Synopsys FPGA Synthesis
Reference Manual

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: Product Overview

Synopsys FPGA and Prototyping Products 38
FPGA Implementation Tools 38
Identify Tool Set 40
Synphony Model Compiler 40
Rapid Prototyping 41

Overview of the Synthesis Tools 42
Common Features 42
Synplify Pro and Synplify Premier Features 43
Synplify Premier Features 44
BEST Algorithms 45
Graphic User Interface 45
Projects and Implementations 51

Starting the Synthesis Tool 52
Starting the Synthesis Tool in Interactive Mode 52
Starting the Tool from the Command Line 53

Logic Synthesis Overview 56
Synthesizing Your Design 57

Getting Help 59

Chapter 2: User Interface Overview

The Project View 62
Synplify Pro and Synplify Premier Project Management View 66

The Project Results View 69
Project Status Tab 69
Implementation Directory 76
Process View 77

Other Windows and Views 80
Dockable GUI Entities 81

Copyright © 2013 Synopsys, Inc.

Watch Window 81
Tcl Script and Messages Windows 84
Tcl Script Window 85
Message Viewer 85
Output Windows (Tcl Script and Watch Windows) 89
RTL View 90
Technology View 91
Hierarchy Browser 93
FSM Viewer Window 95
Text Editor View 97
Context Help Editor Window 99
Demos & Examples 101
Interactive Attribute Examples 102
Search SolvNet 103

FSM Compiler 105
When to Use FSM Compiler 106
Where to Use FSM Compiler (Global and Local Use) 106

FSM Explorer 107

Using the Mouse 107
Mouse Operation Terminology 108
Using Mouse Strokes 108
Using the Mouse Buttons 110
Using the Mouse Wheel 112

User Interface Preferences 112
Managing Views 113

Toolbars 114
Project Toolbar 114
Analyst Toolbar 116
Physical Analyst Toolbar 119
Text Editor Toolbar 119
FSM Viewer Toolbar 120
Tools Toolbar 121
Design PlannerToolbar 122

Keyboard Shortcuts 123

Buttons and Options 131

Copyright © 2013 Synopsys, Inc.

Chapter 3: User Interface Commands

Command Access 139

File Menu 141
New Command 142
Create Image Command 144
Build Project Command 146
Open Project Command 146

Edit Menu 147
Find Command (Text) 149
Find Command (In Project) 150
Find Command (HDL Analyst) 152
Find in Files Command 156
Replace Command 158
Goto Command 159

View Menu 160
Toolbar Command 163
View Sheets Command 164
View Log File Command 165

Project Menu 168
Add Source File Command 170
Change File Command 171
Set VHDL Library Command 172
Add Implementation Command 172
Convert Vendor Constraints Command 173
Archive Project Command 175
Un-Archive Project Command 176
Copy Project Command 179
Hierarchical Project Options Command 182
Insert Sub-project Command 183
Configure Revision Control Tools Command 184
Create New SCM Configuration 187
Confirm Revision-Control Operation Dialog Box 192
SCM Tcl Script File 193

Implementation Options Command 194
Device Panel 195
Options Panel 197
Constraints Panel 200
Implementation Results Panel 203
Timing Report Panel 205

Copyright © 2013 Synopsys, Inc.

High Reliability Panel 206
VHDL Panel 207
Verilog Panel 211
Compiler Directives and Design Parameters 215
Push Tristates Option 220
GCC Panel 221
GCC & Prototyping Tools Panel 222
Place and Route Panel 223

Import Menu 224
Import Xilinx EDK/ISE Project Command 225
Import Altera SOPC Project Command 228
Import Altera QSF Project Command 231
Import Xilinx Constraints 232
Import IP Package Command 235
Launch System Designer Command 236

Run Menu 237
Run Tcl Script Command 240
Run All Implementations Command 241
Translate Vendor IO Command 242
Job Status Command 243
Launch Identify Instrumentor Command 243
Launch Identify Debugger Command 245
Launch SYNCore Command 245
SYNCore FIFO Wizard 247
SYNCore RAM Wizard 257
SYNCore Byte-Enable RAM Wizard 261
SYNCore ROM Wizard 264
SYNCore Adder/Subtractor Wizard 268
SYNCore Counter Wizard 272
Configure and Launch VCS Simulator Command 274

Analysis Menu 284
Timing Report View 285
Timing Report Generation Parameters 291
Activity Analysis Generation 301
Generate Congestion Analysis 303

HDL Analyst Menu 304
HDL Analyst Menu->RTL and Technology View Submenus 305
HDL Analyst Menu: Hierarchical and Current Level Submenus 306
HDL Analyst Menu: Filtering and Flattening Commands 308
HDL Analyst Menu: Timing Commands 311

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu: Analysis Commands 311
HDL Analyst Menu: Selection Commands 315
HDL Analyst Menu: FSM Commands 315
HDL Analyst Menu: VCD Commands 316

Tools Menu 329

Options Menu 330
Configure Compile Point Process Command 332
Project View Options Command 335
Editor Options Command 340
Place and Route Environment Options Command 343
Configure 3rd Party Tools Options Command 344
HDL Analyst Options Command 345
Configure External Programs Command 352

Tech-Support Menu 353
Submit Support Request Command 353
Web Support Command 356

Web Menu 357
Check Resource Center Messages Command 358
Configure Resource Center Command 360

Help Menu 362
Search SolvNet 363
Preferred License Selection Command 364
Tip of the Day Command 365

Popup Menus 365
Watch Window Popup Menu 366
Tcl Window Popup Menu 367
Text Editor Popup Menu 367
Log File Popup Menu 368
FSM Viewer Popup Menu 369

Project View Popup Menus 372
Project Management View Popup Folder Commands 378
Vendor Tool Invocation Popup Menu Command 380
File Options Popup Menu Command 381
Copy File Popup Menu Command 383
Change Implementation Popup Menu Commands 384
Show Compile Points Popup Menu Command 384
Project Options Popup Menu Command 385
Add P&R Implementation Popup Menu Command 386
Options for Place & Route Jobs Popup Menu Command 390

Copyright © 2013 Synopsys, Inc.

Revision Control Management Popup Menu Commands 391
Create Subproject Popup Menu Commands 391
Design Block/Instance Properties Popup Menu Command 395
Insert Sub-project Command 397
Sub-project Parameter Sync 397
Insert & Link Subproject to Module Command 398
Allocate Timing and Resource Budgets 400

RTL and Technology Views Popup Menus 402

Chapter 4: HDL Analyst Tool

HDL Analyst Views and Commands 416
Filtered and Unfiltered Schematic Views 416
Accessing HDL Analyst Commands 417

Schematic Objects and Their Display 418
Object Information 418
Sheet Connectors 419
Primitive and Hierarchical Instances 420
Transparent and Opaque Display of Hierarchical Instances 421
Hidden Hierarchical Instances 423
Schematic Display 423

Basic Operations on Schematic Objects 427
Finding Schematic Objects 427
Selecting and Unselecting Schematic Objects 428
Crossprobing Objects 429
Dragging and Dropping Objects 431

Multiple-sheet Schematics 432
Controlling the Amount of Logic on a Sheet 432
Navigating Among Schematic Sheets 432
Multiple Sheets for Transparent Instance Details 434

Exploring Design Hierarchy 435
Pushing and Popping Hierarchical Levels 435
Navigating With a Hierarchy Browser 438
Looking Inside Hierarchical Instances 440

Filtering and Flattening Schematics 442
Commands That Result in Filtered Schematics 442
Combined Filtering Operations 443
Returning to The Unfiltered Schematic 444

Copyright © 2013 Synopsys, Inc.

Commands That Flatten Schematics 444
Selective Flattening 446
Filtering Compared to Flattening 447

Timing Information and Critical Paths 448
Timing Reports 448
Critical Paths and the Slack Margin Parameter 449
Examining Critical Path Schematics 450

Chapter 5: Constraints

Constraint Types 454

Constraint Files 455

Timing Constraints 457

FDC Constraints 460

Methods for Creating Constraints 461

Constraint Translation 464

Constraint Checking 464

Database Object Search 466

Forward Annotation 467

Auto Constraints 467

Chapter 6: SCOPE Constraints Editor

SCOPE User Interface 470

SCOPE Tabs 471
Clocks 472
Generated Clocks 478
Collections 480
Inputs/Outputs 482
Registers 484
Delay Paths 486
Attributes 488
I/O Standards 489
Compile Points 491
TCL View 494

Industry I/O Standards 496
Industry I/O Standards 497

Copyright © 2013 Synopsys, Inc.

Delay Path Timing Exceptions 500
Multicycle Paths 500
False Paths 503

Specifying From, To, and Through Points 506
Timing Exceptions Object Types 506
From/To Points 506
Through Points 508
Product of Sums Interface 509
Clocks as From/To Points 511

Conflict Resolution for Timing Exceptions 513

SCOPE User Interface (Legacy) 517
SCOPE Tabs (Legacy) 518
Clocks (Legacy) 519
Clock to Clock (Legacy) 525
Inputs/Outputs (Legacy) 527
Registers (Legacy) 531
Delay Paths (Legacy) 532
Other (Legacy) 534

Chapter 7: Physical Synthesis User Interface

Design Planner User Interface 536
Design Planner View 536
Design Planner Toolbar 542
Design Planner Keyboard Shortcuts 542

Design Planner User Interface Commands 543
File Menu: Design Planner Commands 543
Edit Menu: Design Planner Commands 546
View Menu: Design Planner Commands 547
Implementation Options: Design Planner 550
Run Menu: Design Planner Commands 551

Design Planner Tools Menu 552
Tools Menu Commands 552
Show Replicated Assignments Command 553
Preferences Command 553

Design Planner Popup Menus 558

Physical Analyst User Interface 566
Physical Analyst View 566
Physical Analyst Toolbar 571
Physical Analyst Keyboard Shortcuts 572

Copyright © 2013 Synopsys, Inc.

Physical Analyst User Interface Commands 574
Edit Menu: Physical Analyst Commands 574
View Menu: Physical Analyst Commands 574
HDL Analyst Menu: Physical Analyst Commands 579
Physical Analyst 579
Options Menu: Physical Analyst Commands 579

Physical Analyst View Popup Menus 580
Global Commands 580
Commands with Instances Selected 582
Commands with Nets Selected 584
Design, Instance, and Net Properties (Physical Analyst) 584
Find Command (Physical Analyst View) 587
Resolve Selection Dialog Box 591

Chapter 8: Input and Result Files

Input Files 594
HDL Source Files 596

Libraries 599
The Synplicity Generic Technology Library 600

Output Files 601

Log File 608
Compiler Report 610
Premap Report 610
Mapper Report 611
Clock Buffering Report 611
Net Buffering Report 611
Compile Point Information 612
Timing Section 612
Resource Usage Report 612
Retiming Report 613
Errors, Warnings, Notes, and Messages 613

Timing Reports 615
Timing Report Header 616
Performance Summary 616
Clock Relationships 618
Interface Information 619
Detailed Clock Report 620
Asynchronous Clock Report 622

Hierarchical Area Report 623

Copyright © 2013 Synopsys, Inc.

Constraint Checking Report 629
Gated Clock Conversion Report 636

Chapter 9: Verilog Language Support

Support for Verilog Language Constructs 639

Verilog 2001 Support 642
Combined Data, Port Types (ANSI C-style Modules) 643
Comma-separated Sensitivity List 644
Wildcards (*) in Sensitivity List 644
Signed Signals 645
Inline Parameter Assignment by Name 646
Constant Function 646
Configuration Blocks 647
Localparams 656
$signed and $unsigned Built-in Functions 656
$clog2 Constant Math Function 657
Generate Statement 658
Automatic Task Declaration 659
Multidimensional Arrays 660
Variable Partial Select 661
Cross-Module Referencing 663
ifndef and elsif Compiler Directives 666

Verilog Synthesis Guidelines 667
General Synthesis Guidelines 667
Library Support in Verilog 668
Constant Function Syntax Restrictions 672
Multi-dimensional Array Syntax Restrictions 672
Signed Multipliers in Verilog 674
Verilog Language Guidelines: always Blocks 675
Initial Values in Verilog 676
Cross-language Parameter Passing in Mixed HDL 683
Library Directory Specification for the Verilog Compiler 683

Verilog Module Template 685

Scalable Modules 686
Creating a Scalable Module 686
Using Scalable Modules 687

Built-in Gate Primitives 689

Combinational Logic 690
Combinational Logic Examples 690
always Blocks for Combinational Logic 691
Continuous Assignments for Combinational Logic 693
Signed Multipliers 694

Sequential Logic 695
Sequential Logic Examples 695
Flip-flops Using always Blocks 696
Level-sensitive Latches 697
Sets and Resets 699
SRL Inference 704

Verilog State Machines 705
State Machine Guidelines 705
State Values 707
Asynchronous State Machines 708

Verilog Guidelines for RAM Inference 709
RAM Inference from Verilog Code 709
Limited RAM Resources 710
Additional Glue Logic 710
Synchronous READ RAMs 711
Multi-Port RAM Extraction 711

Verilog Coding for Altera RAM Inference 717
Stratix III LUTRAM Verilog Examples 717
Stratix, Arria and Cyclone Dual-Port Verilog RAM Examples 720
Verilog Coding for StratixIII and StratixIV RAMs with Control Signals 724
Verilog Coding for Cyclone III RAMs with Control Signals 730

Verilog Coding for Xilinx RAM Inference 738
Verilog Coding for Xilinx Single-port and Dual-port Block RAMs 738
Verilog Guidelines for Byte-Enable Block RAMs 741
Verilog Guidelines for Byte-Wide Write Enable RAMs 742
Virtex-5 RAM Verilog Examples 743

RAM Instantiation with SYNCORE 745

ROM Inference 746

Instantiating Black Boxes in Verilog 749

PREP Verilog Benchmarks 750

Hierarchical or Structural Verilog Designs 751
Using Hierarchical Verilog Designs 751
Creating a Hierarchical Verilog Design 751

Copyright © 2013 Synopsys, Inc.

synthesis Macro 753
text Macro 754

Verilog Attribute and Directive Syntax 758
Attribute Examples Using Verilog 2001 Parenthetical Comments 760

Chapter 10: SystemVerilog Language Support

Feature Summary 764
SystemVerilog Limitations 767

Unsized Literals 769

Data Types 769
Typedefs 770
Enumerated Types 771
Struct Construct 773
Union Construct 775
Static Casting 777

Arrays 778
Arrays 778
Arrays of Structures 780
Array Querying Functions 781

Data Declarations 781
Constants 782
Variables 782
Nets 783
Data Types in Parameters 784
Type Parameters 784

Operators and Expressions 786
Operators 786
Aggregate Expressions 787
Streaming Operator 789
Set Membership Operator 790
Set Membership Case Inside Operator 791
Type Operator 794

Procedural Statements and Control Flow 797
Do-While Loops 797
For Loops 798
Unnamed Blocks 798
Block Name on end Keyword 799
Unique and Priority Modifiers 799

Copyright © 2013 Synopsys, Inc.

Processes 800
always_comb 801
always_latch 803
always_ff 804

Tasks and Functions 805
Implicit Statement Group 805
Formal Arguments 805
endtask/endfunction Names 808

Hierarchy 809
Compilation Units 809
Packages 811
Port Connection Constructs 813
Extern Module 815

Interface 816
Interface Construct 816
Modports 823
Limitations and Non-Supported Features 823

System Tasks and System Functions 825
$bits System Function 825
Array Querying Functions 826

Generate Statement 827
Conditional Generate Constructs 829

Assertions 832
SVA System Functions 833

Keyword Support 836

Chapter 11: VHDL Language Support

Language Constructs 838
Supported VHDL Language Constructs 838
Unsupported VHDL Language Constructs 839
Partially-supported VHDL Language Constructs 840
Ignored VHDL Language Constructs 840

VHDL Language Constructs 840
Data Types 841
Declaring and Assigning Objects in VHDL 844
VHDL Dynamic Range Assignments 844
Null Ranges 845
Signals and Ports 846

Copyright © 2013 Synopsys, Inc.

Variables 848
VHDL Constants 848
Libraries and Packages 849
Operators 853
Large Time Resolution 855
VHDL Process 858
Common Sequential Statements 860
Concurrent Signal Assignments 862
Resource Sharing 865
Combinational Logic 865
Sequential Logic 865
Component Instantiation in VHDL 866
VHDL Selected Name Support 868
User-defined Function Support 871
Demand Loading 873

VHDL Implicit Data-type Defaults 874

VHDL Synthesis Guidelines 879
General Synthesis Guidelines 879
VHDL Language Guidelines 880
Model Template 881
Constraint Files for VHDL Designs 882
Creating Flip-flops and Registers Using VHDL Processes 883
Clock Edges 884
Defining an Event Outside a Process 885
Using a WAIT Statement Inside a Process 886
Level-sensitive Latches Using Concurrent Signal Assignments 887
Level-sensitive Latches Using VHDL Processes 888
Signed mod Support for Constant Operands 891

Sets and Resets 893
Asynchronous Sets and Resets 893
Synchronous Sets and Resets 894

VHDL State Machines 897
State Machine Guidelines 897
Using Enumerated Types for State Values 901
Simulation Tips When Using Enumerated Types 902
Asynchronous State Machines in VHDL 904

Hierarchical Design Creation in VHDL 906

Copyright © 2013 Synopsys, Inc.

Configuration Specification and Declaration 910
Configuration Specification 910
Configuration Declaration 914
VHDL Configuration Statement Enhancement 920

Scalable Designs 934
Creating a Scalable Design Using Unconstrained Vector Ports 934
Creating a Scalable Design Using VHDL Generics 935
Using a Scalable Architecture with VHDL Generics 936
Creating a Scalable Design Using Generate Statements 938

VHDL Guidelines for RAM Inference 940
RAM Inference in VHDL Designs 940
Limited RAM Resources 942
Additional Components 942
Multi-Port RAM Extraction 942
Synchronous READ RAMs 944
Multi-port RAM Extraction 946

VHDL Coding for Altera RAM Inference 948
Stratix, Arria and Cyclone Dual-Port VHDL RAM Example 948
VHDL Coding for StratixIII and Stratix IV RAMs with Control Signals 951
VHDL Coding for Cyclone III RAMs with Control Signals 951

VHDL Coding for Xilinx RAM Inference 952
VHDL Coding for Xilinx Single-port and Dual-port Block RAMs 952
VHDL Guidelines for Byte-Enable Block RAMs 954
VHDL Guidelines for Byte-Wide Write Enable RAMs 955

Instantiating RAMs with SYNCORE 957

ROM Inference 958

Instantiating Black Boxes in VHDL 961
Black-Box Timing Constraints 962

VHDL Attribute and Directive Syntax 963

VHDL Synthesis Examples 965
Combinational Logic Examples 965
Sequential Logic Examples 966

PREP VHDL Benchmarks 967

Copyright © 2013 Synopsys, Inc.

