct clocks in the Synplify design.

  – Put the gated clock logic in its own hierarchy wherever possible, so it can be isolated during verification.

- RTL RAMs

  – RAMs are typically implemented as block RAMs in FPGAs, and these RAMs do not have good verification models. Enclose any RTL code that the synthesis tools infer as a RAM in a separate hierarchy. This is to enable black boxing of the RAMs and the continued verification of the rest of the design. Put the RAM code in a separate design hierarchy with a fixed boundary (syn_hier =fixed).

Copyright © 2013 Synopsys, Inc.

– If the code implies the use of register banks instead of a RAM implementation, use the syn_ramstyle="register" attribute to force the synthesis tool to implement it as registers, which can be formally verified.

- Complex SystemVerilog data types

  The synthesis tools do not generate guidance information for formal verification for SystemVerilog RTL with complex port types, like the use of structs or multi-dimensional arrays. Consider this example of complex coding:

```
module test (
sig1,
sig2, …

input type1 sig1;
typedef struct packed {
   logic ns;
   .
   .
   .
   logic [4:0] typ;
   logic [1:0] fmt; }
type1;
```

  After synthesis, the sig1 struct type port is changed to `input [90:0] sig1;` and the Formality tool cannot match the original port name to the synthesized port name.

```
sig1[typ][0] ------> sig1 [2]
sig1[fmt][1] ------> sig1 [1]
sig1[fmt][3] ------> sig1 [3]
```

  Try to avoid using such structures. If you must use them, you must manually create a guide file before running Formality as follows:

```
Set_user_match -type port r:/WORK/test/sig1\[fmt][1]
   i:/WORK/test/sig1[1]
```

  As a workaround user can choose to hand type the guidance information but could be very time consuming.

- Multipliers

  Small multiplier structures (<13 bits) that are implemented as logic can be verified formally. For larger structures, enclose them in a separate

Copyright © 2013 Synopsys, Inc.

hierarchy so that it can be black-boxed. Verify the black blocks by other means like simulation.

- DSP blocks

  Logic inferred as DSP blocks are difficult to verify because pipelined registers from surrounding logic to go into the DSP block. The current workaround is to direct the tool to implement the DSP blocks as logic using the syn_dspstyle attribute, or to enclose the DSP structure in its own hierarchy so it can be black-boxed. Verify these black blocks by some other means like simulation.

- Retiming

  Turn off retiming option whenever possible. Wherever possible, turn it off on specific blocks, not globally.

- syn_hier="fixed"

  When synthesizing, it helps to use the syn_hier="fixed" attribute on all levels of hierarchy. Add simple collection command like the following to the Synplify `sdc` file:

  ```
  define_scope_collection all_views {find -hier -view {*}}
  define_attribute {$all_views} {syn_hier} {fixed}
  ```

- Registers with set and reset pins

  Black-box any blocks containing RTL code that describes registers with both set and reset pins. The Xilinx tool does not have a single equivalent primitive that has both pins. It uses two registers, which causes problems for the Formality checker.

Limitations to Verification with Formality

Currently, there are some limits to formal verification with the Formality tool:

- In FSM designs, the FSM state register may be merged or duplicated in synthesis. This makes it hard to generate the guide information for Formality.
- Formality can only verify RAM as flip-flops. The tool treats block RAMs, DSPs, and SRLs as black boxes.

Copyright © 2013 Synopsys, Inc.

- For Xilinx RAMs you can work around the limited functionality by setting RAMs to flip-flops or select RAM. When RAM is inferred as block RAM, set black_box methodology basically ignores the RAM verification.
- For multipliers, the tool can only verify up to 13-bit multipliers that are implemented as logic. It does not handle multipliers that are implemented in macros, like DSPs.

Verification Mode Dependencies

Here are dependencies to consider when using Verification Mode:

- Verification Mode is only supported in the Synplify Premier tool.
- Verification Mode is not the same as Disable Sequential Optimizations. Verification Mode disables many FPGA optimizations that includes sequential optimizations. Disable Sequential Optimizations only disables advanced sequential optimizations.
- Enabling Verification Mode results in a QoR trade-off.
- Verification Mode honors attributes like syn_hier=hard and syn_keep=1.
- During Verification Mode, optimizations like gated clock conversion and generated clock conversion, pipelining, and retiming should be turned off. It is recommended that you disable these optimizations because they might make it hard for the Formality tool to find matching compare points.

Copyright © 2013 Synopsys, Inc.

Index

Symbols

_conv.prj file 207
_conv.sdc file 207
.acf file 126
.adc file 734
.cdc file
  specifying attributes and directives 248
.edf file 801
.ini file
  parallel jobs 1140
.lpf file 126, 785
.ndf file 801
.vhm file 826
.vqm file
  Clearbox 1012

Numerics

3rd party vendor tools
  invoking 836

A

ACTgen macros 790
adc constraints 734
adc file
  creating 734
  object names 738
adc file, using 732
adders
  SYNCore 957
  wide. *See* wide adders/subtractors.
adjust pin view (Design Planner) 616
After 282
alspin
  bus port pin numbers 831
Alt key
  column editing 112
  mapping 585
Altera
  Clearbox. *See* Clearbox
  converting pin assignments to SDC 321
  converting PIN files 321
  design tips 760
  forward-annotation 126
  grey box *See* grey box
  I/O packing 763
  instantiating LPMs as black boxes 412
  LPM megafunction example (Verilog) 412
  LPM megafunction example (VHDL) 414
  multi-port RAMs 382
  netlist 834
  P&R file for untranslated settings 1036
  packing I/Os 763
  physical synthesis flows 55
  place-and-route option file 308
  PLLs. *See* altplls
  Quartus batch mode 771
  Quartus integrated flow 769
  Quartus interactive flow 770
  shift registers 405
  simulating LPMs 767
  Stratix III LUTRAMs 383
  Stratix RAM 376
  Verilog LPM library 418
Altera incremental flows 846
Altera MegaWizard
  generating LPM files 412
Altera shift registers
  report 407
Altera STRATIX (Design Planner)
  additional tips 641
altpll
  component declaration files 761
  constraints 761
  using 760
altshift_tap, set implmentation style 405

Copyright © 2013 Synopsys, Inc.

ALTSYNCRAM for LPMs 412
ALTSYNCRAM, Altera Stratix 376
analysis design constraint file (.adc) 734
analysis design constraints
  design scenarios 733
analysis design constraints (adc) 732
analysis design constraints (adc), using with sdc 734
analyzing netlists (Physical Analyst) 703
archive utility
  using 287
archiving projects 287
area estimation, Design Planner 612
area, optimizing 423
asterisk wildcard
  Find command 573
attr_applied.sdc file 1036
attr_unapplied.sdc file 1036
attributes
  adding 245
  adding in constraint files 123
  adding in SCOPE 250
  adding in Verilog 247
  adding in VHDL 246
  collections 164
  effects of retiming 434
  for FSMs 343, 459
  pipelining 428
  syn_hier (on compile points) 1121
  VHDL package 246
attributes in .cdc file 248
Attributes panel
  using SCOPE 136
audience for the document 29
auto constraints, using 739
Auto route cross probe insts command 695
AutoConstraint_design_name.sdc 742
automatic compile points
  compared to manual 1090
  flow 1109
  using with manual 1123