Chapter 12: VHDL 2008 Language Support

Features 969

Operators 970
- Logical Reduction Operators 970
- Condition Operator 970
- Matching Relational Operators 971
- Bit-string Literals 971

Unconstrained Data Types 973

Unconstrained Record Elements 975

Predefined Functions 976
- Generic Types 977

Packages 978
- New Packages 979
- Modified Packages 979
- Unsupported Packages/Functions 979
- Using the Packages 980

Generics in Packages 981

Context Declarations 981

Case-generate Statements 982

Matching case and select Statements 984

Else/elsif Clauses 985

Syntax Conventions 985
- All Keyword 985
- Comment Delimiters 986
- Extended Character Set 986

Chapter 13: Utilities

sdc2fdc Tcl Shell Command 988
- sdc2fdc Tcl Shell Command 988
- Examples of sdc2fdc Translation 988
- FPGA Design Constraint (FDC) File 989
- Troubleshooting 991

Altera qsf2sdc Utility 994
- QSF2SDC Syntax 994
- Supported QSF Constraints 995
- Example of Translated QSF Constraints 996

Copyright © 2013 Synopsys, Inc.

Altera SOPC2Syn Utility 998
Features of the sopc2syn Utility 998
Syntax Options for the sopc2syn Utility 999

Altera qsf2syn Utility 1003

Xilinx ucf2sdc Utility 1004
ucftosdc Tcl Command 1004
UCF Supported Constraints Summary 1005
Examples of Translated UCF Constraints 1006
Unsupported UCF Constraints 1016

Xilinx ise2syn Utility 1017

Xilinx Vivado Tcl Commands 1018
import_vivado_ip Command 1018
xdc2fdc Command 1020

Synplicity Archive Utility 1025

SYNCore FIFO Compiler 1026
Synchronous FIFOs 1026
FIFO Read and Write Operations 1027
FIFO Ports 1029
FIFO Parameters 1031
FIFO Status Flags 1033
FIFO Programmable Flags 1036

SYNCore RAM Compiler 1043
Single-Port Memories 1043
Dual-Port Memories 1045
Read/Write Timing Sequences 1050

SYNCore Byte-Enable RAM Compiler 1053
Functional Overview 1053
Read/Write Timing Sequences 1054
Parameter List 1057

SYNCore ROM Compiler 1058
Functional Overview 1058
Single-Port Read Operation 1059
Dual-Port Read Operation 1060
Parameter List 1061
Clock Latency 1062

Copyright © 2013 Synopsys, Inc.

SYNCore Adder/Subtractor Compiler 1064
Functional Description 1064
Adder 1065
Subtractor 1068
Dynamic Adder/Subtractor 1071

SYNCore Counter Compiler 1076
Functional Overview 1076
UP Counter Operation 1077
Down Counter Operation 1078
Dynamic Counter Operation 1078

Encryption Scripts 1081
Encryption and Decryption Methodologies 1081
The encryptP1735 Script 1082
The encryptIP Script 1090

Chapter 14: Timing Constraint Syntax

FPGA Timing Constraints 1100
create_clock 1102
create_generated_clock 1104
reset_path 1106
set_clock_groups 1108
set_clock_latency 1111
set_clock_route_delay 1113
set_clock_uncertainty 1114
set_datapathonly_delay 1116
set_false_path 1118
set_input_delay 1120
set_max_delay 1122
set_multicycle_path 1125
set_output_delay 1128
set_reg_input_delay 1131
set_reg_output_delay 1132
Naming Rule Syntax Commands 1132

Synplify-Style Timing Constraints (Legacy) 1135
define_clock 1136
define_clock_delay 1139
define_false_path 1140
define_input_delay 1143
define_multicycle_path 1145
define_output_delay 1149
define_path_delay 1151

Copyright © 2013 Synopsys, Inc.

define_reg_input_delay 1154
define_reg_output_delay 1155
set_datapathonly_delay 1156
Object Naming Syntax 1157

Chapter 15: FPGA Design Constraint Syntax

define_compile_point 1162
define_current_design 1163
define_io_standard 1164

Chapter 16: Attribute and Directive Syntax

How Attributes and Directives are Specified 1166
The SCOPE Attributes Tab 1166
Compiler Directives File Examples 1168

Summary of Attributes and Directives 1169
Attribute and Directive Summary Table (by Vendor) 1169
Attribute and Directive Summary (Alphabetical) 1170

Attribute/Directive Descriptions 1182
alsloc 1182
alspin 1185
alspreserve 1187
altera_io_powerup 1189
altera_logiclock_location 1191
altera_logiclock_size 1193
black_box_pad_pin 1195
black_box_tri_pins 1197
CLOCK_DEDICATED_ROUTE 1199
diff_term 1201
full_case 1203
loc 1206
loop_limit 1207
orca_padtype 1208
orca_props 1210
parallel_case 1212
pragma translate_off/pragma translate_on 1214
syn_allow_retiming 1216
syn_allowed_resources 1221
syn_assign_to_region 1230
syn_assign_to_slr 1231
syn_auto_insert_bufg 1233
syn_auto_partition 1234

Copyright © 2013 Synopsys, Inc.

syn_black_box 1235
syn_clean_reset 1239
syn_clock_priority 1241
syn_cp_use_fast_synthesis 1244
syn_diff_io 1245
syn_direct_enable 1248
syn_direct_reset 1254
syn_direct_set 1258
syn_disable_purifyclock 1262
syn_dspstyle 1264
syn_edif_bit_format 1280
syn_edif_name_length 1284
syn_edif_scalar_format 1286
syn_encoding 1305
syn_enum_encoding 1314
syn_expand_coreloc 1319
syn_fast_auto 1320
syn_force_pads 1327
syn_force_seq_prim 1328
syn_forward_io_constraints 1330
syn_fsm_correction 1332
syn_gatedclk_clock_en 1333
syn_gatedclk_clock_en_polarity 1335
syn_global_buffers 1339
syn_hier 1344
syn_highrel_ioconnector 1352
syn_insert_buffer (Microsemi) 1353
syn_insert_buffer (Lattice) 1358
syn_insert_buffer (Xilinx) 1360
syn_insert_pad 1376
syn_isclock 1380
syn_keep 1382
syn_loc 1388
syn_looplimit 1390
syn_macro 1392
syn_map_dffrs 1395
syn_maxfan 1397
syn_max_memsize_reg 1403
syn_multstyle 1404
syn_netlist_hierarchy 1413
syn_noarrayports 1420
syn_noclockbuf 1421

Copyright © 2013 Synopsys, Inc.

syn_noclockpad 1427
syn_no_compile_point 1429
syn_noprune 1431
syn_packer_effort_level 1439
syn_pad_type 1440
syn_pipeline 1442
syn_placer_effort_level 1447
syn_preserve 1448
syn_preserve_sr_priority 1452
syn_probe 1454
syn_radhardlevel 1462
syn_ramstyle 1467
syn_reduce_controlset_size (Xilinx) 1480
syn_reduce_controlset_size 1484
syn_reference_clock 1490
syn_rename_module 1493
syn_replicate 1494
syn_resources 1497
syn_romstyle (Achronix) 1500
syn_romstyle (Altera) 1502
syn_romstyle (Lattice) 1507
syn_romstyle Attribute (Lattice iCE65/iCE40) 1509
syn_romstyle Attribute (Xilinx) 1511
syn_rw_conflict_logic 1513
syn_safe_case 1517
syn_sharing 1518
syn_slow 1520
syn_srlstyle 1524
syn_state_machine 1529
syn_tco<*n*> *1533*
syn_tpd<*n*> *1536*
syn_tristate 1539
syn_tristatetomux 1540
syn_tsu<*n*> *1543*
syn_unconnected_inputs 1546
syn_unique_inst_module 1551
syn_use_carry_chain (Lattice) 1552
syn_use_carry_chain (Lattice) 1553
syn_use_xilinx_placement 1558
syn_useenables 1559
syn_useioff (Altera) 1564
syn_useioff (Lattice) 1570

Copyright © 2013 Synopsys, Inc.

syn_useioff (Lattice iCE65/iCE40) 1576
syn_useioff (Xilinx) 1579
syn_user_instance 1585
syn_vote_loops 1591
translate_off/translate_on 1593
xc_area_group 1595
xc_clockbuftype 1597
xc_fast 1599
xc_fast_auto 1603
xc_global_buffers 1605
xc_isgsr 1606
xc_loc 1608
xc_map 1610
xc_padtype 1611
xc_props 1613
xc_pullup/xc_pulldown 1620
xc_rloc 1622
xc_slow 1624
xc_use_keep_hierarchy 1625
xc_use_rpms 1627
xc_use_timespec_for_io 1628
xc_uset 1630

Summary of Global Attributes 1632

Chapter 17: Batch Commands and Scripts

Introduction to Tcl 1636
Tcl Conventions 1636
Tcl Scripts and Batch Mode 1636

Tcl Commands for Synthesis 1637
add_file 1639
add_folder 1645
assign_to_region 1645
command_history 1646
constraint_file 1648
create_region 1649
define_link 1650
dh_module_sources 1651
generate_instance_constraints 1652
get_env 1653
get_option 1653
hdl_define 1654

hdl_param 1654
impl 1656
job 1657
launch_system_designer 1658
open_design 1659
open_file 1660
partdata 1660
program_terminate 1661
program_version 1662
project 1662
project_data 1669
project_file 1670
project_folder 1671
recording 1673
report_clocks 1673
run_config 1674
run_tcl 1676
sch_sim_disp 1676
sch_sim_load 1677
sch_sim_watch 1679
set_option 1679
slice_primitive 1714
sxml2pxml Tcl Command 1716
Technology-Specific Tcl Commands 1717

synhooks File Syntax 1719

Log File Commands 1721
log_filter Tcl Command 1721
log_report Tcl Command 1722

High-Reliability Tcl Commands 1723
syn_connect 1723
syn_create_err_net 1724

VIF Commands 1726
vif_add_file 1726
vif_add_library 1727
vif_set_constant 1727
vif_set_equiv 1728
vif_set_fixed_gated_clock 1728
vif_set_fsm 1729
vif_set_fsmreg 1729
vif_set_map_point 1729
vif_set_merge 1730

Copyright © 2013 Synopsys, Inc.

vif_set_port_dir 1730
vif_set_result_file 1731
vif_set_state_map 1731
vif_set_top_module 1732
vif_set_transparent 1732

Netlist Editing Commands 1733
Command Listing 1734
connect_net 1734
copy_view 1736
create_cell 1737
create_cgm 1738
create_instance 1739
create_net 1740
create_port 1741
create_process_hierarchy 1742
define_current_view 1743
disconnect_connector 1744
disconnect_net 1745
get_net 1747
insert_buffer 1748
load_database 1750
optimize_netlist 1752
pop_feedthroughs 1753
propagate_constants 1754
remove_buffer 1755
remove_instance 1756
remove_net 1757
remove_port 1758
reset_generate 1759
set_property 1760
swap_instance 1761
tie_net 1762
tie_pin 1764

Tcl Script Examples 1766
Using Target Technologies 1766
Different Clock Frequency Goals 1767
Setting Options and Timing Constraints 1768
Bottom-up Synthesis 1770

Copyright © 2013 Synopsys, Inc.

Chapter 18: Tcl Find, Expand, and Collections

Tcl find Command 1774
Tcl Find Syntax 1775
Tcl Find Syntax Examples 1781

Tcl Find -filter Command 1785

Tcl expand Command 1791

Collection Commands 1794
c_diff 1795
c_info 1796
c_intersect 1796
c_list 1797
c_print 1798
c_symdiff 1798
c_union 1799
get_prop 1800
define_collection 1801
define_scope_collection 1801
set 1802

Object Query Commands 1803
all_clocks 1805
all_inputs 1805
all_outputs 1805
all_registers 1806
get_cells 1806
get_clocks 1808
get_nets 1810
get_pins 1811
get_ports 1813

Synopsys Standard Collection Commands 1815
add_to_collection 1816
append_to_collection 1818
copy_collection 1820
foreach_in_collection 1821
get_object_name 1823
index_collection 1824
remove_from_collection 1826
sizeof_collection 1828

Copyright © 2013 Synopsys, Inc.

Appendix A: Designing with Achronix

Achronix Components 1830
Achronix LRAM640 Inference 1830
Achronix BRAM80K Inference 1834
Achronix BMACC56 Inference 1835
Achronix ROMs 1837
RAM and ROM Initialization 1837
Achronix Register Inference 1837
Register Initialization for Achronix 1838

Output Files and Forward Annotation 1850
Net Attributes in RTL 1850
Forward Annotation of RTL Attributes to the Netlist 1853

Achronix Technology Support 1855
Device Mapping Options 1855
Tcl set_option Command 1856
Disable I/O Insertion 1859
Time Borrowing 1859
Compile Point Remapping in Achronix Designs 1859
Tcl project Command 1860

Achronix Attribute and Directive Summary 1861

Appendix B: Designing with Altera

Basic Support for Altera Designs 1866

Input for Altera Designs 1876

Altera Components 1878

Altera RAMs and ROMs 1882

Altera LPMs 1898

DSP Blocks for Stratix, Arria, and Cyclone 1904

Altera Optimizations 1910

Altera Reports 1917

Forward-annotation to Altera Tools 1920
Altera Max+Plus II Constraint Files 1921

Integration with Altera Tools and Processes 1923
Altera Place-and-route Tools 1923
Quartus II Incremental Compilation Flow 1924
Clearbox Support in Synplify Pro and Synplify Premier 1924

Copyright © 2013 Synopsys, Inc.

Greybox Support in Synplify Pro and Synplify Premier 1926

Stratix, Arria, and Cyclone Families 1927
Stratix, Arria, and Cyclone Device Mapping Options 1927
Stratix, Arria, and Cyclone File Format Options 1930

MAX II and MAX V Families 1931
Device Mapping Options for MAX II and MAX V 1931
MAX II and MAX V File Format Options 1933

MAX Family 1935
Device Mapping Options for Altera MAX 1935
File Format Options for MAX 1937

set_option Command for Altera Architectures 1937
project Command for Altera Architectures 1943

Altera Attribute and Directive Summary 1944

Appendix C: Designing with Lattice

Lattice Technology Support 1950
Supported Device Families 1950

IP Encryption Flow for Lattice 1951

Lattice Features 1952
Block RAM Support 1952
SYNCORE RAMs 1962
Macros and Black Boxes 1963
Programmable I/O Cell Latches 1963
Initial Value Support for Registers 1963
Hierarchical Attribute Support for Module Generation 1964
Selective Clock Enable Inference 1965
Lattice iCE65 and iCE40 Components 1965
Forward Annotation 1973
Supported Timing Constraints 1973
Synthesis Reports 1974

Lattice Constraints, Attributes, and Options 1976
Constraints and Attributes 1976
Disable I/O Insertion Globally or Port by Port 1977
I/O Buffer Support 1978

Device Mapping Options 1979
Fanout Guide 1980
Disable I/O Insertion 1980
GSR Resource 1980

Copyright © 2013 Synopsys, Inc.

Map Logic to Macrocells 1981
Maximum Cell Fanin 1981
Maximum Terms/Macrocell 1981
Percentage of Design to Optimize for Timing 1982
Compile Point Remapping in Lattice Designs 1982

Design Options 1983
FSM Compiler 1983
Resource Sharing 1984
Pipelining 1984
Retiming 1984
Gated and Generated Clocks 1985

Implementation Results 1985

ORCA Devices 1986
ORCA Device Mapping Options 1986
ORCA Attributes 1986
ORCA set_option Tcl Command 1986

LatticeECP/EC and Later Devices 1989
LatticeECP/EC and Later Device Mapping Options 1989
LatticeECP/EC and Later Attributes 1989
LatticeECP/EC and Later set_option Tcl Command 1990

LatticeSC/SCM and LatticeXP2/XP Devices 1993
LatticeSC/SCM and LatticeXP2/XP Device Mapping Options 1993
LatticeSC/SCM and LatticeXP2/XP Attributes 1993
LatticeSC/SCM and LatticeXP2/XP set_option Tcl Command 1994

MachXO and Platform Devices 1997
MachXO and Platform Manager Device Mapping Options 1997
MachXO and Platform Manager Attributes 1997
MachXO and Platform Manager set_option Tcl Command 1998

ispGAL, ispGDX, and ispLSI Devices 2001
ispGAL, ispGDX, and ispLSI Device Mapping Options 2001
ispGAL, ispGDX, and ispLSI Attributes 2001
ispGAL, ispGDX, and ispLSI set_option Command 2001

ispXPGA Devices 2004
ispXPGA Device Mapping Options 2004
ispXPGA Attributes 2004
ispXPGA set_option Tcl Command 2004
ispXPGA LUT Function Support 2006

Copyright © 2013 Synopsys, Inc.

ispXPLD5000MX and ispLSI5000VE Devices 2007
ispXPLD5000MX and ispLSI5000VE Device Mapping Options 2007
ispXPLD5000MX and ispLSI5000VE Attributes 2007
ispXPLD5000MX and ispLSI5000VE set_option Tcl Command 2008
ispXPLD Selective Carry-Chain Inferencing 2010

ispMACH and MACH Devices 2011
ispMACH and MACH Device Mapping Options 2011
ispMACH and MACH Attributes 2011
ispMACH and MACH set_option Tcl Command 2011

Lattice iCE65 and iCE40 Devices 2014
iCE65 and iCE40 Device Mapping Options 2014
iCE65 and iCE40 set_option Tcl Command 2015

set_option Command for Lattice Architectures 2017

Lattice Attribute and Directive Summary 2021

Appendix D: Designing with Microsemi

Basic Support for Microsemi Designs 2028
Microsemi Device-specific Support 2028
Microsemi Features 2029
Synthesis Constraints and Attributes for Microsemi 2029

Microsemi Components 2031
Macros and Black Boxes in Microsemi Designs 2031
DSP Block Inference 2033
Microsemi RAM Implementations 2036
Instantiating RAMs with SYNCORE 2041

Output Files and Forward-annotation for Microsemi 2042
Forward-annotating Constraints for Placement and Routing 2042
Synthesis Reports 2043
Point-to-Point Timing Analysis 2044

Optimizations for Microsemi Designs 2044
The syn_maxfan Attribute in Microsemi Designs 2045
Promote Global Buffer Threshold 2045
I/O Insertion 2047
Number of Critical Paths 2047
Retiming 2047
Update Compile Point Timing Data Option 2048
Operating Condition Device Option 2049
Radiation-tolerant Applications 2051

Copyright © 2013 Synopsys, Inc.

Integration with Microsemi Tools and Flows 2053
Compile Point Synthesis 2053
Microsemi Place-and-Route Tools 2054

Microsemi Device Mapping Options 2054

Microsemi Tcl set_option Command Options 2056

Microsemi Attribute and Directive Summary 2059

Appendix E: Designing with Xilinx

Basic Support for Xilinx Designs 2064

Input for Xilinx Designs 2071

Xilinx Components 2072
Macros and Black Boxes 2073
Xilinx Registers 2076
DDR Registers 2076
Dynamic Shift Register Lookup (SRL) Table 2079
Latch Mapping 2080
Global Set/Resets and Startup Blocks 2080
Relative Locations 2081
DSP Components in Virtex Designs 2081
Advanced LUT Combining 2082
Wide Adders in Virtex-5 and Virtex-6 Designs 2082
Control Sets in Spartan/Virtex Architectures 2083

Xilinx RAMs and ROMs 2084
ROM Mapping 2084
Xilinx RAMs 2084

Xilinx I/O Support 2096

Xilinx Optimizations 2099
Fanout Limits in Xilinx Designs 2099
Pipelining in Xilinx Designs 2101
Retiming in Xilinx Designs 2101
Xilinx Gated and Generated Clocks 2102
Sequential Optimizations in Xilinx Designs 2102
Verification Mode in Xilinx Designs 2103
Compile Point Remapping in Xilinx Designs 2104
Clock Skew Estimation for Xilinx Designs 2105
Power Optimization for Xilinx Designs 2105

Copyright © 2013 Synopsys, Inc.

Output Files and Forward-annotation for Xilinx 2107
Xilinx Resource Usage Reports 2107
Xilinx Net Buffering Report 2108
Virtex-5 and Virtex-6 Detailed Clock Path Report 2109
Custom Timing Reports for Xilinx 2109
Forward-annotation of Xilinx Constraints 2110
Xilinx Constraint Files 2110

Integration with Xilinx Tools and Flows 2118

Virtex and Spartan-II and Later Technologies 2121
Virtex and Spartan-II and Later Device Mapping Options 2122
Virtex and Spartan-II and Later set_option Tcl Command 2124
Virtex and Spartan-II and Later project Tcl Command 2130
Virtex Components 2130

Xilinx CPLDs 2131
Xilinx CPLD Device Mapping Options 2131
Xilinx CPLD set_option Tcl Command 2132
Xilinx CPLD project Command 2135

Xilinx Attribute and Directive Summary 2136

Copyright © 2013 Synopsys, Inc.

CHAPTER 1

Product Overview

This document is part of a set that includes reference and procedural information for the Synplify®, Synplify Pro®, and Synplify® Premier FPGA synthesis tools. The reference manual details the synthesis tool user interface, commands, and features. The user guide contains "how-to" information, emphasizing user tasks, procedures, design flows, and results analysis.

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

The following provide an introduction to the synthesis tools.

- Synopsys FPGA and Prototyping Products, on page 38
- Overview of the Synthesis Tools, on page 42
- Starting the Synthesis Tool, on page 52
- Logic Synthesis Overview, on page 56
- Getting Help, on page 59

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.

Identify Tool Set

The Identify® tool set allows you to instrument and debug an operating FPGA directly in the source RTL code. The Identify software is used to verify your design in hardware as you would in simulation, however much faster and with in-system stimulus. Designers and verification engineers are able to navigate the design graphically and instrument signals directly in RTL with which they are familiar, as probes or sample triggers. After synthesis, results are viewed embedded in the RTL source code or in a waveform. Design iterations are rapidly performed using incremental place and route. Identify software is closely integrated with synthesis and routing tools to create a seamless development environment.

Synphony Model Compiler

Synphony Model Compiler is a language and model-based high-level synthesis technology that provides an efficient path from algorithm concept to silicon. Designers can construct high-level algorithm models from math languages and IP model libraries, then use the Synphony Model Compiler engine to synthesize optimized RTL implementations for FPGA and ASIC architectural exploration and rapid prototyping. In addition, Synphony Model Compiler generates high performance C-models for system validation and early software development in virtual platforms. Key features for this product include:

- MATLAB Language Synthesis
- Automated Fixed-point Conversion Tools
- Synthesizable Fixed-point High Level IP Model Library
- High Level Synthesis Optimizations and Transformations
- Integrated FPGA and ASIC Design Flows
- RTL Testbench Generation
- C-model Generation for Software Development and System Validation

Copyright © 2013 Synopsys, Inc.

Rapid Prototyping

The Certify® and Identify products are tightly integrated with the HAPS™ and ChipIT® hardware tools.

Certify Product

The Certify software is the leading implementation and partitioning tool for ASIC designers using FPGA-based prototypes to verify their designs. The tool provides a quick and easy method for partitioning large ASIC designs into multi-FPGA prototyping boards. Powerful features allow the tool to adapt easily to existing device flows, therefore, speeding up the verification process and helping with the time-to-market challenges. Key features include the following:

- Graphical User Interface (GUI) Flow Guide
- Automatic/Manual Partitioning
- Synopsys Design Constraints Support for Timing Management
- Multi-core Parallel Processing Support for Faster Runtimes
- Support for Most Current FPGA Devices
- Industry Standard Synplify Premier Synthesis Support
- Compatible with HAPS-5x and HAPS-6x Boards Including HSTDM

Copyright © 2013 Synopsys, Inc.

Overview of the Synthesis Tools

This section introduces the technology, main features, and user interface of the FPGA synthesis tools: Synplify, Synplify Pro, Synplify Premier, and Synplify Premier with Design Planner. See the following for details:

- Common Features, on page 42
- Synplify Pro and Synplify Premier Features, on page 43
- Synplify Premier Features, on page 44
- BEST Algorithms, on page 45
- Graphic User Interface, on page 45
- Projects and Implementations, on page 51

Common Features

The Synopsys FPGA synthesis tools have the following built-in features:

- The HDL Analyst® RTL analysis and debugging environment, a graphical tool for analysis and crossprobing. See *RTL View*, on page 90, *Technology View*, on page 91, and *Analyzing with HDL Analyst and FSM Viewer*, on page 545 in the *User Guide*. This feature is an option with the Synplify software.
- The Text Editor window, with a language-sensitive editor for writing and editing HDL code. See *Text Editor View*, on page 97.
- The SCOPE® (Synthesis Constraint Optimization Environment®) tool, which provides a spreadsheet-like interface for managing timing constraints and design attributes. See *SCOPE User Interface*, on page 470.
- FSM Compiler, a symbolic compiler that performs advanced finite state machine (FSM) optimizations. See *FSM Compiler*, on page 105.
- Integration with the Identify RTL Debugger.

Copyright © 2013 Synopsys, Inc.

Synplify Pro and Synplify Premier Features

The following features are specific to the Synplify Pro, Synplify Premier, and Synplify Premier with Design Planner tools.

- FSM Explorer, which tries different state machine optimizations before picking the best implementation. See *FSM Explorer*, on page 107.
- The FSM Viewer, for viewing state transitions in detail. See *FSM Viewer Window*, on page 95.
- The Tcl window, a command line interface for running TCL scripts. See *Tcl Script Window*, on page 85.
- The Timing Analyst window, which allows you to generate timing schematics and reports for specified paths for point-to-point timing analysis.
- Place-and-Route implementation(s) to automatically run placement and routing after synthesis. You can run place-and-route from within the tool or in batch mode. This feature is supported for the latest Microsemi, Altera and Xilinx technologies (see *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide*).
- Other special windows, or *views*, for analyzing your design, including the Watch Window and Message Viewer (see *The Project View*, on page 62).
- Certain optimizations, like pipelining and retiming are only available with these tools.
- Advanced analysis features like crossprobing and probe point insertion.

Copyright © 2013 Synopsys, Inc.

Synplify Premier Features

The following features are specific to the Synplify Premier tool:

- Fast Synthesis – use this logic synthesis flow to significantly reduce synthesis runtimes. When quick turnaround times are important, there may be a trade-off in quality of results (QoR).
- Enhanced Logic Synthesis – a logic synthesis flow with additional optimizations during logic synthesis and which provides better QoR than basic logic synthesis. Enhanced Optimization is turned on by default. This option is only available for the latest Altera and Xilinx technologies.
- Graph-based physical synthesis with detailed placement available with or without a design plan. This is currently available for the latest Altera and Xilinx technologies.
- Backannotation for place-and-route data. This feature is supported for Xilinx technologies (see *Backannotating Place-and-Route Data*, on page 326).
- Physical optimizations that use physical design characteristics like placement and interconnect delay to affect the actual topology of the circuit. Design plans are used to control the optimizations.
- Physical synthesis based on the design plan lets you perform physical synthesis optimizations based on the coarse placement. This option is only available for the latest Altera and Xilinx families with the Design Planner option, which is a separately licensed option to the Synplify Premier product.
- Expanded HDL Analyst capabilities including support for physical regions.
- Physical Analyst tool to analyze the placement and global routing. (see *Physical Analyst User Interface*, on page 566).
- DesignWare Support – the full Synopsys DesignWare foundation library is supported.

Copyright © 2013 Synopsys, Inc.

BEST Algorithms

The Behavior Extracting Synthesis Technology (BEST™) feature is the underlying proprietary technology that the synthesis tools use to extract and implement your design structures.

During synthesis, the BEST algorithms recognize high-level abstract structures like RAMs, ROMs, finite state machines (FSMs), and arithmetic operators, and maintain them, instead of converting the design entirely to the gate level. The BEST algorithms automatically map these high-level structures to technology-specific resources using module generators. For example, the algorithms map RAMs to target-specific RAMs, and adders to carry chains. The BEST algorithms also optimize hierarchy automatically.

Graphic User Interface

With the exception of the Synplify tool, the Synopsys FPGA family of products share a common graphical user interface (GUI), in order to ensure a cohesive look and feel across the different products. The following figures show the graphical user interfaces for the Synplify, Synplify Pro, and Synplify Premier tools.

Copyright © 2013 Synopsys, Inc.

Implementation Results View
Project Tree View
Project View
Menus
Toolbars
Synplify G-2012.09 - [<no projects loaded>]
File Edit View Project Import Run Analysis HDL-Analyst Options Window Tech-Support Web Help
Synplify
SYNOPSYS
Predictable Success
Frequency (MHz)
1
Symbolic FSM Compiler
Resource Sharing
Source Files
Add
Change
Edit
Implementation Directory
Name
Size
Type
Result File
Change
Target
Change
RUN
View Log
Cancel
Ready
<No projects>
Tab to access Project view
Buttons
Status

Copyright © 2013 Synopsys, Inc.

Menus
Project Management View
Status
Implementation Results View
Toolbars
Buttons
Tabs to access main views
Tabs to access Tcl Script and Messages Window
Output Window
Watch Window
Synplify Premier G-2012.09 - [C:/software/tutorial/tutorial.prj]
File Edit View Project Import Run Analysis HDL-Analyst Options Window Tech-Support Web Help
Run
Synplify Premier
Ready
Demos & Examples
Search SolvNet
Open Project...
Close Project
Add File...
Change File...
Add Implementation...
Implementation Options...
Add P&R Implementation
View Log
Frequency (MHz):
Project Files
Design Hierarchy
Altera STRATIX V : 5SGXEA4H2 : FC35 : -1
[tutorial] - C:\software\tutorial\tutorial.prj
VHDL
Verilog
Logic Constraints (FDC)
rev_1
Project Status
Implementation Directory
Process View
C:\software\tutorial\rev_1
Name
Size
Type
backup
coreip
dm
physical_plus
synlog
syntmp
synwork
verif
eight_bit_uc.edf
eight_bit_uc.fse
eight_bit_uc.htm
tutorial.prj
Information
21 warnings, 43 notes
Find:
Set Filter...
Apply Filter
GroupCommon ID's
Type
ID
Message
Source Location
Running in 64-bit mode
BN225
Writing default property annotation file C:\software...
BW103
Synopsys Constraint File time units using default val...
TCL Script
Messages
Watch
Log Parameter
Worst Slack
clock - Estimated Frequency
clock - Requested Frequency
Log Watch

Copyright © 2013 Synopsys, Inc.

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

For information about...	See...
Project window	The Project View, on page 62
RTL view	*RTL View*, on page 90
Technology view	*Technology View*, on page 91
Text Editor view	*Text Editor View*, on page 97
Synplify Pro and Synplify Premier only	
FSM Viewer window	*FSM Viewer Window*, on page 95
Tcl window	*Tcl Script Window*, on page 85
Watch Window	*Watch Window*, on page 81
Synplify Premier Design Planner	*Design Planner User Interface*, on page 536
SCOPE spreadsheet	SCOPE User Interface, on page 470
Other views and windows	The Project View, on page 62
Menu commands and their dialog boxes	*Menus*, on page 139
Toolbars	*Toolbars*, on page 114
Buttons	*Buttons and Options*, on page 140
Context-sensitive popup menus and their dialog boxes	*Popup Menus*, on page 365
Online help	Use the F1 keyboard shortcut or click the Help button in a dialog box. See *Help Menu*, on page 362, for more information.

Copyright © 2013 Synopsys, Inc.

Projects and Implementations

Projects and implementations are available for all synthesis tools.

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

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

A *Place and Route implementation*, located in the project implementation hierarchy, is created automatically for supported technologies. To view the P&R implementation, select the plus sign to expand the project implementation hierarchy. To add, remove, or set options, right-click on the P&R implementation. You can create multiple P&R implementations for each project implementation. Select a P&R implementation to activate it.

Copyright © 2013 Synopsys, Inc.

Starting the Synthesis Tool

This section describes starting the synthesis tool in interactive and batch mode. Before you can start the synthesis tool, you must install it and set up the software license appropriately. How you start the tool depends on your environment. For details, see the installation instructions for the tool.

Starting the Synthesis Tool in Interactive Mode

You can start interactive use of the synthesis tool in any of the following ways:

- To start the synthesis tool from the Microsoft® Windows® operating system, choose
  - Start->Programs->Synopsys->Synplify *version*
  - Start->Programs->Synopsys->Synplify Pro *version*
  - Start->Programs->Synopsys->Synplify Premier *version*
  - Start->Programs->Synopsys->Synplify Premier DP *version*
- To start the tool from a DOS command line, specify the executable:
  - *installDirectory*\bin\synplify.exe
  - *installDirectory*\bin\synplify_pro.exe
  - *installDirectory*\bin\synplify_premier.exe
  - *installDirectory*\bin\synplify_premier_dp.exe

  The executable name is the name of the product followed by an `exe` file extension.
- To start the synthesis tool from a Linux platform, type the appropriate command at the system prompt:
  - `synplify`
  - `synplify_pro`
  - `synplify_premier`
  - `synplify_premier_dp`

For information about using the synthesis tool in batch mode, see *Starting the Tool from the Command Line*, on page 53.

Starting the Tool from the Command Line

The command to start the synthesis tool from the command line includes a number of command line options. These options control tool action on startup and, in many cases, can be combined on the same command line. To start the synthesis tool, use the following syntax:

toolName [**-***option ...*] [*projectFile*]

In the syntax statement, *toolName* can be any of the synthesis tools:

- synplify
- synplify_pro
- synplify_premier
- synplify_premier_dp

ProjectFile is the name of the project (`prj`) file to open and, if omitted, defaults to the last project file opened. *Option* is any of the following command line options:

-batch *projectFile* |*tclScriptName*
-compile
-evalhostid
-help
-history *historyLogFilename*
-identify_dir
-impl *implementationName*
-license_release
-licensetype *licenseFeatureName*
-license_wait *licenseWaitTime*
-log *logFilename*
-runall
-shell
-tcl *projectFile*|*tclScriptName*
-tclcmd *tclCommandName*
-verbose_log
-version

The following table describes the command line options.

Copyright © 2013 Synopsys, Inc.

Option	Description
toolname	Starts the synthesis tool. Can be any of the synthesis tools: • synplify • synplify_pro • synplify_premier • synplify_premier_dp *projectFile* is the name of the project (`prj`) file to open and, if omitted, defaults to the last project file opened.
-batch	Starts the synthesis tool in batch mode from the specified project or Tcl file without opening the Project window. This option is not available for the Synplify tool or Synplify Pro tool with node-locked licenses.
-compile	Compiles the project, but does not map it.
-evalhostid	Reports host ID for node-locked and floating licenses.
-help	Lists available command line options and descriptions.
-history	Records all Tcl commands and writes them to the specified history log file when the command exits.
-Identify_dir	Specifies the location of the Identify installation directory for launching the Identify tool set. The installation path specified appears in the Configure Identify Launch dialog box (Options->Configure Identify Launch).
-impl	Runs only the specified implementation. You can use this option in conjunction with the -batch keyword (the Synplify tool supports only a single implementation).
-license_release	Releases FPGA synthesis licenses for a session after the place-and-route job is launched. The software allows place and route to continue running even after exiting the synthesis tool so that it does not consume an FPGA license. This command option must be run in batch mode. Specify the following command: `toolName -batch -license_release` For details, see *Releasing the Synthesis License During Place and Route*, on page 845.
-licensetype	Specifies a license if you work in an environment with multiple Synopsys FPGA licenses. You can use this option in conjunction with the -batch keyword.