B

B.E.S.T 589
back annotation
  coreloc.sdc constraint file 326
  place-and-route data 326
backslash
  escaping dot wildcard in Find command 573
  in Find command (Physical Analyst) 688
batch mode 1132
  using find and expand 159
Behavior Extracting Synthesis Technology. *See* B.E.S.T
bit slicing 652
  legal primitives 653
black boxes 330
  adding constraints 334
  adding constraints in SCOPE 337
  adding constraints in Verilog 336
  adding constraints in VHDL 335
  EDIF naming consistency 339
  for IP cores 801
  gated clock attributes 489
  instantiating in Verilog 330
  instantiating in VHDL 332
  internal startup blocks 339
  passing VHDL boolean generics 120
  passing VHDL integer generics 121
  pin attributes 338
  prepared component method (Altera) 417
  setting in hierarchical projects 279
  specifying timing information for Xilinx cores 801
black_box compile point 1096
Block Inputs Map
  Physical Analyst 716
block RAM
  dual-port, mapping with registered address 386
  glue logic in 387
  mapping dual port coding style 392
  mapping ROM (Xilinx) 392
  mapping to single-output dual-port 392
  mapping to single-port 389

single-port
mapping with registered address 385
using registered addresses 385
using registered output 387

Block Utilization Map
Physical Analyst 715

block-first hierarchical development flow 88

blocking-style license queuing 1134

blocks
defining for hierarchical projects 254

bookmarks
in source files 112
using in log files 517

bottom-up design flow
compile point advantages 1088

bottom-up hierarchical synthesis flow 91

breaking up large primitives (Synplify Premier) 652

browsers 560

buffering
crontrolling 446

BUFG
clock priority (Legacy) 198
for fanouts 447

BUFGCTRL
false paths (Legacy) 199

BUFGDLL 820

BUFGMUX
clock priority (Legacy) 198

BUFGMUX_1 inference 815

BUFGMUX_CTRL
false paths (Legacy) 198

BUFGMUX/BUFGMUX_1 inference 815

BUFR clock buffers 816

buses
INIT values for bits 808
RLOC values for bits 820

byte-enable RAMs
SYNCore 946

C

c_diff command, examples 166

c_intersect command, examples 166

c_list command
different from c_print 168
example 170
using 170

c_print command
different from c_list 168
using 169

c_symdiff command, examples 167

c_union command, examples 166

callback functions, customizing flow 1146

carry chains
inferring 775

case sensitivity
Find command (Tcl) 153

cdc file syntax 249

cells
enhancing display in Physical Analyst 673

chip regions (Design Planner) 634

Clearbox
adding instantiated file 1011
implementing megafunctions with 1002
inferring megafunctions 1003
instantiating megafunctions 1007
instantiating with netlist 1010
using 1002
using instantiated netlists in Quartus 1012

clock and path constraints
setting 137

clock buffers 820

clock constraints
setting 137
setting (Legacy) 178

clock conversion report
accessing 476
analyzing 477

clock DLLs 820

clock groups
effect on false path constraints 151

clock path skew (Synplify Premier) 537

clock pins (Design Planner) 622

Copyright © 2013 Synopsys, Inc.

clock skew (Synplify Premier) 537
clock skew example (Synplify Premier) 822, 823
clock trees 725
clocks
  implicit false path 151
Clocks panel
  using SCOPE 135
collections
  adding attributes to 164
  adding objects 165
  concatenating 165
  constraints 163
  copying 169
  creating from common objects 165
  creating from other collections 163
  creating in SCOPE 162
  creating in Tcl 164
  crossprobing objects 163
  definition 161
  diffing 165
  highlighting in HDL Analyst views 168
  listing objects 169
  listing objects and properties 168
  listing objects in a file 169
  listing objects in columnar format 168
  listing objects with c_list 168
  special characters 167
  Tcl window and SCOPE comparison 161
  using Tcl expand command 157
  using Tcl find command 156
  viewing 167
Collections panel
  using SCOPE 135
column editing 112
combination hierarchical synthesis flow 95
commands
  Auto route cross probe insts (Physical Analyst) 695
  Go to Location (Physical Analyst) 690
  Highlight Visible Net Instances (Physical Analyst) 707
  Markers 691
  netlist editing 891
  Select Net Instances 707
  Send Crossprobes when selecting (Physical Analyst) 695
  Signal Flow 678
  slice_primitive 653
comments
  source files 112
compilation process 531
compile point types
  black_box 1096
  hard 1093
  locked 1094
  locked,partition 1096
compile points
  advantages 1088
  Altera incremental flows 846
  analyzing results 1121
  automatic compile point flow 1109
  automatic timing budgeting 1101
  child 1091
  constraint files 1098
  constraints for forward-annotation 1109
  constraints, internal 1108
  creating constraint file 1119
  defined 1088
  defining in constraint files 1117
  described 1090
  fast synthesis 1124
  feature summary 1097
  Identify flow 887
  incremental synthesis 1126
  manual compile point flow 1113
  multiprocessing 1125
  nested 1091
  optimization 1105
  order of synthesis 1105
  parent 1091
  preserving with syn_hier 1121
  Quartus II Incremental Compilation 849
  resynthesis 1107
  setting constraints 1120
  setting type 1118
  syn_hier 1121
  synthesis process 1104
  synthesizing 1109
  types 1092
  using automatic and manual compile points together 1123
  using syn_allowed_resources attribute 1121
  Xilinx incremental flows 869

compile points and hierachical project management 86
Compile Points panel
using SCOPE 136
compile-point flow
Xilinx 873
compile-point synthesis
interface logic models 1100
compile-point synthesis flow
defining compile points 1117
setting constraints 1119
compiler directives 102
using New Constraint File 104
compiler directives (Verilog)
specifying 240
compiler directives syntax 105
Conformal 897
congestion
using estimation report 538
congestion analysis after logic synthesis using 538
congestion analysis report 538
congestion map
controls 712
constants
extracting from VHDL source code 243
constraint file
coreloc.sdc 328
constraint files
Altera I/O constraints 193
applying to a collection 163
compile point 1098, 1109
creating in a text editor 123
defining clocks (Legacy) 173
defining register delays (Legacy) 174
editing 143
effects of retiming 434
forward-annotating 125
options 235
setting for compile points 1120
specifying through points 147
type of constraints 135
vendor-specific 125
constraint files (Legacy)
types of 175
constraints
altplls 761
translating Altera I/O constraints 193
translating with ise2syn 1058
translating xdc to fdc 190
translating Xilinx contraints for logic synthesis 201
translating Xilinx contraints for physical synthesis 203
Vivado flow 854
context
for object in filtered view 592
context help editor 108
SystemVerilog 108
context window (Physical Analyst) 669
continue on error 233, 531
enabling 532
reporting 533
control panel
Physical Analyst view 666
CoreGen 801
coreloc.sdc 328
coreloc.sdc file 326
cores, instantiating in Xilinx designs 801
counters
SYNCore 964
critical paths
delay 727
flat view 726
hierarchical view 726
negative slack on clock enables (Legacy) 184
slack time 727
using -route 425
viewing 725
critical paths (Design Planner)
assigning to regions 638
critical paths (Physical Analyst)
tracing backward 757
tracing forward 755
crossprobing 581
and retiming 434
collection objects 163
filtering text objects for 586
from FSM viewer 588
from log file 517

Copyright © 2013 Synopsys, Inc.