Option	Description
-license_wait	Specifies how long to wait for a Synopsys FPGA license. License queuing allows you to wait until a license becomes available or specify a wait time in seconds. You can use this option in conjunction with the -batch keyword. For details, see *License Queuing*, on page 1134.
-log	Writes all output to the specified log file.
-runall	Runs all the implementations in the project file (the Synplify tool supports only a single implementation).
-shell	Starts synthesis tool in shell mode. Note: The FPGA synthesis tools only support the -shell option on UNIX and Linux platforms.
-tcl	Starts the synthesis tool in the graphical user interface using the specified project or Tcl file.
-tclcmd	Specifies Tcl command to be executed on startup.
-verbose_log	Writes messages to stdout.log in verbose mode.
-version	Reports version of specified synthesis tool.

Copyright © 2013 Synopsys, Inc.

Logic Synthesis Overview

When you run the synthesis tool, it performs *logic synthesis*. This consists of two stages:

- logic compilation (HDL language synthesis) and optimization
- technology mapping

Logic compilation and optimization: The synthesis tool first compiles input HDL source code, which describes the design at a high level of abstraction, to known structural elements. Next, it optimizes the design, making it as small as possible to improving circuit performance. These optimizations are technology independent.

Technology mapping: During this stage, the tool optimizes the logic for the target technology, by mapping it to technology-specific components. It uses architecture-specific techniques to perform additional optimizations. Finally, it generates a design netlist for placement and routing.

Physical synthesis: Run logic synthesis first as a good starting point for the physical synthesis flow. Then, include placement locations for all I/O pins before you run physical synthesis.

Synthesizing Your Design

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

You perform the following actions to synthesize your design. For detailed information, see the Tutorial.

1. Access your design project: open an existing project or create a new one.
2. Specify the input source files to use. Right-click the project name in the Project view, then choose Add Source Files.
   - Select the desired Verilog, VHDL, or IP files in formats such as EDIF, NGC, and NGO, then click OK. (See the examples in the directory *installation_dir*/examples, where *installation_dir* is the directory where the product is installed.)
   - You can also add source files in the Project view by dragging and dropping them there from a Windows® Explorer folder (Microsoft® Windows® operating system only).
   - *Top-level file:* The last file compiled is the top-level file. You can designate a new top-level file by moving the desired file to the bottom of the source files list in the Project view, or by using the Implementation Options dialog box.
3. Add design constraints. Use the SCOPE spreadsheet to assign system-level and circuit-path timing constraints that can be forward-annotated.

   See *SCOPE Tabs,* on page 471, for details on the SCOPE spreadsheet.
4. Choose Project->Implementation Options, then define the following:
   - Target architecture and technology specifications
   - Optimization options and design constraints
   - Outputs

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

Copyright © 2013 Synopsys, Inc.

5. Synthesize the design by clicking the Run button.

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

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

6. After synthesis, do one of the following:
   - If there were no synthesis warnings or error messages (Done!), analyze your results in the RTL and Technology views. You can then resynthesize with different implementation options, or use the synthesis results to simulate or place-and-route your design.
   - If there were synthesis warnings (Done (warnings)) or error messages (Errors!), check them in the log file. From the log file, you can jump to the corresponding source code or display information on the specific error or warning. Correct all errors and any relevant warnings and then rerun synthesis.

Copyright © 2013 Synopsys, Inc.

Getting Help

Before calling Synopsys SolvNet Support, look through the documented for information. You can access the information online from the Help menu, or refer to the corresponding manual. The following table shows you how the information is organized.

Finding Information

For help with...	Refer to the...
How to...	*User Guide* and various application notes available on the Synplicity support web site
Flow information	*User Guide* and various application notes available on the Synopsys SolvNet support web site
FPGA Implementation Tools	Synopsys Web Page (Web->FPGA Implementation Tools menu command from within the software)
Synthesis features	*User Guide* and *Reference Manual*
Language and syntax	*Reference Manual*
Attributes and directives	*Reference Manual*
Tcl language	Online help (Help->Tcl Help)
Synthesis Tcl commands	*Reference Manual* or type help followed by the command name in the Tcl window (not available in Synplify)
Using tool-specific features and attributes	*User Guide*
Error and warning messages	Click on the message ID code

Contacting Customer Support

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

- Support at https://solvnet.synopsys.com
- Call us at +1 800 245-8005

Copyright © 2013 Synopsys, Inc.

CHAPTER 2

User Interface Overview

This chapter presents tools and technologies that are built into the Synopsys FPGA synthesis software to enhance your productivity.

This chapter describes the following aspects of the graphical user interface (GUI):

- The Project View, on page 62
- The Project Results View, on page 69
- Other Windows and Views, on page 80
- FSM Compiler, on page 105
- FSM Explorer, on page 107
- Using the Mouse, on page 107
- User Interface Preferences, on page 112
- Toolbars, on page 114
- Keyboard Shortcuts, on page 123
- Buttons and Options, on page 131

Copyright © 2013 Synopsys, Inc.

The Project View

The Project View is the main interface to the tool. The Project view consists of a Project Management View on the left and a Project Results View on the right. The interface and available functionality varies, depending on the synthesis tool you use. See the following for an overview:

- Synplify Pro and Synplify Premier Project View, on page 62
- Synplify Project View, on page 64

Synplify Pro and Synplify Premier Project View

The Project Management view is on the left side of the window, and is used to create or open projects, create new implementations, set device options, and initiate design synthesis. You can use it to manage and synthesize hierarchical designs. The Project Results view is on the right.

The following figure shows the main parts of the interface. Additional detail about the Synplify Pro and Synplify Premier project view is described here:

- Synplify Pro and Synplify Premier Project Management View, on page 66
- The Project Results View

Copyright © 2013 Synopsys, Inc.

The Synplify Pro and Synplify Premier Project view has the following main parts:

Project View Interface	Description
Status	Displays the tool name or the current status of the synthesis job that is running. Clicking in this area displays additional information about the current job.

Copyright © 2013 Synopsys, Inc.

Project View Interface	Description
Buttons and options	Allow immediate access to some of the more common commands. See *Buttons and Options*, on page 131 for details.
Hierarchical Project Management view	Lists the projects and implementations, and their associated HDL source files and constraint files. There are two tabs with different views to facilitate working with hierarchical designs. • Project Files Tab • Design Hierarchy Tab
Implementation Results view	Lists the result of the synthesis runs for the implementations of your design. You can only view one set of implementation results at a time. Click an implementation in the Project view to make it active and view its result files. The Project Results view includes the following: • Project Status Tab—provides an overview of the project settings and at-a-glance summary of synthesis messages and reports. • Implementation Directory—lists the names and types of the result files, and the dates they were last modified. • Process View—gives you instant visibility to the synthesis and place-and-route job flows. See *The Project Results View*, on page 69 for more information.

Synplify Project View

The Project Management view is on the left side of the window, and is used to create or open projects, create new implementations, set device options, and initiate design synthesis. The Project Results view is on the right. The Synplify tool does not include support for hierarchical designs.

The Synplify Project view has the following main parts:

Project View Interface	Description
Status	Displays the current status of the synthesis job that is running. Clicking in this area displays additional information about the current job (see *Job Status Command*, on page 243).
Buttons and options	Allow immediate access to some of the more common commands. See *Buttons and Options*, on page 131 for details.
Project Management view	Lists the projects and implementations, and their associated HDL source files and constraint files. See *Projects and Implementations*, on page 51 for details.
Implementation Results view	Lists the result of the synthesis runs for the implementations of your design.

Copyright © 2013 Synopsys, Inc.

To customize the Project view display, use the Options->Project View Options command (*Project View Options Command*, on page 335).

Synplify Pro and Synplify Premier Project Management View

The Project Management view is on the left side of the window, and is used to create or open projects, create new implementations, set device options, and initiate design synthesis. The graphical user interface (GUI) lets you manage hierarchical designs that can be synthesized independently and imported back to the top-level project in a team design flow. The following figure shows the Project view as it appears in the Synplify Pro and Synplify Premier interfaces.

The Synplify Pro and Synplify Premier tools provide hierarchical management support for large designs. The tool lets you manage hierarchical projects in a team design flow, where you have independent hierarchical subprojects. For information on working with hierarchical projects, see *Hierarchical Project Management Flows*, on page 86 *in the User Guide*.

The Project view contains two tabs with different views of the design that help you manage hierarchical projects:

- Project Files Tab
- Design Hierarchy Tab

Both tabs in this view have right-click popup menu commands for managing design files and hierarchy. For descriptions of these commands, see *Synplify Pro and Synplify Premier Project Management Commands*, on page 376.

Project Files Tab

The Project Files view displays the top-level design and any sub-projects that can be synthesized.

Design Hierarchy Tab

The Design Hierarchy view displays the instance block and design block hierarchy for a design.

Copyright © 2013 Synopsys, Inc.

The colors for the block icons represent the following:

Icon Description	Designates the following...
White rectangle surrounding a b	Black box
Yellow	Design block (subproject)
Yellow with a T inside	Top-level design
Green with a P inside	Design block that has been exported as a sub-project

Copyright © 2013 Synopsys, Inc.

The Project Results View

The Project Results view appears on the right side of the Project view and contains the results of the synthesis runs for the implementations of your design. The Project Results view includes the following:

- Project Status Tab
- Implementation Directory
- Process View

Project Status Tab

The Project Status view provides an overview of the project settings and at-a-glance summary of synthesis messages and reports such as an area or optimization summary for the active implementation. You can track the status and settings for your design and easily navigate to reports and messages in the Project view.

To display this window, click on the Project Status tab in the Project view. An overview for the project is displayed in a spreadsheet format for each of the following sections:

- Project Settings
- Run Status
- Reports

Copyright © 2013 Synopsys, Inc.

Project Status | Implementation Directory | Process View

Project Settings

Project Name	physical_synthsys	Implementation Name	logical_synthesis
Top Module	eight_bit_uc	Pipelining	1
Retiming	0	Resource Sharing	1
Use Xilinx Xflow	0	Fanout Guide	10000
Disable I/O Insertion	0	Disable Sequential Optimizations	0
Clock Conversion	1	Use Xilinx Partition Flow	0

Run Status

Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	out-of-date	21	0	0	-	0m:00s	-	12/13/2012 1:33:39 PM
Premap Detailed report	out-of-date	3	3	0	0m:00s	0m:01s	218MB	12/13/2012 1:25:40 PM
Map & Optimize Detailed report	out-of-date	24	17	0	0m:20s	0m:22s	756MB	12/13/2012 1:26:03 PM
Xilinx P&R (xtclsh) Detailed report	out-of-date				0m:00s			5/25/2010 12:13:48 PM

Area Summary

I/O ports	58	Non I/O Register bits	371 (0%)
I/O Register bits	0	Block Rams	0 (156)
DSP48s	0 (288)	LUTs	450 (0%)
Detailed report		Hierarchical Area report	

Timing Summary

Detailed report

Optimizations Summary

You can expand or collapse each section of the Project Status view by clicking on the + or - icon in the upper left-corner of each section.

Project Status | Implementation Directory | Process View

Project Settings

Project Name	physical_synthsys	Implementation Name	logical_synthesis

Project Settings

The Project Settings table gets populated with the project settings from the run_options.txt file after a synthesis run. This section displays information, such as:

- Project name, top-level module, and implementation name
- Project options currently specified, such as Pipeling, Retiming, Resource Sharing, Fanout Guide, and Disable I/O Insertion.

Run Status

The Run Status table gets updated during and after a synthesis run. This section displays job status information for the compiler, premap job, mapper, and place-and-route runs, as needed. This section displays information, such as:

- Job name - For example, jobs include HDL Compiler, Premap Job, and Mapper. The job might have a Detailed Report link. When you click on this link, a Report tab is created that displays information for the selected job. Thereafter, information is rewritten to the Report tab for subsequent links. Use the arrow icon ( ) to get back to the main Project Status view.

Copyright © 2013 Synopsys, Inc.

- Job status - Job can be running or completed.
- Notes, warnings, and errors – A message count is displayed with a link to information about these messages. When you click on this link, a Report tab is created that displays these messages. Thereafter, subsequent links overwrite information to this Report tab . Use the arrow icon (⇦) to get back to the main Project Status view.

```
Compile Input : Notes
@N|Running in 32-bit mode
@N: CG364 :"C:\SummaryReport\aquaris_top\src_sbg\sbg_aquaris_top\sbg_aquaris_top.v"
@N: CG364 :"C:\SummaryReport\aquaris_top\src_sbg\input_deframer\src\input_deframer.
@N: CG364 :"C:\SummaryReport\aquaris_top\src\mem.v":242:7:242:9|Synthesizing module
@N: CG364 :"C:\SummaryReport\aquaris_top\src\decode.v":79:7:79:12|Synthesizing modu
@N: CG179 :"C:\SummaryReport\aquaris_top\src\decode.v":907:43:907:52|Removing redun
@N: CG364 :"C:\SummaryReport\aquaris_top\src\mult.v":80:7:80:10|Synthesizing module
@N: CG364 :"C:\SummaryReport\aquaris_top\src\register.v":66:7:66:14|Synthesizing mo
@N: CL214 :"C:\SummaryReport\aquaris_top\src\register.v":134:25:134:38|Found multi-
@N: CL214 :"C:\SummaryReport\aquaris_top\src\register.v":133:25:133:38|Found multi-
@N: CG364 :"C:\SummaryReport\aquaris_top\src\datapath.v":74:7:74:14|Synthesizing mo
@N: CG364 :"C:\SummaryReport\aquaris_top\src\cpu.v":66:7:66:9|Synthesizing module c
@N: CG364 :"C:\SummaryReport\aquaris_top\src\memory_fpga.v":67:7:67:12|Synthesizing
@N: CL134 :"C:\SummaryReport\aquaris_top\src\memory_fpga.v":151:4:151:9|Found RAM R
@N: CL134 :"C:\SummaryReport\aquaris_top\src\memory_fpga.v":151:4:151:9|Found RAM R
@N: CL134 :"C:\SummaryReport\aquaris_top\src\memory_fpga.v":151:4:151:9|Found RAM R
```

- Real and CPU times, peak memory, and a timestamp

Reports

The mapper summary table generates various reports such as an Area Summary, Compile Point Summary, or Optimization Summary. Click on the Detailed Report link when applicable, then a Report tab is created that displays information about these summary tables. Thereafter, subsequent links overwrite information to the Report tab. These reports are written to the synlog folder for the active implementation.

For example, the Area Summary contains a resource usage count for components such as registers, LUTs, and I/O ports in the design. Click on the Detailed report link to display the usage count information in the design for this report.

Copyright © 2013 Synopsys, Inc.

A Hierarchical Area report is generated for newer Xilinx designs (*Virtex-5 and later, and Spartan-6 families*) that you can access from the Project Status view. This report generates area usage for components such as:

- Sequential and combinational logic
- RAM and DSP blocks

You can locate the Hierarchical Area report file in the following Implementation Directory: `/synlog/report`.

Project Settings

Project Name	tutorial	Implementation Name	rev_1

Run Status

Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	27	0	0	-	0m:02s	-	11/9/2011 3:18:57 PM
Premap Detailed report	Complete	5	0	0	0m:00s	0m:00s	75MB	11/9/2011 3:18:58 PM
Map & Optimize Detailed report	Complete	77	12	0	0m:10s	0m:10s	86MB	11/9/2011 3:19:09 PM

Area Summary

I/O ports	26	Non I/O Register bits	253 (4%)
I/O Register bits	0	Block Rams	1 (12)
DSP48s	1 (8)	LUTs	260 (10%)
Detailed report		Hierarchical Area report	

Timing Summ

Detailed report

Project Status | Implementation Directory | Process View | Report

Area Summary : Hierarchical Area report

Module name	LUTS	REGISTERS	SYNC RAMS	DSP48
eight_bit_uc	265	253	1	1
INS_ROM	21	1	1	0
alu	38	10	0	1
data_mux	17	16	0	0
ins_decode	33	18	0	0
prep4	37	16	0	0
prgm_cntr	84	106	0	0
reg_file	0	5	0	0
spcl_regs	3	73	0	0

Copyright © 2013 Synopsys, Inc.

Implementation Directory

An implementation is one version of a project, run with certain parameter or option settings. You can synthesize again, with a different set of options, to get a different implementation. In the Project view, an implementation is shown in the folder of its project; the active implementation is highlighted. You can display multiple implementations in the same Project view. The output files generated for the active implementation are displayed in the Implementation Directory.

Process View

As process flow jobs become more complex, the benefits of exposing the underlying job flow is extremely valuable. The Process View gives you this visibility to track the design progress for the synthesis and place-and-route job flows. This feature is recommended when you use the Physical Plus flow, but can be useful for all logic and physical synthesis flows. See the *Xilinx Physical Plus*, on page 64.

Click the Process View tab on the right side of the Project Results view. This displays the job flow hierarchy run on the active implementation and is a function of this current implementation and its project settings.

Copyright © 2013 Synopsys, Inc.

Process View Displays and Controls

The Process View shows the current state of a job and allows you to control the run. You can see various aspects of the synthesis process flow, such as logical synthesis, premap, map, and placement. If you run place and route, you can see its job processes as well.

Appropriate jobs of the process flow contains the following information:

- Job Input and Output Files
- Completion State

  Displays if the job generated an error, warning, or was canceled.
- Job State
  - Out-of-date – Job needs to be run.
  - Running – Job is active.
  - Complete – Job has completed and is up-to-date.
  - Complete * – Job is up-to-date, so the job is skipped.
- Run/File Time – Job process flow runtime in real time or file creation date timestamp.
- Job TCL Command – Job process name.

Each job has the following control commands that allows you run jobs at any stage of the design process, for example `map`. Right-click on any job icon and select one of the following commands from the popup menu:

- Cancel *jobProcess* that is running
- Disable *jobProcess* that you do not want to run
- Run this *jobProcess* only
- Run to this *jobProcess* from the beginning of run
- Run from this *jobProcess* to the end of run

Hierarchical Job Flows

A hierarchical job flow runs two or more subordinate jobs. Primitive jobs launch an executable, but have no subordinate jobs. The Logical Synthesis flow is a hierarchical job that runs the Compile and Map flows.

The state of a hierarchical job depends on the state of its subordinate jobs.

- If a subordinate job is out-of-date, then its parent job is out-of-date.
- If a subordinate job has an error, then its parent job terminates with this error.
- If a subordinate job has been canceled, then its parent job is canceled as well.
- If a subordinate job is running, then its parent job is also running.

The Process View is a hierarchical tree view. To collapse or expand the main hierarchical tree, enable or disable the Show Hierarchy option. Use the plus or minus icon to expand or collapse each process flow to show the details of the jobs. The icons below are used to show the information for the state of each process:

- Red arrow ( ) – Job is out-of-date and needs to be rerun.
- Green arrow ( ) – Job is up-to-date.
- Red Circle with! ( ) - Job encountered an error.

Copyright © 2013 Synopsys, Inc.

Other Windows and Views

Besides the Project view, the Synopsys FPGA synthesis tools provide other windows and views that help you manage input and output files, direct the synthesis process, and analyze your design and its results. The following windows and views are described here:

- Dockable GUI Entities, on page 81
- Watch Window, on page 81
- Tcl Script and Messages Windows, on page 84
- Tcl Script Window, on page 85
- Message Viewer, on page 85
- Output Windows (Tcl Script and Watch Windows), on page 89
- RTL View, on page 90
- Technology View, on page 91
- Hierarchy Browser, on page 93
- FSM Viewer Window, on page 95
- Text Editor View, on page 97
- Context Help Editor Window, on page 99
- Demos & Examples, on page 101
- Interactive Attribute Examples, on page 102
- Search SolvNet, on page 103

See the following for descriptions of other views and windows that are not covered here:

Project view	The Project View, on page 62
SCOPE	SCOPE Tabs, on page 471
Synplify Premier physical synthesis	Design Planner User Interface, on page 536 Physical Analyst User Interface, on page 566.

Copyright © 2013 Synopsys, Inc.

Dockable GUI Entities

Some of the main GUI entities can appear as either independent windows or docked elements of the main application window. These entities include the menu bar, Watch window, Tcl window, and various toolbars (see the description of each entity for details). Docked elements function effectively as *panes* of the application window: you can drag the border between two such panes to adjust their relative areas.

Watch Window

The Watch window is available with the Synplify Pro and Synplify Premier tools. This window displays selected information from the log file (see *Log File*, on page 608) as a spreadsheet of parameters that you select to monitor. The values are updated when synthesis finishes.

Watch Window Display

Display of the Watch window is controlled by the View ->Watch Window command. By default, the Watch window is below the Project view in the lower right corner of the main application window.

To access the Watch window configuration menu, right-click in any cell. Select Configure Watch to display the Log Watch Configuration dialog box.

Copyright © 2013 Synopsys, Inc.

In the Watch window, indicate which implementations to watch under Watch Selection. The selected implementation(s) will display in the Watch window.

You can move the Watch window anywhere on the screen; you can make it float in its own window (named Watch Window) or dock it at a docking area (an edge) of the application window. Double-click in the banner to toggle between docked and floating.

The Watch window has a special positioning popup menu that you access by right-clicking the window border. The following commands are in the menu:

Command	Description
Allow Docking	A toggle: when enabled, the window can be docked.
Hide	Hides the window; use View ->Watch Window to show it again.
Float in Main Window	A toggle: when enabled, the window is floated (undocked).

Right-clicking the window *title bar* when the Watch window is floating displays an alternative popup menu with commands Hide and Move; Move lets you position the window using either the arrow keys or the mouse.

Using the Watch Window

You can view and compare the results of multiple implementations in the Watch window.

To choose log parameters from a pull-down menu, click in the Log Parameter section of the window. Click the pull-down arrow that appears to display the parameter list choices:

The Watch window creates an entry for each implementation of a project:

Log Parameter	rev_2	rev_4
Worst Slack	-0.418	-1.266
eight_bit_uc|clock - Estimated Frequency	299.6 MHz	130.0 MHz
eight_bit_uc|clock - Requested Frequency	342.4 MHz	155.6 MHz

To choose the implementations to watch, use the Log Watch Configuration dialog box. To display this box, right-click in the Watch window, then choose Configure Watch in the popup menu. Enable Watch Selected Implementations, then choose the implementations you want to watch in the list Selected Implementations to watch. The other buttons let you watch only the active implementation or all implementations.

Copyright © 2013 Synopsys, Inc.

Tcl Script and Messages Windows

The Tcl window is available with the Synplify Pro and Synplify Premier tools. This window has tabs for the Tcl Script and Messages windows. By default, the Tcl windows are located below the Project Tree view in the lower left corner of the main application window.

You can float the Tcl windows by clicking on a window edge while holding the Ctrl or Shift key. You can then drag the window to float it anywhere on the screen or dock it at an edge of the application window. Double-click in the banner to toggle between docked and floating.

Right-clicking the Tcl windows *title bar* when the window is floating displays a popup menu with commands Hide and Move. Hide removes the window (use View ->Tcl Window to redisplay the window). Move lets you position the window using either the arrow keys or the mouse.

For more information about the Tcl windows, see *Tcl Script Window*, on page 85 and *Message Viewer*, on page 85.

Tcl Script Window

The Tcl Script window is an interactive command shell that implements the Tcl command-line interface. You can type or paste Tcl commands at the prompt ("`%` "). For a list of the available commands, type "`help *`" (without the quotes) at the prompt. For general information about Tcl syntax, choose Help ->TCL.

The Tcl script window also displays each command executed in the course of running the synthesis tool, regardless of whether it was initiated from a menu, button, or keyboard shortcut. Right-clicking inside the Tcl window displays a popup menu with the Copy, Paste, Hide, and Help commands.

See also

- *Tcl Commands for Synthesis*, on page 1637, for information about the Tcl synthesis commands.
- *Generating a Job Script*, on page 1140 in the *User Guide*.

Message Viewer

To display errors, warnings, and notes after running the synthesis tool, click the Messages tab in the Tcl Window. A spreadsheet-style interactive interface appears.

Copyright © 2013 Synopsys, Inc.

Interactive tasks in the Messages panel include:

- Drag the pane divider with the mouse to change the relative column size.
- Click on the ID entry to open online help for the error, warning, or note.
- Click on a Source Location entry to go to the section of code in the source HDL file that is causing the message.
- Click on a Log Location entry to go to its location in the log file.

The following table describes the contents of the Messages panel. You can sort the messages by clicking the column headers. For further sorting, use Find and Filter. For details about using this window, see *Checking Results in the Message Viewer*, on page 521 in the *User Guide*.

Item	Description
Find	Type into this field to find errors, warnings, or notes.
Filter	Opens the Warning Filter dialog box. See *Messages Filter*, on page 88.
Apply Filter	Enable/disable the last saved filter.

Item	Description
Group Common ID's	Enable/disable grouping of repeated messages. Groups are indicated by a number next to the type icon. There are two types of 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.
Type	The icons indicate the type of message: Error Warning Note Advisory A plus sign next to an icon indicates that repeated messages are grouped together. Click the plus sign to expand and view the various occurrences of the message.
ID	This is the message ID. You can select an underlined ID to launch help on the message.
Message	The error, warning, or note message text.
Source Location	The HDL source file that generated the error, warning, or note message.
Log Location	The location of the error, warning, or note message in the log file.
Time	The time the error, warning or note message was recorded in the log file for the various stages of synthesis (for example: compiler, premap, and map). If you rerun synthesis, only new messages generate a new timestamp for this session. **Note:** Once synthesis has run to completion, all the `srr` files for the different stages of synthesis are merged into one unified `srr` file. If you exit the GUI, these timestamps remain the same when you re-open the same project in the GUI again.
Report	Indicates which section of the Log File report the error appears, for example Compiler or Mapper.

Copyright © 2013 Synopsys, Inc.

Messages Filter

You filter which errors, warnings, and notes appear in the Messages panel of the Tcl Window using match criteria for each field. The selections are combined to produce the result. You can elect to hide or show the warnings that match the criteria you set. See *Checking Results in the Message Viewer*, on page 521 in the *User Guide*.

Item	Description
Hide Filter Matches	Hides matched criteria in the Messages Panel.
Show Filter Matches	Shows matched criteria in the Messages Panel.
Syntax Help	Gives quick syntax descriptions.
Apply	Applies the filter criteria to the Messages Panel report, without closing the window.
Type, ID, Message, Source Location, Log Location, Time, Report	Log file report criteria to use when filtering.

The following is a filtering example.

Show Filter Matches

Hide Filter Matches

Output Windows (Tcl Script and Watch Windows)

Output windows are available with the Synplify Pro and Synplify Premier tools. You can display or remove the Tcl Script and Log Watch output windows simultaneously from the Project view, by selecting View->Output Windows from the main menu. Refer to *Watch Window*, on page 81 and *Tcl Script and Messages Windows*, on page 84 for more information.

Copyright © 2013 Synopsys, Inc.

RTL View

The RTL view provides a high-level, technology-independent, graphic representation of your design after compilation, using technology-independent components like variable-width adders, registers, large multiplexers, and state machines. RTL views correspond to the `srs` netlist files generated during compilation. RTL views are only available after your design has been successfully compiled. For information about the other HDL Analyst view (the Technology view generated after mapping), see *Technology View*, on page 91.

To display an RTL view, first compile or synthesize your design, then select HDL Analyst->RTL and choose Hierarchical View or Flattened View, or click the RTL icon ( ).

An RTL view has two panes: a Hierarchy Browser on the left and an RTL schematic on the right. You can drag the pane divider with the mouse to change the relative pane sizes. For more information about the Hierarchy Browser, see *Hierarchy Browser*, on page 93. Your design is drawn as a set of schematics. The schematic for a design module (or the top level) consists of one or more sheets, only one of which is visible in a given view at any time. The title bar of the window indicates the current hierarchical schematic level, the current sheet, and the total number of sheets for that level.

The design in the RTL schematic can be hierarchical or flattened. Further, the view can consist of the entire design or part of it. Different commands apply, depending on the kind of RTL view.

The following table lists where to find further information about the RTL view:

For information about...	See...
Hierarchy Browser	Hierarchy Browser, on page 93
Procedures for RTL view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Schematic Views, on page 546 of the *User Guide.*
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool, on page 415
Commands for RTL view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands, on page 417 HDL Analyst Menu, on page 304
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands, on page 162
History commands: Back and Forward	View Menu: RTL and Technology Views Commands, on page 162
Search command	Find Command (HDL Analyst), on page 152

Technology View

A Technology view provides a low-level, technology-specific view of your design after mapping, using components such as look-up tables, cascade and carry chains, multiplexers, and flip-flops. Technology views are only available after your design has been synthesized (compiled and mapped). For information about the other HDL Analyst view (the RTL view generated after compilation), see *RTL View*, on page 90.

Copyright © 2013 Synopsys, Inc.

To display a Technology view, first synthesize your design, and then either select a view from the HDL Analyst->Technology menu (Hierarchical View, Flattened View, Flattened to Gates View, Hierarchical Critical Path, or Flattened Critical Path) or select the Technology view icon ( ).

A Technology view has two panes: a Hierarchy Browser on the left and an RTL schematic on the right. You can drag the pane divider with the mouse to change the relative pane sizes. For more information about the Hierarchy Browser, see *Hierarchy Browser*, on page 93. Your design is drawn as a set of schematics at different design levels. The schematic for a design module (or the top level) consists of one or more sheets, only one of which is visible in a given view at any time. The title bar of the window indicates the current schematic level, the current sheet, and the total number of sheets for that level.

The schematic design can be hierarchical or flattened. Further, the view can consist of the entire design or a part of it. Different commands apply, depending on the kind of view. In addition to all the features available in RTL views, Technology views have two additional features: critical path filtering and flattening to gates.

The following table lists where to find further information about the Technology view:

For information about...	See...
Hierarchy Browser	Hierarchy Browser, on page 93
Procedures for Technology view operations like crossprobing, searching, pushing/popping, filtering, flattening, etc.	Working in the Schematic Views, on page 546 of the *User Guide*
Explanations or descriptions of features like object display, filtering, flattening, etc.	HDL Analyst Tool, on page 415
Commands for Technology view operations like filtering, flattening, etc.	Accessing HDL Analyst Commands, on page 417 HDL Analyst Menu, on page 304
Viewing commands like zooming, panning, etc.	View Menu: RTL and Technology Views Commands, on page 162
History commands: Back and Forward	View Menu: RTL and Technology Views Commands, on page 162
Search command	Find Command (HDL Analyst), on page 152

Hierarchy Browser

The Hierarchy Browser is the left pane in the RTL and Technology views. (See *RTL View*, on page 90 and *Technology View*, on page 91.) The Hierarchy Browser categorizes the design objects in a series of trees, and lets you browse the design hierarchy or select objects. Selecting an object in the Browser selects that object in the schematic. The objects are organized as shown in the following table, with a symbol that indicates the object type. See *Hierarchy Browser Symbols*, on page 94 for common symbols.

Instances	Lists all the instances and primitives in the design. In a Technology view, it includes all technology-specific primitives.

Copyright © 2013 Synopsys, Inc.

Ports	Lists all the ports in the design.
Nets	Lists all the nets in the design.
Clock Tree	Lists all the instances and ports that drive clock pins in an RTL view. If you select everything listed under Clock Tree and then use the Filter Schematic command, you see a filtered view of all clock pin drivers in your design. Registers are not shown in the resulting schematic, unless they drive clocks. This view can help you determine what to define as clocks.

A tree node can be expanded or collapsed by clicking the associated icons: the square plus (+) or minus (−) icons, respectively. You can also expand or collapse all trees at the same time by right-clicking in the Hierarchy Browser and choosing Expand All or Collapse All.

You can use the keyboard arrow keys (left, right, up, down) to move between objects in the Hierarchy Browser, or you can use the scroll bar. Use the Shift or Ctrl keys to select multiple objects. See *Navigating With a Hierarchy Browser*, on page 438 for more information about using the Hierarchy Browser for navigation and crossprobing.

Hierarchy Browser Symbols

Common symbols used in Hierarchy Browsers are listed in the following table.

Symbol	Description	Symbol	Description
	Folder		Buffer
	Input port		AND gate
	Output port		NAND gate
	Bidirectional port		OR gate
	Net		NOR gate
	Other primitive instance		XOR gate
	Hierarchical instance		XNOR gate
	Technology-specific primitive or inferred ROM		Adder

Symbol	Description	Symbol	Description
	Register or inferred state machine		Multiplier
	Multiplexer		Equal comparator
	Tristate		Less-than comparator
	Inverter		Less-than-or-equal comparator

FSM Viewer Window

The FSM Viewer windows is available in the Synplify Pro and Synplify Premier tools. Pushing down into a state machine primitive in the RTL view displays the FSM Viewer and enables the FSM toolbar. The FSM Viewer contains graphical information about the finite state machines (FSMs) in your design. The window has a state-transition diagram and tables of transitions and state encodings.

For the FSM Viewer to display state machine names for a Verilog design, you must use the Verilog parameter keyword. If you specify state machine names using the define keyword, the FSM Viewer displays the binary values for the state machines, rather than their names.

Copyright © 2013 Synopsys, Inc.

You can toggle display of the FSM tables on and off with the Toggle FSM Table icon ( ) on the FSM toolbar. The FSM tables are in the following panels:

- The Transitions panel describes, for each transition, the From State, To State, and Condition of transition.
- The RTL Encodings panel describes the correlation, in the RTL view, between the states (State) and the outputs (Register) of the FSM cell.
- The Mapped Encodings panel describes the correlation, in the Technology view, between the states (State) and their encodings into technology-specific registers. The information in this panel is available only after the design has been synthesized.

The following table describes FSM Viewer operations.

To accomplish this . . .	Do this . . .
Open the FSM Viewer	Run the FSM Compiler or the FSM Explorer. Use the push/pop mode in the RTL view to push down into the FSM and open the FSM Viewer window.
Hide/display the table	Use the FSM icons.
Filter selected states and their transitions	Select the states. Right-click and choose the filter criteria from the popup, or use the FSM icons.
Display the encoding properties of a state	Select a state. Right-click to display its encoding properties (RTL or Mapped).
Display properties for the state machine	Right-click the window, outside the state-transition diagram. The property sheet shows the selected encoding method, the number of states, and the total number of transitions among states.
Crossprobe	Double-click a register in an RTL or Technology view to see the corresponding code. Select a state in the FSM view to highlight the corresponding code or register in other open views.

See also:

- *Pushing and Popping Hierarchical Levels*, on page 435, for information on the operation of pushing into a state machine.
- *FSM Viewer Toolbar*, on page 120, for information on the FSM icons.

- See *Using the FSM Viewer*, on page 606 of the *User Guide* for more information on using the FSM viewer.

Text Editor View

The Text Editor view displays text files. These can be constraint files, source code files, or other informational or report files. You can enter and edit text in the window. You use this window to update source code and fix syntax or synthesis errors. You can also use it to crossprobe the design. For information about using the Text Editor, see *Editing HDL Source Files with the Built-in Text Editor*, on page 111 in the *User Guide*.

Opening the Text Editor

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

- Double-click a source code file (v or vhd) in the Project view.

Copyright © 2013 Synopsys, Inc.

- Choose File ->Open. In the dialog box displayed, double-click a file to open it.

  With the Microsoft® Windows® operating system, you can instead drag and drop a source file from a Windows folder into the gray background area of the GUI (*not* into any particular view).

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

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

The Text Editor colors HDL source code keywords such as module and output blue and comments green.

Text Editor Features

The Text Editor has the features listed in the following table.