from message viewer 523
from text files 584
from text files to Physical Analyst 698
Hierarchy Browser 581
importance of encoding style 588
paths 585
Physical Analyst view 695
RTL view 582
Technology view 582
Technology view to Physical Analyst 701
Text Editor view 582
text file example 585
to FSM Viewer 587
to place-and-route file 557
Verilog file 582
VHDL file 582
View Cross Probing commands 695
within RTL and Technology views 581

crossprobing (Physical Analyst)
auto route crossprobing 701
RTL view 695, 699
Technology view 695

crossprobing commands (Synplify Premier)
Physical Analyst view 695

current level
expanding logic from net 596
expanding logic from pin 596
searching current level and below 570

custom folders
creating 220
hierarchy management 220

customization
callback functions 1146

D

data block 973

data key 973

default enum encoding 242

define_attribute 253

Delay Paths panel
using SCOPE 136

design flow
customizing with callback functions 1146

design flows
hierarchical project management 86
team design 86

design guidelines 422

design hierarchy
viewing 590

design plan
options 236

Design Plan Editor view
preserving region resources 633

design plan file
creating 618
logic synthesis 44
physical synthesis 52, 618

Design Planner 612
assigning pins 619
creating chip regions 634
displaying IP core areas 634
guidelines 611
logic synthesis 44
opening 612
physical synthesis 52
working with SSI devices 645

design planning 612

design size
amount displayed on a sheet 557

design views
moving between views 556

DesignWare
building blocks 988
DW_Foundation_Arith package 990
foundation library 988
importing cores 992
minPower library 989
multiprocessing licenses 989

device options
See also implementation options

directives
adding 245
adding in Verilog 247
adding in VHDL 246
black box 335, 336
for FSMs 343
specifying for Verilog compiler 240
syn_state_machine 457
syn_tco 336
adding black box constraints 335

syn_tpd 336
adding black box constraints 335
syn_tsu 336
adding black box constraints 335
directives in .cdc file 248
dissolving instances for flattening hierarchy 603
distributed TMR 346
example 348
DLLs
defining clocks (Legacy) 183
dot wildcard
Find command 573
drivers
preserving duplicates with syn_keep 438
selecting 599
dual-port RAMs 382
block RAMs with single registered output, Xilinx 392
Stratix 377
SYNCore parameters 943
DW_Foundation_Arith package 990

E

ECC RAM 357
EDIF
structural, for Xilinx IP cores 801
Editing window 111
editor
compiler directives 102
editor view
context help 108
EDK
specifying cores as white boxes 1059
EDN core 1046
emacs text editor 116
encoding styles
and crossprobing 588
default VHDL 242
FSM Compiler 456
encrypted IP objects (Physical Analyst)
identifying cells 693
encryption
synenc 992
encryption flow. *See* ReadyIP, encryptIP
encryptip output constraints 981
encryptip output method
effect on output netlists 981
encryptIP script
controlling output 979
encrypting IP 978
output methods 979
enhanced optimization
compared to fast synthesis 39
using 40
environment variables
PAR_BELDLYRPT 304
SYN_TCL_HOOKS 1146
equivalence checking
VIF file 893
equivalency checking
handling failure 898
error codes 1132
error correction code. *See* ECC 357
error messages
gated clock report 479
error mitigation. *See* high reliability
errors
black boxing 532
continuing 233, 531
definition 111
filtering 522
sorting 522
source files 110
Verilog 110
VHDL 110
expand
batch mode 159
Expand command
connection logic 599
connections in Physical Analyst 709
pin and net logic 595
using 596
expand command
different from Tcl search 576
Expand command (Physical Analyst) 704
expand command (Tcl). *See* Tcl expand command
Expand Inwards command
using 596

Copyright © 2013 Synopsys, Inc.

Expand Path Backward command 757
Expand Path Forward command 755
Expand Paths command
  different from Isolate Paths 599
expand pin view (Design Planner) 615
Expand to Register/Port command
  using 596
Expand to Register/Port command
  (Physical Analyst) 704
expanding
  connections 599
  connections (Physical Analyst) 709
  pin and net logic 595
  pin and net logic (Physical Analyst) 704

F

false paths
  defining between clocks (Legacy) 189
  I/O paths 151
  impact of clock group assignments 151
  impact of clock group assignments
    (Legacy) 188
  ports 151
  ports (Legacy) 188
  registers 151
  registers (Legacy) 188
  setting constraints 151
  setting constraints (Legacy) 188
fanout
  replicating instances (Design
    Planner) 638
fanouts
  buffering vs replication 446
  hard limits 445
  soft global limit 444
  soft module-level limit 445
  using syn_keep for replicaton 439
  using syn_maxfan 444
fast synthesis
  compile points 1124
  using 1062, 1063
fault tolerance. *See* high reliability
fdc
  converting xdc constraints 190
feature comparison
  FPGA tools 26
FIFOs
  compiling with SYNCore 934
files
  .acf 126
  .lpf 126, 785
  .prf file 524
  altpll component declarations 761
  dependent list 269
  filtered messages 526
  fsm.info 457
  log 514
  message filter (prf) 524
  output 834
  rom.info 563
  searching 284
  statemachine.info 609
  synhooks.tcl 1146
  Tcl 1139
    See also Tcl commands
  Tcl batch script 1133
Filter Schematic command, using 593
Filter Schematic icon, using 593
filtering 593
  advantages over flattening 593
  using to restrict search 570
filtering (Physical Analyst) 703
Find command
  570
  browsing with 569
  hierarchical search 571
  long names 569
  message viewer 522
  Physical Analyst view 686
  reading long names 572
  search scope, effect of 573
  search scope, setting 571
  searching the mapped database 572
  searching the output netlist 578
  setting limit for results 572
  using in RTL and Technology views 570
  using wildcards 573
  wildcard examples 575
find command
  different from Tcl search 576
  hierarchy 576
  nuances and differences 577
Find command (Physical Analyst)
  using Filter Search option 689

using wildcards 688
Find command (Tcl)
See also Tcl find command
finding information
information organization 30
finding objects
Physical Analyst view 686
Fix Gated Clocks option. *See* gated clocks
Flatten Current Schematic command
transparent instances 601
using 601
Flatten Schematic command
using 601
flattening 600
See also dissolving
compared to filtering 593
dissolving instances 603
hidden instances 602
transparent instances 601
using syn_hier 442
using syn_netlist_hierarchy 442
floorplan file. *See* sfp file, design plan file
floorplan. *See* Design Planner
formal verification
Formality 1159
Formality
using 1159
forward annotation
frequency constraints in Xilinx 795
vendor-specific constraint files 125
forward-annotation
compile point constraints 1108
constraints 784
Xilinx core files 1048
foundation library 988
FPGA Design Constraints Editor
using TCL View 141
frequency
clocks (Legacy) 181
defining for non-clock signals (Legacy) 182
internal clocks (Legacy) 181
setting global 234
from constraints
specifying 146
FSM Compiler
advantages 454
enabling 456
FSM encoding
user-defined 344
using syn_enum_encoding 344
FSM Explorer 454
running 460
when to use 454
FSM view
crossprobing from source file 584
FSM Viewer 606
crossprobing 588
fsm.info file 457
FSMs
See also FSM Compiler, FSM Explorer
attributes and directives 343
defining in Verilog 340
defining in VHDL 341
definition 340
Hamming3 360
optimizing with FSM Compiler 454
properties 609
safe. *See* safe FSMs
single-bit errors 361
state encodings 608
transition diagram 606
viewing 606

G

gated clocks
attributes for black boxes 489
conversion example 469
conversion requirements 468
defining (Legacy) 186
error messages 479
examples 467
procedure for fixing 475
restrictions 494
gated-clock conversion
excluding elements 482
Generated Clocks panel
using SCOPE 135
generated-clock conversion 499
generics
extracting from VHDL source code 243
passing boolean 120

Copyright © 2013 Synopsys, Inc.

passing integer 121

global comments
initializing Xilinx RAM 400

global optimization options 231

global sets/resets
Xilinx designs 798

glue logic
Altera Stratix RAM 376

Go to Location command 690
adding markers 692

graph-based physical synthesis
Altera 55
description 46
logic synthesis validation phase 47
physical synthesis phase 47

grey box
netlist file 1016

grey box flow
MegaCore with greybox netlist 1016

grey boxes
using 1012

greybox flow
MegaCore with IP package 1019
NIOS II cores 1023
SOPC cores 1023

GSR resources 774

GSR, Xilinx 797

H

Hamming Distance 3 360

HAPS technology selection 229

HDL Analyst
See also RTL view, Technology view
critical paths 725
crossprobing 581
filtering schematics 593
Push/Pop mode 563, 566
quick load option 234
traversing hierarchy with mouse strokes 561
traversing hierarchy with Push/Pop mode 563
using 589

HDL Analyst Quick Load option 234

HDL Analyst tool
deselecting objects 554
selecting/deselecting objects 553

HDL Analyst views
highlighting collections 168

HDL views, annotating timing information 723

help
information organization 30

hidden instances
consequences of saving 591
flattening 602
restricting search by hiding 570
specifying 591
status in other views 591

hierarchical design
expanding logic from nets 596
expanding logic from pins 595

hierarchical instances
dissolving 603
hiding. *See* hidden instances, Hide Instances command
multiple sheets for internal logic 592
pin name display 594
viewing internal logic 591

hierarchical objects
pushing into with mouse stroke 562
traversing with Push/Pop mode 563

hierarchical project management and compile points 86

hierarchical project management flows
bottom-up development flow 88
bottom-up synthesis flow 91
top-down development flow 89
top-down synthesis flow 93

hierarchical project managment flows
mixed block synthesis flow 95

hierarchical projects 86
analyzing 282
configuring for synthesis 278
instance-based synthesis 257
multiple implementations 270
synthesis options 278

hierarchical search 570

hierarchy
flattening 601
netlist restructuring 303

Copyright © 2013 Synopsys, Inc.

traversing 560

hierarchy browser
clock trees 725
controlling display 557
crossprobing from 581
defined 560
finding objects 568
traversing hierarchy 560

hierarchy management (custom folders) 220

high reliability
distributed TMR 346
ECC RAM 357
FSMs with Hamming3 360
local TMR 356
safe FSMs 359
using safe FSM 359
using TMR 346

high reliability design 345

Highlight Visible Net Instances command 707

hyper source
example 995
for IPs 993
for prototyping 993
IP design hierarchy 993
threading signals 994

I

I/O insertion 452, 783
VHDL manual (Xilinx) 814

I/O locations
assigning automatically (Xilinx) 809
manually assigning (Xilinx) 814

I/O pads
specifying I/O standards 140

I/O paths
false path constraint 151

I/O standards
specifying 140

I/O Standards panel
using SCOPE 136

I/Os
auto-constraining 740
constraining 139
constraining (Legacy) 187
packing in Altera designs 763
packing in Xilinx designs 804
preserving 453, 784
specifying pad type (Xilinx) 823
Verilog black boxes 330
VHDL black boxes 332

I/Os (Design Planner)
critical paths from pin-locked I/Os 638

IBUFDS
inference 816

IBUFGDS
inference 816

Identify
compile points 887

Implementation Maps
Physical Analsyt 710

implementation options 228
design plan file 236
device 228
global frequency 234
global optimization 231
netlist optimizations 302
part selection 228
specifying results 237

implementations
copying 227
deleting 227
multiple. *See* multiple implementations.
overwriting 227
renaming 227

incremental flows
Quartus Fast Fit 847
SmartGuide 870
Vivado 864
Xilinx partion (before ISE 12.1) 878
Xilinx partition 873

incremental synthesis
compile points 1126
locked,partition compile points 1096
other tools 1129

inference
BUFGMUX/BUFGMUX_1 815
Xilinx BUFGMUX/BUFGMUX_1 815
Xilinx I/O buffers 815

INIT property
initializing Xilinx RAMs, Verilog 399
initializing Xilinx RAMs, VHDL 402

Copyright © 2013 Synopsys, Inc.

specifying with attributes 403
INITvalues
Xilinx registers 807
Input and output constraints
defining 139
input constraints, setting 139
input constraints, setting (Legacy) 187
Inputs/Outputs panel
using SCOPE 136
instance-based synthesis
hierarchical projects 257
instances
preserving with syn_noprune 438
properties 549
properties of pins 549
instances (Physical Analyst)
adding markers 691
displaying instances 672
ILM *See* interface logic models
interface logic models 1100
interface timing 1101
IOBUFDS
inference 816
IP
encryption-decryption flow 971
re-encryption 974
IP core areas (Design Planner) 634
IP cores 801
Vivado 854
IP design hierarchy
hyper source 993
IP encryption flow overview 970
IP encryption scheme 975
IP license queuing 1137
IP vendors
directory structure for package 984
encrypting IP 975
package file list for ecnypted IP flow 983
packaging for evaluation 981
supplying vendor information 986
IPs
Altera 998
encrypting 975
encryption flow 970
SYNCore 934
SYNCore byte-enable RAMs 946
SYNCore counters 964
SYNCore FIFOs 934
SYNCore RAMs 939
SYNCore ROMs 952
SYNCore subtractors 957
system-level models, providing 985
using hyper source for debug 993
IP-XACT models 982
ise2syn
description 1051
relationship with ucf2sdc 1051
ise2syn utility
converting Xilinx projects 1051
using 1051
Isolate Paths command
different from Expand Paths 599, 600
ispLEVER
forward-annotating constraints for 784

J

job management
up-to-date checking 509

K

key assignments
customizing 1147
key block 973
keywords
completing words in Text Editor 112

L

latches
generated-clock conversion 499
Lattice
constraint file 126
forward annotation 126
I/O insertion 453
macro libraries 772
PICs 775
Lattice netlist 834
lcell primitive
Clearbox 1002

libraries
Xilinx post-synthesis simulation 827
license queuing 1134
blocking-style 1134
IP 1137
non blocking-style 1135
licenses
DesignWare multiprocessing 989
lists
dependent files 269
local TMR
RAM 356
location constraints
RLOC_ORIGIN 819
RLOCs with synthesis attribute 819
RLOCs with xc_attributes 817
log file
physical synthesis 662
log files
checking FSM descriptions 460
checking information 514
crossprobing to Physical Analyst 698
pipelining description 429
retiming report 433
setting default display 514
shift register report (Altera) 407
state machine descriptons 456
viewing 514
logic
expanding between objects 599
expanding from net 596
expanding from net (Physical Analyst) 707
expanding from pin 595, 704
logic preservation
syn_hier 442
syn_keep for nets 438
syn_keep for registers 438
syn_noprune 438
syn_preserve 438
logic synthesis
in Altera physical synthesis flow 60
in physical synthesis flows 39
translating UCF constraints 201
with design plan 44
logical folders
creating 220
LPM_RAM_DQ
VHDL example 417
LPMs
Altera megafunction example (Verilog) 412
Altera megafunction example (VHDL) 414
black box method simulation flow 767
comparison of Altera instantiation methods 411
creating synthesis projects 1039
generics method, Cypress 416
in .vqm 412
including in physical synthesis 998
instantiating as black boxes 411
instantiating as black boxes (Altera) 412
instantiating with a Verilog library (Altera methodology) 412
instantiating with a Verilog library (Synplicity methodology) 418
instantiating with VHDL prepared components 416
prepared components (Altera), example 416
using in Altera simulation flows 767
Verilog library simulation flow 768
VHDL prepared component simulation flow 768
VHDL prepared components instantiation example 417
LPMs, Altera 411
LUTRAMs, inferring 383