Feature	Description
Color coding	Keywords are blue, comments green, and strings red. All other text is black.
Editing text	You can use the Edit menu or keyboard shortcuts for basic editing operations like Cut, Copy, Paste, Find, Replace, and Goto.
Completing keywords	To complete a keyword, type enough characters to make the string unique and then press the Esc key.
Indenting a block of text	The Tab key indents a selected block of text to the right. Shift-Tab indents text to the left.
Inserting a bookmark	Click the line you want to bookmark. Choose Edit ->Toggle Bookmark, type Ctrl-F2, or click the Toggle Bookmark icon ( ) on the Edit toolbar. The line number is highlighted to indicate that there is a bookmark at the beginning of the line.
Deleting a bookmark	Click the line with the bookmark. Choose Edit ->Toggle Bookmark, type Ctrl-F2, or click the Toggle Bookmark icon ( ) on the Edit toolbar.
Deleting all bookmarks	Choose Edit ->Delete all Bookmarks, type Ctrl-Shift-F2, or click the Clear All Bookmarks icon ( ) on the Edit toolbar.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Editing columns	Press and hold Alt, then drag the mouse down a column of text to select it.
Commenting out code	Choose Edit ->Advanced ->Comment Code. The rest of the current line is commented out: the appropriate comment prefix is inserted at the current text cursor position.
Checking syntax	Use Run ->Syntax Check to highlight syntax errors, such as incorrect keywords and punctuation, in source code. If the active window shows an HDL file, then only that file is checked. Otherwise, the entire project is checked.
Checking synthesis	Use Run ->Synthesis Check to highlight hardware-related errors in source code, like incorrectly coded flip-flops. If the active window shows an HDL file, then only that file is checked. Otherwise, the entire project is checked.

See also:

- *Editor Options Command*, on page 340, for information on setting Text Editor preferences.
- *File Menu*, on page 141, for information on printing setup operations.
- *Edit Menu Commands for the Text Editor*, on page 148, for information on Text Editor editing commands.
- *Text Editor Popup Menu*, on page 367, for information on the Text Editor popup menu.
- *Text Editor Toolbar*, on page 119, for information on bookmark icons of the Edit toolbar.
- *Keyboard Shortcuts*, on page 123, for information on keyboard shortcuts that can be used in the Text Editor.

Context Help Editor Window

Use the Context Help button to copy SystemVerilog constructs into your source file. This feature is currently supported for some of the SystemVerilog constructs. When you load a SystemVerilog file into the UI, the Context Help button displays at the bottom of the window. Click on this button to display the Context Help Editor.

Copyright © 2013 Synopsys, Inc.

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. 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. 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.

Field/Option	Description
Top	Takes you to the top of the context help page for the selected construct.
Back	Takes you back to the last context help page previously viewed.
Forward	Once you have gone back to a context help page, use Forward to return to the original context help page from where you started.
Online Help	Brings up the interactive online help for the synthesis tool.
Copy	Allows you to copy selected code from the Template file and paste it into the editor file.
Insert Template	Automatically copies the code description in its entirety from the Template file to the editor file.

Demos & Examples

A Demos & Examples feature has been added to the Project view of the GUI and consists of What's New, What's Cool, and Examples pages that display information on new and commonly used features in the release to highlight. In addition there are example designs that can be loaded into the Project view to learn about these new features.

The feature consists of the following categories:

- What's New – New features in the current major release providing Learn More links to demos and additional documentation as appropriate.
- What's Cool – Commonly used features with an overview of how to use them.
- Examples – Small design examples of attributes and features identified as helpful, beneficial, or "good to know" information. These examples can be loaded into a project file and run in the tool to demonstrate the feature. For details on how to run selected attributes, see *Interactive Attribute Examples*, on page 102.

Note: When running code examples, make sure to change the Implementation Directory from the installation directory to your own results directory.

Copyright © 2013 Synopsys, Inc.

Interactive Attribute Examples

The Interactive Attribute Examples wizard lets you select pre-defined attributes to run in a project. To use this tool:

1. Launch the wizard from Help->Demos & Examples or click the Demos & Examples button in the Project view.

2. Click the Examples button; then the Interactive Attribute Examples link.

Double-click on attribute to bring up the Interactive Attribute wizard

3. Double-click on an attribute to start the wizard.
4. Specify the Working Directory location to write your project.
5. Click Generate to generate a project for your attribute.

   A project will be created with an implementation for each attribute value selected.
6. Click Generate Run to run synthesis for all the implementations. When synthesis completes:
   - The Technology view opens to show how the selected attribute impacts synthesis.
   - You can compare resource utilization and timing information between implementations in the Log Watch window.

Search SolvNet

The Synopsys FPGA synthesis tools provide an easy way to access SolvNet from within the Project view. Click on the Search SolvNet button in the GUI, then a Search SolvNet dialog box appears.

You can search the SolvNet database for Articles and Application Notes using the following methods:

Copyright © 2013 Synopsys, Inc.

- Specify a topic in the Search application notes and articles field, then click the Go button—takes you to Application Notes and Articles on SolvNet related to the topic.
- Click on the Browse all application notes link—takes you to a SolvNet page that links to all the Synopsys FPGA products Application Notes.
- Click on the Browse all articles link—takes you to the Browse Articles by Product SolvNet page.
- Click on the Go to tutorial link—takes you to the tutorial page for the Synopsys FPGA product you are using (same as Help->Tutorial).

Copyright © 2013 Synopsys, Inc.

FSM Compiler

The FSM Compiler performs proprietary, state-machine optimization techniques (other synthesis tools treat state machines as regular logic). You enable the FSM compiler to take advantage of these techniques; you do not need special directives or attributes to locate the state machines in your design. You can also, however, enable the FSM compiler selectively for individual state machines, using synthesis directives in the HDL description.

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

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

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

The log file contains a description of each state machine extracted, including a list of the reachable states and the state encoding method used.

Copyright © 2013 Synopsys, Inc.

When to Use FSM Compiler

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

To view a textual description of a state machine in terms of inputs, states, and transitions, select the state machine in the RTL view, right-click, then choose View FSM Info File in the popup menu. You can view the same information graphically with the FSM viewer. The graphical description of a state machine makes it easier to verify behavior. For information on the FSM Viewer, see *FSM Viewer Window*, on page 95.

See also:

- *Log File*, on page 608, for information on the log file.
- *RTL View and Technology View Popup Menu Commands*, on page 402, for information on the command View FSM Info File.

Where to Use FSM Compiler (Global and Local Use)

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

The following table summarizes the operations you can perform. For more information, see *Deciding when to Optimize State Machines*, on page 454 of the *User Guide*.

To...	Do this...
Globally enable (disable) the FSM Compiler	Enable (disable) the FSM Compiler check box in the Project view.
Enable (disable) the FSM compiler for a specific register	Disable (enable) the FSM Compiler check box and set the Verilog syn_state_machine directive to 1 (0), or the VHDL syn_state_machine directive to true (false), for that instance of the state register.

Copyright © 2013 Synopsys, Inc.

FSM Explorer

The FSM Explorer automatically explores different encoding styles for state machines and picks the style best suited to your design. The FSM explorer runs the FSM viewer to identify the finite state machines in a design, then analyzes the FSMs to select the optimum encoding style for each. The FSM Explorer is only available with the Synplify Pro and Synplify Premier tools. The FSM Explorer is available for certain Microsemi, Altera, and Xilinx designs.

To enable the FSM Explorer, do one of the following:

- Turn on the FSM Explorer check box in the Project view
- Display the Implementation Options dialog box (Project ->Implementation Options) and enable the FSM Explorer option on the Options/Constraints panel.

The FSM Explorer runs during synthesis. The cost of running analysis is significant, so when analysis finishes, the encoding information is saved to a file. The synthesis tool reuses the file in subsequent synthesis iterations, which reduces overhead and saves runtime by not reanalyzing the design when you recompile. However, if you make changes to your design or your state machine, you must rerun the FSM Explorer (Run ->FSM Explorer or the F10 key) to reanalyze the encoding.

For more information about using the FSM Explorer, see *Running the FSM Explorer*, on page 459 in the *User Guide*.

Using the Mouse

The mouse button operations in Synopsys FPGA products are standard; refer to Mouse Operation Terminology for a summary of supported functions. The Synopsys FPGA tools also provide support for:

- Using Mouse Strokes, on page 108
- Using the Mouse Buttons, on page 110
- Using the Mouse Wheel, on page 112

Copyright © 2013 Synopsys, Inc.

Mouse Operation Terminology

The following terminology is used to refer to mouse operations:

Term	Meaning
Click	Click with the *left* mouse button: press then release it without moving the mouse.
Double-click	Click the left mouse button twice rapidly, without moving the mouse.
Right-click	Click with the right mouse button.
Drag	Press the left mouse button, hold it down while moving the mouse, then release it. Dragging an object moves the object to where the mouse is released; then, releasing is sometimes called "*dropping*". Dragging initiated when the mouse is not over an object often traces a selection rectangle, whose diagonal corners are at the press and release positions.
Press	Depress a mouse button; unless otherwise indicated, the left button is implied. It is sometimes used as an abbreviation for "press and hold".
Hold	Keep a mouse button depressed. It is sometimes used as an abbreviation for "press and hold".
Release	Stop holding a mouse button depressed.

Using Mouse Strokes

Mouse strokes are used to quickly perform simple repetitive commands. Mouse strokes are drawn by pressing and holding the right mouse button as you draw the pattern. The stroke must be at least 16 pixels in width or height to be recognized. You will see a green mouse trail as you draw the stroke (the actual color depends on the window background color).

Some strokes are context sensitive. That is, the interpretation of the stroke depends upon the window in which the stroke is started. For example, in an Analyst view, the right stroke means "Next Sheet." In a dialog box, the right stroke means "OK."

For information on each of the available mouse strokes, consult the Mouse Stroke Tutor.

Copyright © 2013 Synopsys, Inc.

The strokes you draw are interpreted on a grid of one to three rows. Some strokes are similar, differing only in the number of columns or rows, so it may take a little practice to draw them correctly. For example, the strokes for Redo and Back differ in that the Redo stroke is back and forth horizontally, within a single-row grid, while the Back stroke involves vertical movement as well.

Redo Last Operation

Back to Previous View

The Mouse Stroke Tutor

Do one of the following to access the Mouse Stroke Tutor:

- Help->Stroke Tutor
- Draw a question mark stroke ("?")
- Scribble (Show tutor when scribbling must be enabled on the Stroke Help dialog box)

Copyright © 2013 Synopsys, Inc.

The tutor displays the available strokes along with a description and a diagram of the stroke. You can draw strokes while the tutor is displayed.

Mouse strokes are context sensitive. When viewing the Stroke Tutor, you can choose All Strokes or Current Context to view just the strokes that apply to the context of where you invoked the tutor. For example, if you draw the "?" stroke in an Analyst window, the Current Context option in the tutor shows only those strokes recognized in the Analyst window.

You can display the tutor while working in a window such as the Analyst RTL view. However you cannot display the tutor while a modal dialog is displayed, as input is restricted to the modal dialog.

Using the Mouse Buttons

The operations you can perform using mouse buttons include the following:

- You select an object by clicking it. You deselect a selected object by clicking it. Selecting an object by clicking it deselects all previously selected objects.
- You can select and deselect multiple objects by pressing and holding the Control key (Ctrl) while clicking each of the objects.

- You can select a range of objects in a Hierarchy Browser, as follows:
  - select the first object in the range
  - scroll the tree of objects, if necessary, to display the last object in the range
  - press and hold the Shift key while clicking the last object in the range

  Selecting a range of objects in a Hierarchy Browser crossprobes to the corresponding schematic, where the same objects are automatically selected.
- You can select all of the objects in a region by tracing a selection rectangle around them (lassoing).
- You can select text by dragging the mouse over it. You can alternatively select text containing no white space (such as spaces) by double-clicking it.
- Double-clicking sometimes selects an object and immediately initiates a default action associated with it. For example, double-clicking a source file in the Project view opens the file in a Text Editor window.
- You can access a contextual popup menu by clicking the right mouse button. The menu displayed is specific to the current context, including the object or window under the mouse.

  For example, right-clicking a project name in the Project view displays a popup menu with operations appropriate to the project file. Right-clicking a source (HDL) file in the Project view displays a popup menu with operations applicable to source files.

  Right-clicking a selectable object in an HDL Analyst schematic also *selects* it, and deselects anything that was selected. The resulting popup menu applies only to the selected object. See *RTL View*, on page 90, and *Technology View*, on page 91, for information on HDL Analyst views.

Most of the mouse button operations involve selecting and deselecting objects. To use the mouse in this way in an HDL Analyst schematic, the mouse pointer must be the cross-hairs symbol: ┼. If the cross-hairs pointer is not displayed, right-click the schematic background to display it.

Copyright © 2013 Synopsys, Inc.

Using the Mouse Wheel

If your mouse has a wheel and you are using a Microsoft Windows platform, you can use the wheel to scroll and zoom, as follows:

- Whenever only a horizontal scroll bar is visible, rotating the wheel scrolls the window horizontally.
- Whenever a vertical scroll bar is visible, rotating the wheel scrolls the window vertically.
- Whenever both horizontal and vertical scroll bars are visible, rotating the wheel while pressing and holding the Shift key scrolls the window horizontally.
- In a window that can be zoomed, such as a graphics window, rotating the wheel while pressing and holding the Ctrl key zooms the window.

User Interface Preferences

The following table lists the commands with which you can set preferences and customize the user interface. For detailed procedures, see the *User Guide*.

Preferences	Description	For option descriptions, see...
Text Editor	Fonts and colors	Editor Options Command
HDL Analyst tool (RTL/Technology views)	HDL Analyst options	HDL Analyst Menu
Project view	Organization and display of project files	Project View Options Command

Copyright © 2013 Synopsys, Inc.

Managing Views

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

1. Enable the option View ->Workbook Mode.

   Below the Project view are tabs, one for each open view. The icon accompanying the view name on a tab indicates the type of view. This example, shows tabs for four views: the Project view, an RTL view, a Technology view, and a Verilog Text Editor view.

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

   If you previously minimized the view, it will be activated but will remain minimized. To display it, double-click the minimized view.

3. To activate the next view and bring it to the front, type Ctrl-F6. Repeating this keyboard shortcut cycles through all open views. If the next view was minimized it remains minimized, but it is brought to the front so that you can restore it.

4. To close a view, type Ctrl-F4 in the view, or choose File ->Close.

5. You can rearrange open windows using the Window menu: you can cascade them (stack them, slightly offset), or tile them horizontally or vertically.

Copyright © 2013 Synopsys, Inc.

Toolbars

Toolbars provide a quick way to access common menu commands by clicking their icons. The following standard toolbars are available:

- Project Toolbar — Project control and file manipulation.
- Analyst Toolbar — Manipulation of RTL and Technology views.
- Physical Analyst Toolbar — Provides access to a visual display of the floorplan, placement, and global routing of the design (Synplify Premier tool only).
- Text Editor Toolbar — Text Editor bookmark commands.
- FSM Viewer Toolbar — Display of finite state machine (FSM) information. Toolbar is available in the Synplify Pro and Synplify Premier tools.
- Tools Toolbar — Opens supporting tools.

You can enable or disable the display of individual toolbars – see *Toolbar Command*, on page 163.

By dragging a toolbar, you can move it anywhere on the screen: you can make it float in its own window or dock it at a docking area (an edge) of the application window. To move the menu bar to a docking area without docking it there (that is, to leave it floating), press and hold the Ctrl or Shift key while dragging it.

Right-clicking the window *title bar* when a toolbar is floating displays a popup menu with commands Hide and Move. Hide removes the window. Move lets you position the window using either the arrow keys or the mouse.

Project Toolbar

The Project toolbar provides the following icons, by default:

The following table describes the default Project icons. Each is equivalent to a File or Edit menu command; for more information, see the following:

- *File Menu*, on page 141
- *Edit Menu*, on page 147

Icon	Description
Open Project	Displays the Open Project dialog box to create a new project or to open an existing project. Same as File ->Open Project.
New HDL file	Opens the Text Editor window with a new, empty source file. Same as File ->New, Verilog File or VHDL File.
New Constraint File (SCOPE)	Opens the SCOPE spreadsheet with a new, empty constraint file. Same as File ->New, Constraint File (SCOPE).
New Design Plan	*Available for the Synplify Premier tool with Design Planner option.* Displays the Design Planner view and RTL schematic view for a compiled design. From there, you constrain the critical path for physical optimization. Same as File ->New, Design Plan.
Open	Displays the Open dialog box, to open a file. Same as File ->Open.

Copyright © 2013 Synopsys, Inc.

Icon	Description
Save	Saves the current file. If the file has not yet been saved, this displays the Save As dialog box, where you specify the filename. The kind of file depends on the active view. Same as File ->Save.
Save All	Saves all files associated with the current design. Same as File ->Save All.
Cut	Cuts text or graphics from the active view, making it available to Paste. Same as Edit ->Cut.
Paste	Pastes previously cut or copied text or graphics to the active view. Same as Edit ->Paste.
Undo	Undoes the last action taken. Same as Edit ->Undo.
Redo	Performs the action undone by Undo. Same as Edit ->Redo.
Find	Finds text in the Text Editor or objects in an RTL view or Technology view. Same as Edit ->Find.

Analyst Toolbar

The Analyst toolbar becomes active after a design has been compiled. The toolbar provides the following icons, by default:

The following table describes the default Analyst icons. Each is equivalent to an HDL Analyst menu command – see *HDL Analyst Menu*, on page 304, for more information.

Icon	Description
RTL View	Opens a new, hierarchical RTL view: a register transfer-level schematic of the compiled design, together with the associated Hierarchy Browser. Same as HDL Analyst ->RTL ->Hierarchical View.
Technology View	Opens a new, hierarchical Technology view: a technology-level schematic of the mapped (synthesized) design, together with the associated Hierarchy Browser. Same as HDL Analyst ->Technology ->Hierarchical View.
Timing Report View	*Synplify Premier tool only* Opens the Timing Report View to display the synthesis timing report summary and the P&R timing and correlation report. This tool lets you compare synthesis timing results with place-and-route (P&R) Static Timing Analysis (STA) results and determine if paths for timing end points, start points, and requested periods match. Same as Analysis ->Timing Report View.
Filter Schematic	Filters your entire design to show only the selected objects. The result is a *filtered* schematic. Same as HDL Analyst ->Filter Schematic.

Copyright © 2013 Synopsys, Inc.