M

mac_mult primitive
Clearbox 1002
mac_out primitive
Clearbox 1002
macro libraries
Lattice 772
macro libraries (Xilinx) 795
macros (Xilinx) 795
manhattan distance (Physical Analyst) 693
manual compile points
compared to automatic 1090
flow 1113

Copyright © 2013 Synopsys, Inc.

using with automatic 1123
markers (Physical Analyst)
adding 691
adding with Go to Location 692
deleting 692
finding objects with 690
measuring with 693
moving 692
navigating between 693
using 691
Markers command 691
max_parallel_jobs variable 1141
maximum parallel jobs 1140, 1156
MaxParallelJobs variable 1140
MegaCore
grey box flow with grey box netlist 1016
greybox flow with IP package 1019
Megacore IPs
importing in a Quartus design 1038, 1039
megafunctions
altplls 760
creating synthesis project 1039
grey boxes 1012
including in physical synthesis 998
inferring Clearbox information 1003
instantiating Clearbox 1007
instantiating Clearbox with netlist 1010
using Clearbox 1014
using grey box netlist 1016
Megawizard
altplls 760
memory usage
maximizing with HDL Analyst 605
Message viewer
filtering messages 523
keyboard shortcuts 522
saving filter expressions 524
searching 522
using 521
using the F3 key to search forward 522
using the Shift-F3 key to search backward 522
messages
demoting 528
filtering 523
promoting 528
saving filter information from command line 525
saving filter information from GUI 524
severity levels 529
suppressing 528
writing messages to file 526
Microsemi
ACTgen macros 790
macro libraries 789
output netlist 834
pin numbers for bus ports 831
minPower library 989
mitigation technology 345
mixed block hierarchical synthesis flow 95
mixed designs
troubleshooting 120
mixed language files 117
mouse strokes
pushing/popping objects 561
mouse strokes (Physical Analyst)
navigating between views 668
multicycle constraints
forward-annotating 784
multicycle paths
setting constraints 137
setting constraints (Legacy) 179
multiple implementations 226
hierarchical projects 270
running from project 227
multipliers
pipelining restriction 426
multipliers, pipelining 426
multi-port RAMs
See also dual-port RAMs
Altera Stratix 382
multiprocessing
compile points 1125
maximum parallel jobs 1140, 1156
multisheet schematics 555
for nested internal logic 592
searching just one sheet 570
transparent instances 555

N

name spaces
  output netlist 578
  technology view 572

navigating among design views 556

ncf file
  cores 1048

ncf files
  output physical constraints (Legacy) 195
  translating to sdc 206
  using as input for logic design 201

netlist editing 889
  commands 891
  RTL-level 889
  Tcl commands 891

netlists
  restructuring options 302

netlists (Physical Analyst)
  analyzing 703

netlists for different vendors 834

nets
  expanding logic from 596
  preserving for probing with syn_probe 438
  preserving with syn_keep 438
  properties 549
  selecting drivers 599

nets (Physical Analyst)
  adding markers 691
  expanding logic from 707
  resetting the display 678
  routing 676
  selecting instances 707
  signal flow 678
  unfiltering for Find command 686

New property 551

NGC cores 1046

NGO core 1046

NIOS II, importing as greybox 1023

non blocking-style license queuing 1135

non-secure core flow
  synthesis 1049

notes
  filtering 522
  sorting 522

notes, definition 111

nram primitive. *See* dual-port RAMs, multi-port RAMs

O

objects
  finding on current sheet 570
  flagging by property 550
  selecting/deselecting 553

objects (Physical Analyst)
  finding 686
  finding by location 690
  overlapping 679
  select overlapping 679
  selecting 679

OBUFDS
  inference 816

OBUFTDS
  inference 816

open_design
  with find and expand 159

optimization
  for area 423
  for timing 424
  generated clock 499
  logic preservation. *See* logic preservation.
  mapper effort. *See* fast synthesis 425
  preserving hierarchy 442
  preserving objects 438
  tips for 422

optimizing
  enhanced logic optimizations 41

options
  HDL Analyst Quick Load 234

options file (place-and-route) 308

orig_inst_of property 552

output constraints, setting 139

output constraints, setting (Legacy) 187

output files 834
  specifying 237

output netlists
  finding objects 578

Copyright © 2013 Synopsys, Inc.

P

p_nram primitive. *See* dual-port RAMs, multi-port RAMs, nram primitive

package library, adding 214

pad types
  industry standards 140

PAR_BELDLYRPT 304

parallel jobs 1140

parameter passing 121
  boolean generics 120

parameters
  extracting from Verilog source code 240

part selection options 228

partition flow, Xilinx 873

partitioning (Synplify Premier)
  bit slicing 652

path constraints
  false paths 151
  false paths (Legacy) 188

pathnames
  using wildcards for long names (Find) 573

paths
  crossprobing 585
  tracing between objects 599
  tracing from net 596
  tracing from pin 595

paths (Physical Analyst)
  tracing between objects 709
  tracing from net 707
  tracing from pin 704

pattern matching
  Find command (Tcl) 153

pattern searching 284

PDF
  cutting from 112

Physical Analyst
  analyzing netlists 703
  context window 669
  control panel 666
  crossprobing from text files 698
  crossprobing RTL view 699
  crossprobing to Technology view 701
  displaying instances 672
  identifying encrypted IP objects 693
  opening 665
  overlapping objects 679
  properties 681
  using Block Inputs Map 716
  using Block Utilization Map 715
  using implementation maps 710
  using Routing Congestion map 713
  using Slack Map 717

Physical Analyst view
  adding markers 692
  critical paths 752
  crossprobing 695
  displaying net signal flow 678
  Expand commands 704
  filtering 703
  finding objects 686
  Go to Location command 690
  selecting objects 679
  tool tips (Physical Analyst) 684
  using markers 691
  zoom selected objects 667

physical constraints
  design plan-based physical synthesis 52
  design-plan based logic synthesis 44
  translating UCF constraints 203
  Xilinx output file (Legacy) 195

physical constraints (Design Planner - Altera)
  Altera guidelines 641

physical constraints (Design Planner -Xilinx)
  Xilinx guidelines 650

physical coordinates
  marking 691

Physical Plus Flow
  running 74, 78
  Xilinx 64

physical synthesis
  Altera 55
  analyzing results 662
  graph-based (Altera) 55
  improve performance (Altera) 769, 825
  running place-and-route 842
  translating UCF constraints 203
  using design plan file 618
  with back annotation 326
  with design plan file 52