Icon	Description
Show Critical Path	Filters your design to show only the instances (and their paths) whose slack times are within the slack margin of the worst slack time of the design (see HDL Analyst ->Set Slack Margin). The result is flat if the entire design was already flat. Icon Show Critical Path also enables HDL Analyst ->Show Timing Information. Available only in a Technology view. Not available in a Timing view. Same as HDL Analyst ->Show Critical Path.
Timing Analyst	Generates and displays a custom timing report and view. The timing report provides more information than the default report (specific paths or more than five paths) or one that provides timing based on additional analysis constraint files. See *Analysis Menu*, on page 284. Only available for certain device technologies. (Not available in Synplify.) Same as Analysis ->Timing Analyst.
VCD Panel	When enabled, displays the VCS-Analyst Simulation Panel in the HDL Analyst Technology view.
Back	Goes backward in the history of displayed sheets of the current HDL Analyst view. Same as View ->Back.
Forward	Goes forward in the history of displayed sheets of the current HDL Analyst view. Same as View ->Forward.
Zoom 100%	Zooms in at a 1:1 ratio and centers the active view where you click. If the view is already normal size, it re-centers the view at the new click location. Same as View ->Normal View.[a]
Zoom In Zoom Out	Zooms the view in or out. Buttons stay active until deselected. Same as View ->Zoom In or View ->Zoom Out.[a]
Zoom Full	Zoom that reduces the active view to display the entire design. Same as View ->Full View.[b]

Copyright © 2013 Synopsys, Inc.

Icon	Description
Zoom Selected	When selected, zooms in on only the selected objects to the full window size.
Push/Pop Hierarchy	Toggles traversing the hierarchy using the push/pop mode. Same as View ->Push/Pop Hierarchy.
Previous Sheet	Displays the previous sheet of a multiple-sheet schematic. Same as View ->Previous Sheet.
Next Sheet	Displays the next sheet of a multiple-sheet schematic. Same as View ->Previous Sheet.
Select Tool	Switches from zoom to the selection tool.

a. Available only in the SCOPE spreadsheet, FSM Viewer, RTL views, and Technology views.
b. Available only in the FSM Viewer, RTL views, and Technology views.

Physical Analyst Toolbar

For details, see *Physical Analyst Toolbar*, on page 571.

Text Editor Toolbar

The Edit toolbar is active whenever the Text Editor is active. You use it to edit *bookmarks* in the file. (Other editing operations are located on the Project toolbar – see *Project Toolbar*, on page 114.) The Edit toolbar provides the following icons, by default:

Copyright © 2013 Synopsys, Inc.

The following table describes the default Edit icons. Each is available in the Text Editor, and each is equivalent to an Edit menu command there – see *Edit Menu Commands for the Text Editor*, on page 148, for more information.

Icon	Description
Toggle Bookmark	Alternately inserts and removes a bookmark at the line that contains the text cursor. Same as Edit ->Toggle bookmark.
Next Bookmark	Takes you to the next bookmark. Same as Edit ->Next bookmark.
Previous Bookmark	Takes you to the previous bookmark. Same as Edit ->Previous bookmark.
Clear All Bookmarks	Removes all bookmarks from the Text Editor window. Same as Edit ->Delete all bookmarks.

FSM Viewer Toolbar

When you push down into a state machine primitive in an RTL view, the FSM Viewer displays and enables the FSM toolbar. You must use either the Synplify Pro or Synplify Premier tools. The FSM Viewer graphically displays the states and transitions. It also lists them in table form. By default, the FSM toolbar provides the following icons, providing access to common FSM Viewer commands.

The following table describes the default FSM icons. Each is available in the FSM viewer, and each is equivalent to a View menu command available there – see *View Menu*, on page 160, for more information.

Icon	Description
Toggle FSM Table	Toggles the display of state-and-transition tables. Same as View->FSM Table.
Unfilter FSM	Restores a filtered FSM diagram so that all the states and transitions are showing. Same as View->Unfilter.
Filter by outputs	Hides all but the selected state(s), their output transitions, and the destination states of those transitions. Same as View->Filter->By output transitions.

Tools Toolbar

The Tools Toolbar opens supporting tools.

Icon	Description
Constraint Check	Checks the syntax and applicability of the timing constraints in the constraint file for your project and generates a report (*project_name*_cck.rpt). Same as Run->Constraint Check.
Launch Identify Instrumentor	Launches the Synopsys Identify Instrumentor product. For more information, see *Working with the Identify Tools*, on page 880 of the User Guide.
Launch Identify Debugger	Launches the Synopsys Identify Debugger product. For more information, see *Working with the Identify Tools*, on page 880 of the User Guide.

Copyright © 2013 Synopsys, Inc.

Icon	Description
Launch SYNCore	Launches the SYNCore IP wizard. This tool helps you build IP blocks such as memory models for your design. For more information, see *Launch SYNCore Command*, on page 245.
Launch SystemDesigner	Launches the System Designer tool. This tool helps you build embedded IP cores and then synthesize them in the Synopsys PFGA synthesis tools. For more information, refer to the System Designer tool.
VCS Simulator	Configures and launches the VCS simulator.

Design PlannerToolbar

For details, see *Design Planner Toolbar*, on page 542.

Keyboard Shortcuts

Keyboard shortcuts are key sequences that you type in order to run a command. Menus list keyboard shortcuts next to the corresponding commands. For the Synplify Premier physical synthesis tools keyboard shortcuts, see *Design Planner Keyboard Shortcuts*, on page 542 and *Physical Analyst Keyboard Shortcuts*, on page 572.

For example, to check syntax, you can press and hold the Shift key while you type the F7 key, instead of using the menu command Run ->Syntax Check.

The following table describes the keyboard shortcuts.

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
b	In an RTL or Technology view, shows all logic between two or more selected objects (instances, pins, ports). The result is a *filtered* schematic. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Expand Paths (see *HDL Analyst Menu: Filtering and Flattening Commands*, on page 308).
Ctrl-++ (number pad)	Available in the Synplify Pro and Synplify Premier tools. In the FSM Viewer, hides all but the selected state(s), their output transitions, and the destination states of those transitions. Same as View ->Filter ->By output transitions.
Ctrl-+- (number pad)	Available in the Synplify Pro and Synplify Premier tools. In the FSM Viewer, hides all but the selected state(s), their input transitions, and the origin states of those transitions. Same as View ->Filter ->By input transitions.
Ctrl-+* (number pad)	Available in the Synplify Pro and Synplify Premier tools. In the FSM Viewer, hides all but the selected state(s), their input and output transitions, and their predecessor and successor states. Same as View ->Filter ->By any transition.
Ctrl-1	In an RTL or Technology view, zooms the active view, when you click, to full (normal) size. Same as View ->Normal View.
Ctrl-a	Centers the window on the design. Same as View ->Pan Center.
Ctrl-b	In an RTL or Technology view, shows all logic between two or more selected objects (instances, pins, ports). The result is a *filtered* schematic. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst ->Hierarchical ->Expand Paths (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).
Ctrl-c	Copies the selected object. Same as Edit ->Copy. This shortcut is sometimes available even when Edit ->Copy is not. See, for instance, *Find Command (HDL Analyst)*, on page 152.)

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
Ctrl-d	In an RTL or Technology view, selects the driver for the selected net. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical ->Select Net Driver (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).
Ctrl-e	In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, to the nearest objects (no farther). The result is a *filtered* schematic. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical ->Expand (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).
Ctrl-Enter (Return)	Available in the Synplify Pro and Synplify Premier tools. In the FSM Viewer, hides all but the selected state(s). Same as View->Filter->Selected (see *View Menu*, on page 160).
Ctrl-f	Finds the selected object. Same as Edit->Find.
Ctrl-F2	Alternately inserts and removes a bookmark to the line that contains the text cursor. Same as Edit->Toggle bookmark (see *Edit Menu Commands for the Text Editor*, on page 148).
Ctrl-F4	Closes the current window. Same as File ->Close.
Ctrl-F6	Toggles between active windows.
Ctrl-g	In the Text Editor, jumps to the specified line. Same as Edit->Goto (see *Edit Menu Commands for the Text Editor*, on page 148). In an RTL or Technology view, selects the sheet number in a multiple-page schematic. Same as View->View Sheets (see *View Menu: RTL and Technology Views Commands*, on page 162).
Ctrl-h	In the Text Editor, replaces text. Same as Edit->Replace (see *Edit Menu Commands for the Text Editor*, on page 148).
Ctrl-i	In an RTL or Technology view, selects instances connected to the selected net. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical->Select Net Instances (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
Ctrl-j	In an RTL or Technology view, displays the unfiltered schematic sheet that contains the net driver for the selected net. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical->Goto Net Driver (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).
Ctrl-l	Available in the Synplify Pro and Synplify Premier tools. In the FSM Viewer, or an RTL or Technology view, toggles zoom locking. When locking is enabled, if you resize the window the displayed schematic is resized proportionately, so that it occupies the same portion of the window. Same as View->Zoom Lock (see *View Menu Commands: All Views*, on page 160).
Ctrl-m	In an RTL or Technology view, expands inside the subdesign, from the lower-level port that corresponds to the selected pin, to the nearest objects (no farther). Same as HDL Analyst->Hierarchical->Expand Inwards (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).
Ctrl-n	Creates a new file or project. Same as File->New.
Ctrl-o	Opens an existing file or project. Same as File->Open.
Ctrl-p	Prints the current view. Same as File->Print.
Ctrl-q	In an RTL or Technology view, toggles the display of visual properties of instances, pins, nets, and ports in a design.
Ctrl-r	In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, until registers, ports, or black boxes are reached. The result is a *filtered* schematic. Operates hierarchically, on lower levels as well as the current schematic. Same as HDL Analyst->Hierarchical->Expand to Register/Port (see *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306).
Ctrl-s	In the Project View, saves the file. Same as File ->Save.
Ctrl-t	Available in the Synplify Pro and Synplify Premier tools. Toggles display of the Tcl window. Same as View ->Tcl Window (see *View Menu*, on page 160).

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
Ctrl-u	Available in the Synplify Pro and Synplify Premier tools. In the Text Editor, changes the selected text to lower case. Same as Edit->Advanced->Lowercase (see *Edit Menu Commands for the Text Editor*, on page 148). In the FSM Viewer, restores a filtered FSM diagram so that all the states and transitions are showing. Same as View->Unfilter (see *View Menu: FSM Viewer Commands*, on page 162).
Ctrl-v	Pastes the last object copied or cut. Same as Edit ->Paste.
Ctrl-x	Cuts the selected object(s), making it available to Paste. Same as Edit ->Cut.
Ctrl-y	In an RTL or Technology view, goes forward in the history of displayed sheets for the current HDL Analyst view. Same as View->Forward (see *View Menu: RTL and Technology Views Commands*, on page 162). In other contexts, performs the action undone by Undo. Same as Edit->Redo.
Ctrl-z	In an RTL or Technology view, goes backward in the history of displayed sheets for the current HDL Analyst view. Same as View->Back (see *View Menu: RTL and Technology Views Commands*, on page 162). In other contexts, undoes the last action. Same as Edit ->Undo.
Ctrl-Shift-F2	Removes all bookmarks from the Text Editor window. Same as Edit ->Delete all bookmarks (see *Edit Menu Commands for the Text Editor*, on page 148).
Ctrl-Shift-h	In an RTL or Technology view, shows all pins on selected *transparent* hierarchical (non-primitive) instances. Pins on primitives are always shown. Available only in a filtered schematic. Same as HDL Analyst ->Show All Hier Pins (see *HDL Analyst Menu: Analysis Commands*, on page 311).
Ctrl-Shift-i	In an RTL or Technology view, selects all instances on the current schematic level (all sheets). This does *not* select instances on other levels. Same as HDL Analyst->Select All Schematic->Instances (see *HDL Analyst Menu*, on page 304).

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
Ctrl-Shift-p	In an RTL or Technology view, selects all ports on the current schematic level (all sheets). This does *not* select ports on other levels. Same as HDL Analyst->Select All Schematic->Ports (see *HDL Analyst Menu*, on page 304).
Ctrl-Shift-u	In the Text Editor, changes the selected text to lower case. Same as Edit->Advanced->Uppercase (see *Edit Menu Commands for the Text Editor*, on page 148).
d	In an RTL or Technology view, selects the driver for the selected net. Limited to the current schematic. Same as HDL Analyst ->Current Level ->Select Net Driver (see *HDL Analyst Menu*, on page 304).
Delete (DEL)	Removes the selected files from the project. Same as Project->Remove Files From Project.
e	In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, to the nearest objects (no farther). Limited to the current schematic. Same as HDL Analyst->Current Level->Expand (see *HDL Analyst Menu*, on page 304).
F1	Provides context-sensitive help. Same as Help->Help.
F2	In an RTL or Technology view, toggles traversing the hierarchy using the push/pop mode. Same as View->Push/Pop Hierarchy (see *View Menu: RTL and Technology Views Commands*, on page 162). In the Text Editor, takes you to the next bookmark. Same as Edit->Next bookmark (see *Edit Menu Commands for the Text Editor*, on page 148).
F4	In the Project view, adds a file to the project. Same as Project->Add Source File (see *Build Project Command*, on page 146). In an RTL or Technology view, zooms the view so that it shows the entire design. Same as View->Full View (see *View Menu: RTL and Technology Views Commands*, on page 162).
F5	Displays the next source file error. Same as Run->Next Error/Warning (see *Run Menu*, on page 237).

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
F7	Compiles your design, without mapping it. Same as Run->Compile Only (see *Run Menu*, on page 237).
F8	Synthesizes (compiles and maps) your design. Same as Run->Synthesize (see *Run Menu*, on page 237).
F10	Available in the Synplify Pro and Synplify Premier tools. In the Project view, runs the FSM Explorer to determine optimum encoding styles for finite state machines. Same as Run ->FSM Explorer (see *Run Menu*, on page 237). In an RTL or Technology view, lets you pan (scroll) the schematic by dragging it with the mouse. Same as View ->Pan (see *View Menu: RTL and Technology Views Commands*, on page 162).
F11	Toggles zooming in. Same as View->Zoom In (see *View Menu: RTL and Technology Views Commands*, on page 162).
F12	In an RTL or Technology view, filters your entire design to show only the selected objects. Same as HDL Analyst->Filter Schematic – see *HDL Analyst Menu: Filtering and Flattening Commands*, on page 308.
i	In an RTL or Technology view, selects instances connected to the selected net. Limited to the current schematic. Same as HDL Analyst->Current Level->Select Net Instances (see *HDL Analyst Menu*, on page 304).
j	In an RTL or Technology view, displays the unfiltered schematic sheet that contains the net driver for the selected net. Same as HDL Analyst->Current Level->Goto Net Driver (see *HDL Analyst Menu*, on page 304).
r	In an RTL or Technology view, expands along the paths from selected pins or ports, according to their directions, until registers, ports, or black boxes are reached. The result is a *filtered* schematic. Limited to the current schematic. Same as HDL Analyst ->Current Level->Expand to Register/Port (see *HDL Analyst Menu*, on page 304).
Shift-F2	In the Text Editor, takes you to the previous bookmark.
Shift-F4	Allows you to add source files to your project (Project->Add Source Files).

Copyright © 2013 Synopsys, Inc.

Keyboard Shortcut	Description
Shift-F5	Displays the previous source file error. Same as Run->Previous Error/Warning (see *Run Menu*, on page 237).
Shift-F7	Checks source file syntax. Same as Run->Syntax Check (see *Run Menu*, on page 237).
Shift-F8	Checks synthesis. Same as Run->Synthesis Check (see *Run Menu*, on page 237).
Shift-F10	Checks the timing constraints in the constraint files in your project and generates a report (*project_name*_cck.rpt). Same as Run->Constraint Check (see *Run Menu*, on page 237).
Shift-F11	Toggles zooming out. Same as View->Zoom Out (see *View Menu*, on page 160).
Shift-Left Arrow	Displays the previous sheet of a multiple-sheet schematic.
Shift-Right Arrow	Displays the next sheet of a multiple-sheet schematic.
Shift-s	Dissolves the selected instances, showing their lower-level details. Dissolving an instance one level replaces it, in the current sheet, by what you would see if you pushed into it using the push/pop mode. The rest of the sheet (not selected) remains unchanged. The number of levels dissolved is the Dissolve Levels value in the Schematic Options dialog box. The type (filtered or unfiltered) of the resulting schematic is unchanged from that of the current schematic. However, the effect of the command is different in filtered and unfiltered schematics. Same as HDL Analyst ->Dissolve Instances – see *Dissolve Instances*, on page 313.

Copyright © 2013 Synopsys, Inc.

Buttons and Options

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

Synplify Pro Tool Standard GUI

Synplify Premier Tool Standard GUI

Synplify Tool Standard GUI

The following table describes the Project View buttons and options.

Copyright © 2013 Synopsys, Inc.

Conitnu

Button/Option	Action
Open Project...	Opens a new or existing project. Same as File->Open Project (see *Open Project Command*, on page 146).
Close Project	Closes the current project. Same as File->Close Project (see *Run Menu*, on page 237).
Add File...	Adds a source file to the project. Same as Project->Add Source File (see *Build Project Command*, on page 146).
Change File...	Replaces one source file with another. Same as Project ->Change File (see *Change File Command*, on page 171).
Add Implementation	Creates a new implementation.
Implementation Options/	Displays the Implementation Options dialog box, where you can set various options for synthesis. Same as Project ->Implementation Options (see *Configure Revision Control Tools Command*, on page 184).
Add P&R Implementation	*Synplify Pro and Synplify Premier* Creates a place-and-route implementation to control and run place and route from within the synthesis tool. See *Add P&R Implementation Popup Menu Command*, on page 386 for a description of the dialog box, and *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide* for information about using this feature.
View Log	Displays the log file. Same as View ->View Log File (see *View Menu*, on page 160).
Frequency (MHz)	Sets the global frequency, which you can override locally with attributes. Same as enabling the Frequency (MHz) option on the Constraints panel of the Implementation Options dialog box.

Copyright © 2013 Synopsys, Inc.