PICs 775

pin assignment (Design Planner) 619
  assigning clock pins 622
  crossprobing 628
  temporary assigns 624
pin assignment tool (Design Planner) 615
pin assignments
  converting to constraints 321
pin assignments (Design Planner)
  temporary 624
pin loc constraint files
  converting to SDC 321
pin locations
  specifying (Xilinx) 809
pin names, displaying 594
pins
  expanding logic from 595, 704
  properties 549
pipelining
  adding attribute 428
  definition 426
  multipliers 426
  prerequisites 426
  whole design 427
place-and-route
  creating implementation 303
  customizing ISE option file 311
  customizing option file 308
  customizing Vivado option file 860
  placement constraint file 326
  with back annotation 326
  with physical synthesis 842
place-and-route implementations 303
PLLs
  defining clocks (Legacy) 183
ports
  false path constraint 151
  false path constraint (Legacy) 188
  properties 549
POS interface
  using 147
post_route.dcp file 866
post-synthesis constraints with adc 733
post-synthesis simulation, Xilinx 826
preferences
  crossprobing to place-and-route file 557
  displaying Hierarchy Browser 557
  displaying labels 558
  RTL and Technology views 557
  sheet size (UI) 557
preplace.srm file 663
preserving region resources
  Design Plan Editor view 633
primitives
  pin name display 594
  pushing into with mouse stroke 562
  viewing internal hierarchy 590
primitives (Synplify Premier)
  breaking up large 652
probes
  adding in source code 462
  definition 462
  retiming 436
process-level hierarchy 652
Product of Sums interface. *See* POS interface
project command
  archiving projects 287
  copying projects 294
  unarchiving projects 291
project file hierarchy 220
project files
  adding files 216
  adding source files 212
  batch mode 1132
  creating 212
  definition 212
  deleting files from 216
  opening 215
  replacing files in 216
  updating include paths 219
  VHDL file order 215
  VHDL library 214
projects
  archiving 287
  copying 294
  creating from Vivado IP 1043
  files after importing from Quartus 1036
  importing from Quartus 1033

Copyright © 2013 Synopsys, Inc.

restoring archives 291
properties
copying and pasting (Physical Analyst) 685
displaying with tooltip 549
encrypted IP cells (Physical Analyst) 693
finding objects with Tcl find -filter 154
orig_inst_of 552
reporting for collections 168
viewing for individual objects 549
prototyping
using hyper source threading 993
Push/Pop mode
HDL Analyst 561
keyboard shortcut 563
using 561, 563

Q

qsf
importing 1033
translating I/O constraints 193
qsf file
translated files for synthesis 1036
qsf2sdc
translating constraints 193
qsf2syn.log file 1035
Quartus
batch mode 771
imported settings and constraints 1037
importing design with Megacore IP 1038
importing design with megafunctions 1039
importing LPMs 1039
importing megafunctions 1039
importing projects from 1033
incremental flows 846
integrated flow 769
interactive flow 770
supported constraints for import 1037
supported project settings for import 1037
synthesis project files 1036
using instantiated Clearbox netlist files 1012
Quartus Fast Fit flow 847
Quartus II
using synthesis results to run 769
Quartus II Incremental Compilation flow 848
Quartus II Incremental Synthesis
running 848
Quartus incremental compilation flow 846
QUARTUS_ROOTDIR variable
inferring Clearbox megafunctions 1004
instantiating Clearbox 1007
question mark wildcard, Find command 573

R

radiation effects. *See* high reliability
RAM
local TMR 356
RAM inference
multi-port RAMs, Altera 382
Stratix dual-port 377
ram_block primitive
Clearbox 1002
RAMs
Altera Stratix 376
compiling with SYNCore 939
dual-port, Stratix 377
ECC 357
initializing 394
initializing values (Xilinx) 398
mapping LUTRAMs 383
multi-port. *See* dual-port RAMs, multi-port RAMs
SYNCore 939
SYNCore, byte-enable 946
TMR 356
RAMs, inferring 369
advantages 369
Xilinx block RAMs 385
region clock buffers (BUFR) 816
regions
retiming 437
regions (Design Planner)
preserving logic and memory resources 633

replicating logic manually
replicating instances (Design Planner) 638

register balancing. *See* retiming

register constraints, setting (Legacy) 178

register packing
See also syn_useioff attribute 804
Altera 763
Xilinx 804

registers
false path constraint 151
false path constraint (Legacy) 188
INIT value 807

Registers panel
using SCOPE 136

relative placement. *See* RLOCs

replication
controlling 446

reports
gated clock conversion 477
shift registers, Altera 407

resource sharing 775
optimization technique 423
overriding option with syn_sharing 448
results example 448
using 448

resynthesis
compile points 1108
forcing with Resynthesize All 1108
forcing with Update Compile Point Timing Data 1108

retiming
effect on attributes and constraints 434
example 432
overview 430
probes 436
regions 437
report 433
simulation behavior 436

return codes 1132

RLOC_ORIGINs
specifying 819

RLOCs 817, 819
specifying with synthesis attribute 819
specifying with xc attributes 817

ROM
block RAM mapping (Xilinx) 392

rom.info file 563

ROMs
pipelining 426
SYNCore 952
viewing data table 563

Routing Congestion map
Physical Analyst 713

RTL
netlist editing 889

RTL view
See also HDL Analyst
analyzing clock trees 725
crossprobing collection objects 163
crossprobing description 581
crossprobing from 582
crossprobing from Text Editor 584
defined 547
description 546
filtering 593
finding objects with Find 570
finding objects with Hierarchy Browser 568
flattening hierarchy 601
highlighting collections 168
opening 548
selecting/deselecting objects 553
sequential shift components 406
setting preferences 557
state machine implementation 457
traversing hierarchy 560

RTL views
hierarchical projects 282
hierarchical subprojects 268

run_vivado.tcl
*See also*Vivado, options file

run_vivado.tcl file 859

S

safe case 359

safe FSM 359
using Hamming Distance 3 360
using safe case 359

schematics
multisheet. *See* multisheet schematics

Copyright © 2013 Synopsys, Inc.

page size 557
selecting/deselecting objects 553
SCOPE
adding attributes 250
adding probe insertion attribute 463
assigning Xilinx pin locations 810
Attributes panel 136
case sensitivity for Verilog designs 153
Clocks panel 135
collections compared to Tcl script window 161
Collections panel 135
Compile Points panel 136
creating compile-point constraint file 1119
defining compile points 1116
Delay Paths panel 136
drag and drop 144
editing operations 145
Generated Clocks panel 135
I/O pad type 140
I/O Standards panel 136
Inputs/Outputs panel 136
multicycle paths 150
pipelining attribute 427
Registers panel 136
setting compile point constraints 1120
setting constraints (FDC) 130
specifying constraints 135
specifying RLOCs 817, 819
state machine attributes 343
TCL View 136
SCOPE editor
using 130
scope of the document 29
SCOPE panels
entering and editing constraints 135
SCOPE TCL View
using 141
sdc
converting from Xilinx ucf 206
sdc constraints
manually converting UCF 200
search
browsing objects with the Find command 569
browsing with the Hierarchy Browser 568
finding objects on current sheet 570
setting limit for results 572
setting scope 571
using the Find command in HDL Analyst views 570
secure core flow
synthesis 1049
See also search
Select Net Instances command (Physical Analyst) 707
select RAM
initializing 404
selecting objects (Physical Analyst) 679
Send Crossprobes when selecting command 695
sequential shift components
Altshift_tap 405
mapping 405
SRL16 primitives 405
Verilog 410
VHDL 409
sequential shift components *See* shift registers
set command
collections 169
set_option command 230
sfp file
creating 618
logic synthesis 44
physical synthesis 52
sheet connectors
navigating with 556
sheet size
setting number of objects 557
shift register lookup table. *See* sequential shift components
shift registers
inferring 405
Shift-F3 key
Message Viewer 522
Show Cell Interior option 590
Show Context command
different from Expand 592
using 592
signal flow (Physical Analyst) 678
displaying 678

Signal Flow command 678
signal pins (Physical Analyst)
  displaying 675
signals
  threading with hyper source. *See* hyper source
simulation, effect of retiming 436
single-port RAMs
  block RAM with registered output, Xilinx 389
  SYNCore parameters 942
site columns
  properties (Physical Analyst) 684
sites (Physical Analyst)
  properties 684
slack 728
  setting margins 725
Slack Map
  Physical Analyst 717
slack time display 722
slice_primitive command 653
Slow property 551
SmartGuide flow 870
SOPC Builder
  importing embedded systems 1023
source code
  adding pipelining attribute 428
  commenting with synthesis on/off 243
  crossprobing from Tcl window 587
  defining FSMs 340
  fixing errors 113
  opening automatically to crossprobe 583
  optimizing 422
  specifying RLOCs 817, 819
source files
  See also Verilog, VHDL.
  adding comments 112
  adding files 212
  checking 110
  column editing 112
  copying examples from PDF 112
  creating 100
  crossprobing 584
  crossprobing to Physical Analyst 698
  editing 111
  editing operations 111
  mixed language 117
  specifying default encoding style 242
  specifying top level file for mixed language projects 118
  specifying top level in Project view 215
  specifying top-level file 242
  state machine attributes 343
  using bookmarks 112
special characters
  Tcl collections 167
SRLs *See* shift registers
STA 729
STA, generating custom timing reports 729
STA, using analysis design constraints (adc) 732
stand-alone timing analyst. *See* STA
startup block (Xilinx) 797
state machines
  See also FSM Compiler, FSM Explorer, FSM viewer, FSMs.
  attributes 343
  descriptions in log file 456
  implementation 457
  parameter and 'define comparison 341
statemachine.info file 609
Stratix
  dual-port rams 377
sub-projects
  creating 254
  linking updates to top level 256
  multiple implementations 270
  synchronizing device options with top level 281
subprojects
  block-based 254
  compiling 266
  instance-based 257
  nested 263
subtractors
  SYNCore 957
syn_allow_retiming
  using for retiming 431
syn_allowed_resources
  compile points 1121
syn_black_box
  instantiating LPMs (Altera) 412

Copyright © 2013 Synopsys, Inc.

syn_dspstyle attribute
inferring wide adders/subtractors 799
syn_edif_bit_format attribute 801
syn_edif_scalar_format attribute 801
syn_encoding attribute 343
syn_enum_encoding directive
FSM encoding 344
syn_force_pad attribute
using 453, 783
syn_forward_io_constraints attribute 125
syn_hier attribute
Altera Quartus II Incremental Compilation flow 850
controlling flattening 442
preserving hierarchy 442
using with compile points 1121
syn_highrel_ioconnector
uisng 353
syn_insert_buffer attribute
BUFGMUX 815
syn_isclock
black box clock pins 338
syn_keep
inferring Altera shift registers 406
inferring Lattice PICs 776
replicating redundant logic 439
syn_keep attribute
preserving nets 438
preserving shared registers 438
syn_keep directive
effect on buffering 446
syn_macro
specifying encrypted IP as white box 980
white-boxing non-secure cores 1049
syn_maxfan attribute
setting fanout limits 444
Xilinx buffers 447
syn_noarrayports attribute
use with alspin 831
syn_noprune directive
inferring Altera shift registers 406
preserving instances 438
syn_pipeline attribute 428
syn_preserve
effect on buffering 446
preserving power-on for retiming 432
preserving registers with INIT values 807
syn_preserve directive
preserving FSMs from optimization 343
preserving logic 438
syn_probe attribute 462
inserting probes 462
preserving nets 438
syn_radhardlevel
distributed DWC 350
distributed TMR 346
syn_ramstyle attribute
glue logic for Altera Stratix RAMs 376
multi-port RAM inference 373
preventing glue logic (no_rw_check) 387
syn_reference_clock constraint (Legacy) 173
syn_replicate attribute
using buffering 447
syn_sharing directive
overriding default 448
syn_srlstyle attribute
altshift_tap 405
mapping sequential shift components to registers 405
setting shift register style 405
syn_state_machine directive
using with value=0 457
SYN_TCL_HOOKS environment variable 1146
syn_tco attribute
adding in SCOPE 337
syn_tco directive 336
adding black box constraints 335
syn_tpd attribute
adding in SCOPE 337
syn_tpd directive 336
adding black box constraints 335
syn_tsu attribute
adding in SCOPE 337
syn_tsu directive 336
adding black box constraints 335

syn_use_carry_chain attribute
using 775
syn_useioff
preventing flops from moving during retiming 432
syn_useioff attribute
inferring Altera shift registers 406
packing registers (Altera) 763
packing registers (Xilinx) 804
SYN_XILINX_GLOBAL_PLACE_OPT environment variable 319
SYNCore
adders 957
counters 964
FIFO compiler 934
RAMs 939
RAMs, byte-enable 946
RAMs, dual-port parameters 943
RAMs, single-port parameters 942
ROMs 952
ROMs, parameters 956
subtractors 957
synenc encryption 992
synhooks
automating message filtering 526
synhooks.tcl file 1146
Synopsys
FPGA product family 24
synplicity.ucf file
non-secure cores 1048
relation to ncf file (Legacy) 195
secure cores 1048
Synplify Premier
list of design flows 39
logic synthesis flows 39
Synplify Premier flows
Xilinx Physical Plus 64
Synplify Premier synthesis tool
overview 24
Synplify Pro synthesis tool
overview 24
synplify UNIX command 30
synplify_premier UNIX command 30
synplify_premier_dp UNIX command 30
synplify_pro UNIX command 30
synplify.ucf 207
synplify.vhd 827
syntax
checking source files 110
syntax check 110
synthesis
hierarchical projects 278
Xilinx non-secure cores 1049
Xilinx secure cores 1049
synthesis check 110
synthesis_on/off
using 243
SystemDesigner
using with Xilinx IP 982
SystemVerilog keywords
context help 108

T

ta file 729
Tcl
max_parallel_jobs variable 1141
tcl callbacks
customizing key assignments 1147
Tcl commands
batch script 1133
netlisting editing 891
running 1139
Tcl expand
using 152
Tcl expand command
crossprobing objects 163
usage tips 157
using in SCOPE 162
Tcl files 1139
creating 1141
for bottom-up synthesis 1145
guidelines 123
naming conventions 124
recording from commands 1140
synhooks.tcl 1146
using variables 1143
wildcards 124
Tcl find
batch mode 159
filtering results by property 154
search patterns 152
using 152

Tcl find command
annotating properties 154
case sensitivity 153
crossprobing objects 163
database differences 162
pattern matching 153
Tcl window vs SCOPE 161
usage tips 156
useful -filter examples 155
using in SCOPE 162

Tcl Script window
crossprobing 587
message viewer 521

Tcl script window
collections compared to SCOPE 161

Tcl scripts
See Tcl files.

TCL View 141
uisng 141
using SCOPE 136

team design. *See* hierarchical projects, hierarchical project management flows

Technology view
See also HDL Analyst
critical paths 725
crossprobing 581, 582
crossprobing collection objects 163
crossprobing from source file 584
filtering 593
finding objects 572
finding objects with Find 570
finding objects with Hierarchy Browser 568
flattening hierarchy 601
general description 546
highlighting collections 168
opening 548
selecting/deselecting objects 553
setting preferences 557
state machine implementation in 457
traversing hierarchy 560

temporary assigns (Design Planner) 624
drag and drop 624
empty 624
return assignment 624

text editor
built-in 111
external 116
using 111

Text Editor view
crossprobing 582

Text Editor window
colors 114
crossprobing 114
fonts 114

text files
crossprobing 584

The Synopsys FPGA Product Family 24

third-party vendor tools
invoking 836

through constraints 147
AND lists 148
OR lists 147

time stamp, checking on files 217

time stamps
Xilinx partition flow 876

TimeQuest
supported constraints for import 1038

timing
after logic synthesis 664

timing analysis 722

timing analysis using STA 729

timing budgeting
compile points 1101

timing constraints 123
translating qsf 1036
Xilinx output file (Legacy) 195

timing constraints (Legacy) 173

timing exceptions, adding constraints after synthesis 733

timing exceptions, modifying with adc 733

timing failures 728

timing information commands 722

timing information in HDL views 723

timing information, critical paths 727

timing optimization 424

Timing Report View 744
using 744

timing report, stand-alone 729

timing reports
specifying format options 238

Copyright © 2013 Synopsys, Inc.

timing reports, custom 729
timing_applied.sdc file 1036
timing_unapplied.sdc file 1036
tips
  memory usage 605
TMR
  description 345
  distributed 346
  local. *See* local TMR 356
  using 346
  using for ECC 356
  using for RAM 356
  voter insertion examples 348
to constraints
  specifying 147
tool tags
  creating 836
  definition 836
tool tips
  Physical Analyst 684
tooltips (Physical Analyst)
  copying information from 685
top level
  specifying 242
top-down design flow
  compile point advantages 1088
top-down hierarchical develpment flow
  sub-projects 254
top-down hierarchical synthesis flow 93
top-first hierarchical development flow 89
transparent instances
  flattening 601
  lower-level logic on multiple sheets 555
triple modular redundancy. See TMR

U

UCF
  Vivado support 854
UCF constraints 200
  converting to sdc manually 200
  input files 206
  supported 208
  translating for logic synthesis 201
  translating for physical synthesis 203
  unsupported 209
ucf file
  using as input for logic design 201
ucf file (Legacy). *See also* synplicity.ucf
ucf2sdc 200
  relationship with ise2syn 1051
ucf2sdc.log file 207
UINISIM library
  simulation 827
unisim libraries
  Virtex 2 with ISE 11 795
UNISIM library 795
UNIX commands
  synplify 30
  synplify_premier 30
  synplify_premier_dp 30
  synplify_pro 30
unsupported.ucf 207
up-to-date checking 509
  copying job logs to log file 511
  limitations 512

V

VCS-Analyst Integration tool 904
  using 904
vendor-specific netlists 834
verification
  using VIF file 893
Verilog
  ‘define statements 240
  adding attributes and directives 247
  adding probes 462
  Altera LPM library 418
  Altera LPM megafunction example 412
  Altera PLLs 760
  always block hierarchy 303
  black boxes 330
  black boxes, instantiating 330
  case sensitivity for Tcl Find command 153
  checking source files 110
  choosing a compiler 239
  clock DLLs 821
  creating source files 100
  crossprobing from HDL Analyst view 582

crossprobing to Physical Analyst 698
defining FSMs 340
defining state machines with parameter and 'define 341
editing operations 111
extracting parameters 240
include paths, updating 219
initializing RAMs 394
instantiating LPMs as black boxes (Altera) 412
macro library (Xilinx) 795
Microsemi ACTgen macros 790
mixed language files 117
RAM structures for inference 370
RLOCs 818
sequential shift components 410
specifying compiler directives 240
specifying top-level module 242
structural, for instantiated Clearbox 1010

Verilog 2001
setting global option from the Project view 239
setting option per file 239

Verilog macro libraries
Lattice 772
Microsemi 789

Verilog model (.vmd) 1100

VHDL
adding attributes and directives 246
adding probes 462
Altera LPM megafunction example 414
Altera PLLs 760
black boxes 332
black boxes, instantiating 332
case sensitivity for Tcl Find comand 153
checking source file 110
clock DLLs 821
constants 243
creating source files 100
crossprobing from HDL Analyst view 582
crossprobing to Physical Analyst 698
defining FSMs 341
editing operations 111
extracting generics 243
file order in mixed designs 120
global signals in mixed designs 120
initializing RAMs with variable declarations 397
initializing with signal declarations 395
instantiating LPMs as black boxes (Altera) 412
LPM instantiation example 417
macro libraries, Microsemi 789
macro library (Xilinx) 795
mixed language files 117
prepared components method of instantiation 417
process hierarchy 303
RAM structures for inference 370
RLOCs 818
sequential shift components 409
specifying top-level entity 242
structural, for instantiated Clearbox 1010

VHDL files
adding library 214
adding third-party package library 214
order in project file 215
ordering automatically 215

VHDL macro libraries
Lattice 773

vi text editor 116

VIF file
using 893

vif2conformal.tcl script 897

Virtex
block RAM. *See also block RAM.*
clock buffers 820
I/O buffers 823
netlist 834
PCI core 801

virtual clock, setting (Legacy) 178

Vivado
environment variables 855
flow 853
netlists 853
options file 860, 861
running 856
running incrementally 864
running place-and-route 852

Vivado IP
importing 1043

vqm
inferred Clearbox 1005

Copyright © 2013 Synopsys, Inc.

instantiated Clearbox 1007
instantiated Clearbox with netlist 1012

W

warning messages
definition 111
warnings
feedback muxes 425
filtering 522
handling 530
sorting 522
Watch window 518
moving 518, 521
multiple implementations 227
resizing 518, 521
white boxes
defined for ise2syn flow 1059
EDK cores 1059
using syn_macro on non-secure cores 1049
wide adders/subtractors
example 800
inferring 798
prerequisites for inference 799
wildcards
effect of search scope 573
Find command (Tcl) 153
message filter 524
wildcards (Find)
examples 575
how they work 573
wildcards (Physical Analyst)
in Find command 688
write modes, Virtex-II 388

X

xc_clockbuftype attribute
specifying 820
xc_fast attribute
for critical paths 794
xc_loc attribute
assigning locations in SCOPE 810
xc_map attribute
relative location 817
xc_padtype attribute
specifying I/Os 823
xc_rloc attribute
specifying relative location 818
xc_uset attribute
grouping instances for relative placement 818
using to group instances 818
xcf files
translating to sdc 206
xdc
converting to fdc 190
xdc2fdc
converting constraints 190
debugging translation 191
xflow script 311
Xilinx
block RAMs 385
clock buffers 820
converting PAD files 321
converting pin assignments to SDC 321
CoreGen 801
design guidelines 794
forward-annotation 126
GSR 797
I/O buffers 823
I/O insertion, manual 814
I/O locations 809
including cores for synthesis 1049
INIT property 399
INIT property, VHDL 402
IP cores 801, 1046
macro libraries 795
macros 795
netlist 834
non-secure core flow 1049
packing registers 804
partition flow 869
place-and-route option file, ISE 311
place-and-route option file, Vivado 860
post-synthesis simulation 826
secure core flow 1049
shift registers 405
specifying pin location 809
startup blocks 797
synthesis constraint files (Legacy) 195
tips for optimizing 794
using BUFR 816

Copyright © 2013 Synopsys, Inc.

Virtex-II write modes 388
Xilinx differential I/O buffer inference 815
Xilinx projects
converting with ise2syn 1051
xpartition.pxml file 876
xtclsh flow 311

Z

zoom selected objects (Physical Analyst) 667