Button/Option	Action
Auto Constrain	*Synplify Pro and Synplify Premier* (not in physical synthesis mode). When Auto Constrain is enabled and no clocks are defined, the software automatically constrains the design to achieve best possible timing by reducing periods of individual clock and the timing of any timed I/O paths in successive steps. See *Using Auto Constraints*, on page 739 in the *User Guide* for detailed information about using this option. You can also set this option on the Constraints panel of the Implementation Options dialog box.
Fast Synthesis	*Synplify Premier only* Fast Synthesis is a logic synthesis design flow that is specific to Synplify Premier. 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. See Chapter 19, *Fast Synthesis* in the *User Guide*.
Auto Compile Point	*Synplify Pro and Synplify Premier* Enables the automatic compile point flow, which can analyze a design and identify modules that can automatically be defined as compile points and mapped in parallel using multiprocessing. See *The Automatic Compile Point Flow*, on page 1109 in the *User Guide*.
Continue on Error	*Synplify Pro and Synplify Premier* When enabled during compilation, allows the compiler to continue, if possible, after encountering a non-syntax error within a design unit and to resume compilation with the next design unit without stopping. Continuing on a compilation error is only available with the Synplify Premier products. When enabled for compile-point synthesis, allows the operation to continue on error and synthesize the remaining compile points.

Copyright © 2013 Synopsys, Inc.

Button/Option	Action
Physical Plus	*Synplify Premier; Xilinx only* When enabled, the Physical Plus flow is a modular flow that can be run from RTL to fully optimized and placed netlist. This flow allows you to reiterate through various stages of the design process flow after you have run Synplify Premier logic synthesis with Enhanced Optimization and Xilinx P&R once. You have the flexibility to choose the part of the flow for which you can implement netlist or timing changes. Use the Process view GUI to guide you through the steps that can or must be executed based on updates to the input files. For more information, see *Xilinx Physical Plus*, on page 64 in the *User Guide*.
FSM Compiler	Turning on this option enables special FSM optimizations. Same as enabling the FSM Compiler option on the Options panel of the Implementation Options dialog box (see *FSM Compiler*, on page 105 and *Optimizing State Machines*, on page 454 in the *User Guide*).
FSM Explorer	*Synplify Pro and Synplify Premier* When enabled, the FSM Explorer selects an encoding style for the finite state machines in your design. Same as enabling the FSM Explorer option on the Options panel of the Implementation Options dialog box. For more information, see *FSM Explorer*, on page 107 and *Running the FSM Compiler*, on page 455 in the *User Guide*.
Resource Sharing	When enabled, makes the compiler use resource sharing techniques. This option does not affect resource sharing by the mapper. The option is the same as the Resource Sharing option on the Options panel of the Implementation Options dialog box. See *Sharing Resources*, on page 448 in the *User Guide* for usage details.
Pipelining	*Synplify Pro and Synplify Premier* When enabled, uses register balancing and pipeline registers on multipliers and ROMs. Same as enabling the Pipelining option on the Options panel of the Implementation Options dialog box. See *Pipelining*, on page 426in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Button/Option	Action
Retiming	*Synplify Pro and Synplify Premier* When enabled, improves the timing performance of sequential circuits. The retiming process moves storage devices (flip-flops) across computational elements with no memory (gates/LUTs) to improve the performance of the circuit. This option also adds a retiming report to the log file. Same as enabling the Retiming option on the Options panel of the Implementation Options dialog box. Use the syn_allow_retiming attribute to enable or disable retiming for individual flip-flops. See *syn_allow_retiming*, on page 1216 for syntax details. Note: Pipelining is automatically enabled when retiming is enabled.
Run	Runs synthesis (compilation and mapping). Same as the Run->Synthesize command (see *Run Menu*, on page 237).
Technical Resource Center	Goes to the web page for the Synopsys Technical Resource Center, which contains Synplicity Business Group (SBG) product Messages.

Copyright © 2013 Synopsys, Inc.

CHAPTER 3

User Interface Commands

The following describe the commands and ways to access them in the graphical user interface (GUI). For the Synplify Premier physical synthesis UI commands, see Chapter 7, *Physical Synthesis User Interface*.

- Command Access, on page 139
- File Menu, on page 141
- Edit Menu, on page 147
- View Menu, on page 160
- Project Menu, on page 168
- Implementation Options Command, on page 194
- Import Menu, on page 224
- Run Menu, on page 237
- Analysis Menu, on page 284
- HDL Analyst Menu, on page 304
- Tools Menu, on page 329
- Options Menu, on page 330
- Tech-Support Menu, on page 353
- Web Menu, on page 357
- Help Menu, on page 362
- Popup Menus, on page 365

Copyright © 2013 Synopsys, Inc.

- Project View Popup Menus, on page 372
- RTL and Technology Views Popup Menus, on page 402

Command Access

The product interface provides access to the commands in the following ways:

- Menus, on page 139
- Context-sensitive Popup Menus, on page 140
- Toolbars, on page 140
- Keyboard Shortcuts, on page 140
- Buttons and Options, on page 140
- Tcl Command Equivalents, on page 140

Menus

The set of commands on the pull-down menus in the menu bar varies depending on the view, design status, task to perform, and selected object(s). For example, the File menu commands in the Project view differ slightly from those in the RTL view. Menu commands that are not available for the current context are grayed out. The menu bar in the Project view is shown below:

The individual menus, their commands, and the associated dialog boxes are described in the following sections:

- File Menu, on page 141
- Edit Menu, on page 147
- View Menu, on page 160
- Project Menu, on page 168
- Import Menu, on page 224
- Run Menu, on page 237
- Analysis Menu, on page 284
- HDL Analyst Menu, on page 304
- Tools Menu, on page 329
- Options Menu, on page 330

Copyright © 2013 Synopsys, Inc.

- Tech-Support Menu, on page 353
- Web Menu, on page 357
- Help Menu, on page 362

Context-sensitive Popup Menus

Popup menus, available by right-clicking, offer access to commonly used commands that are specific to the current context. See *Popup Menus*, on page 365, *Project View Popup Menus*, on page 372, and *RTL and Technology Views Popup Menus*, on page 402 for information on individual popup menus.

Toolbars

Toolbars contain icons associated with commonly used commands. For more information about toolbars, see *Toolbars*, on page 114.

Keyboard Shortcuts

Keyboard shortcuts are available for commonly used commands. The shortcut appears next to the command in the menu. See *Keyboard Shortcuts*, on page 123 for details.

Buttons and Options

The Project view has buttons for quick access to commonly used commands and options. See *Buttons and Options*, on page 131 for details.

Tcl Command Equivalents

The Tcl Window is not available in the Synplify tool. The Tcl (Tool Command Language) commands can be entered directly in the Tcl window or included in Tcl scripts that you can run in batch mode. For information about Tcl commands, see Chapter 17, *Batch Commands and Scripts*.

Copyright © 2013 Synopsys, Inc.

File Menu

Use the File menu for opening, creating, saving, and closing projects and files. The following table describes the File menu commands.

Command	Description
New	Creates one of the following types of files: Text, Tcl Script, VHDL, Verilog, Xilinx Options, Synopsys Design Constraints, Compiler Directives, Constraint, Analysis Design Constraint, or Project. See *New Command*, on page 142. The Synplify Premier Design Planner option also includes Design Plan and Netlist Restructure files.
Open	Opens a project or file.
Close	Closes a project file.
Save	Saves a project or a file.
Save As	Saves a project or a file to a specified name.
Save All	Saves all projects or files.
Print	Prints a file. For more information about printing, see the operating system documentation.
Print Setup	Specify print options.
Create Image	This command is available in the following views: • HDL Analyst Views • FSM Viewer • Design Planner (Synplify Premier tools) • Physical Analyst (Synplify Premier tools) A camera pointer ( ) appears. Drag a selection rectangle around the region for which you want to create an image, then release the mouse button. You can also simply click in the current view, then the Create Image dialog appears. See *Create Image Command*, on page 144.
Build Project	Creates a new project based on the file open in the Text Editor (if active), or lets you choose files to add to a new project. See *Build Project Command*, on page 146.
Open Project	Opens a project. See *Open Project Command*, on page 146.

Copyright © 2013 Synopsys, Inc.

Command	Description
New Project	Creates a new project. If a project is already open, it prompts you to save it before creating a new one. If you want to open multiple projects, select Allow multiple projects to be opened in the Project View dialog box. See *Project View Options Command*, on page 335.
Close Project	Closes the current project.
Recent Projects	Lists recently accessed projects. Choose a project listed in the submenu to open it.
Recent files (listed as separate menu items)	Lists the last six files you opened as separate menu items. Choose a file to open it.
Exit	Exits the session.

New Command

Select File->New to display the New dialog box, where you can select a file type to be created (Verilog, VHDL, text, Tcl script, Xilinx options, Synopsys design constraints, compiler directives, constraint, analysis design constraints, project, design plan, or netlist restructure). For most file types, a text editor window opens to allow you to define the file contents. You must provide a file name. You can automatically add the new file to your project by enabling the Add To Project checkbox before clicking OK.

File Type	Opens Window	Directory Name	Extension
Verilog	Text Editor	Verilog	.v
VHDL	Text Editor	VHDL	.vhd
Text	Text Editor	Other	.txt
Tcl Script	Text Editor	Tcl Script	.tcl
Xilinx Options	Text Editor	P&R Options	.opt
FPGA Design Constraints	SCOPE	Constraint	.fdc
Compiler Directives[1]	Modified Text Editor	Compiler Directive	.cdc
Analysis Design Constraints	SCOPE	Analysis Design Constraint	.adc

Copyright © 2013 Synopsys, Inc.

File Type	Opens Window	Directory Name	Extension
Project	None	None	.prj
Design Plan[1]	Design Planner UI	Design Plan	.sfp
Netlist Restructure[1]	Bit Slice UI	Netlist Restructure	.nrf

1. Available in the Synplify Premier tools.

Create Image Command

Select File->Create Image to create a capture image from any of the following views:

- HDL Analyst Views
- FSM Viewer
- Design Planner (Synplify Premier tools)
- Physical Analyst (Synplify Premier tools)

Drag the camera cursor to define the area for the image. When you release the cursor, the Create Image dialog box appears. Use the dialog box to copy the image, save the image to a file, or to print the image.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
Copy to Clipboard	Copies the image to the clipboard so you can paste it into a selected application (for example, a Microsoft Word file). When you copy an image to the clipboard, a green check mark appears in the Copy To Clipboard field.
Save to File	Saves the image to the specified file. You can save the file in a number of formats (platform dependent) including bmp, jpg, png, ppm, tif, xbm, and xpm.
Add to Project	Adds the saved image file to the Images folder in the Project view. This option is enabled by default.
Save to File button	You must click this button to save an image to the specified file. When you save the image, a green check mark appears in the Save To File field.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
Print	Prints the image. When you print the image, a green check mark appears in the Print field.
Options	Allows you to select the resolution of the image saved to a file or copied to the clipboard. Use the Max Pixels slider to change the image resolution.
Caption	Allows you to enter a caption for a saved or copied image. The caption is overlayed at the top-left corner of the image.

Build Project Command

Select File->Build Project to build a new project. This command behaves differently if an HDL file is open in the Text Editor.

- When an active Text Editor window with an HDL file is open, File->Build Project creates a project with the same name as the open file.
- If no file is open, File->Build Project prompts you to add files to the project using the Select Files to Add to New Project dialog box. The name of the new project is the name of the first HDL file added. See *Add Source File Command*, on page 170.

Open Project Command

Select File->Open Project to open an existing project or to create a new project.

Field/Option	Description
Existing Project	Displays the Open Project dialog box for opening an existing project.
New Project	Creates a new project and places it in the Project view.

Edit Menu

You use the Edit menu to edit text files (such as HDL source files) in your project. This includes cutting, copying, pasting, finding, and replacing text; manipulating bookmarks; and commenting-out code lines. The Edit menu commands available at any time depend on the active window or view (Project, Text Editor, SCOPE spreadsheet, RTL, or Technology views).

For the Synplify Premier physical synthesis tools Edit menu commands, see *Edit Menu: Design Planner Commands*, on page 546 and *Edit Menu: Physical Analyst Commands*, on page 574.

The available Edit menu commands vary, depending on your current view. The following table describes all of the Edit menu commands:

Command	Description
Basic Edit Menu Commands	
Undo	Cancels the last action.
Redo	Performs the action undone by Undo.
Cut	Removes the selected text and makes it available to Paste.
Copy	Duplicates the selected text and makes it available to Paste.
Paste	Pastes text that was cut (Cut) or copied (Copy).
Delete	Deletes the selected text.

Copyright © 2013 Synopsys, Inc.

Command	Description
Find	Searches the file for text matching a given search string; see *Find Command (Text)*, on page 149. In the RTL view, opens the Object Query dialog box, which lets you search your design for instances, symbols, nets, and ports, by name; see *Find Command (HDL Analyst)*, on page 152. In the project view, searches files for text strings; see *Find Command (In Project)*, on page 150.
Find Next	Continues the search initiated by the last Find.
Find in Files	Performs a string search of the target files. See *Find in Files Command*, on page 156.
Edit Menu Commands for the Text Editor	
Select All	Selects all text in the file.
Replace	Finds and replaces text. See *Replace Command*, on page 158.
Goto	Goes to a specific line number. See *Goto Command*, on page 159.
Toggle bookmark	Toggles between inserting and removing a bookmark on the line that contains the text cursor.
Next bookmark	Takes you to the next bookmark.
Previous bookmark	Takes you to the previous bookmark.
Delete all bookmarks	Removes all bookmarks from the Text Editor window.
Advanced->Comment Code	Inserts the appropriate comment prefix at the current text cursor location.
Advance -> Uncomment Code	Removes comment prefix at the current text cursor location.
Advanced->Uppercase	Makes the selected string all upper case.
Advanced->Lowercase	Makes the selected string all lower case.
Select->All	Selects all text in the file (same as All).

Copyright © 2013 Synopsys, Inc.

Find Command (Text)

Select Edit->Find to display the Find dialog box. In the SCOPE window, the FSM Viewer (Synplify Pro and Synplify Premier tools only), and the Text Editor window, the command has basic text-based search capabilities. Some search features, like regular expressions and line-number highlighting, are available only in the Text Editor. See *Find Command (In Project)*, on page 150, to search for files in the Project.

The HDL Analyst Find command is different; see *Find Command (HDL Analyst)*, on page 152 for details.

Field/Option	Description
Find What/Search for	Search string matching the text to find. In the text editor, you can use the pull-down list to view and reuse search strings used previously in the current session.
Match whole word only (text editor only)	When enabled, matches the entire word rather than a portion of the word.
Match Case	When enabled, searching is case sensitive.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
Regular expression (text editor only)	When enabled, wildcard characters (* and ?) can be used in the search string: ? matches any single character; * matches any string of characters, including an empty string.
Direction/Reverse	Changes search direction. In the text editor, buttons select the search direction (Up or Down).
Find Next	Initiates a search for the search string (see Find What/Search for). In the text editor, searching starts again after reaching the end (Down) or beginning (Up) of the file.
Wrap (SCOPE only)	When enabled, searching starts again after reaching the end or beginning (Reverse) of the spread sheet.
Mark All (Text editor only)	Highlights the line numbers of the text matching the search string and closes the Find dialog box.

Find Command (In Project)

Select Edit->Find to display the Find File dialog box. In the Project view, the command has basic text-based search capabilities to locate files in the project.

Field/Option	Description
All or part of the file name	Search string matching the file to find. You can specify all or part of the file name.
Look in	Search for files in all projects or limit the search to files only in the specified project.
Match Case	When enabled, searching is case sensitive.
Search up	Searches in the up direction (search terminates when an end of tree is reached in either direction).
Exclude path	Excludes the path name during the search.
Find Next	Initiates a search for the file name string.

Copyright © 2013 Synopsys, Inc.

Find Command (HDL Analyst)

In the RTL or Technology view, use Edit->Find to display the Object Query dialog box. For a detailed procedure about using this command, see Using Find for Hierarchical and Restricted Searches, on page 570 of the *User Guide*.

The available Find menu commands vary, depending on your current view. The following table describes all of the Find menu commands:

Field/Option	Description
Instances, Symbols, Nets, Ports	Tabbed panels for finding different kinds of objects. Choose a panel for a given object type by clicking its tab. In terms of memory consumption, searching for Instances is most efficient, and searching for Nets is least efficient.
Search	Where to search: Entire Design, Current Level & Below, or Current Level Only. See Using Find for Hierarchical and Restricted Searches, on page 570 of the *User Guide*.

Field/Option	Description
UnHighlighted	Names of all objects of the current panel type, in the level(s) chosen to Search, that match the Highlight Search (*?) filter. This list is populated by the Find 200 and Find All buttons. To select an object as a candidate for highlighting, click its name in this list. The complete name of the selected object appears near the bottom of the dialog box. You can select part or all of this complete name, then use the Ctrl-C keyboard shortcut to copy it for pasting. You can select multiple objects by pressing the Ctrl or Shift key while clicking; press Ctrl and click a selection to deselect it. The number of objects selected, and the total number listed, are displayed above the list, after the UnHighlighted: label: # selected of # total. To confirm a selection for highlighting and move the selected objects to the Highlighted list, click the -> button.
Highlight Search (*?)	Determines which object names appear in the UnHighlighted area, based on the case-sensitive filter string that you enter. For tips about using this field, see Using Wildcards with the Find Command, on page 573 of the *User Guide.* The filter string can contain the following wildcard characters: • * (asterisk) – matches any sequence of characters • ? (question mark) – matches any single character • . (period) – does not match any characters, but indicates a change in hierarchical level. Wildcards * and ? only match characters within the current hierarchy level; a*b*, for example, will not cross levels to match alpha.beta (where the period indicates a change in hierarchy). If you must match a period character occurring in a name, use \. (backslash period) in the filter string. The backslash prevents interpreting the period as a wildcard. The filter string is matched at each searched level of the hierarchy (the Search levels are described above). Use filter strings that are as specific as possible to limit the number of unwanted matches. Unnecessarily extensive search can be costly in terms of memory performance.
->	Moves the selected names from the UnHighlighted area to the Highlighted area, and highlights their objects in the RTL and Technology views.
<-	Moves the selected names from the Highlighted area to the UnHighlighted area, and unhighlights their objects in the RTL and Technology views.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
All ->	Moves all names from the UnHighlighted to the Highlighted area, and highlights their objects in the RTL and Technology views.
<- All	Moves all names from the Highlighted to the UnHighlighted area, and unhighlights their objects in the RTL and Technology views.
Highlighted	Complementary and analogous to the UnHighlighted area. You select object names here as candidates for moving to the UnHighlighted list. (You move names to the UnHighlighted list by clicking the <- button which unselects and unhighlights the corresponding objects.) When you select a name in the Highlighted list, the view is changed to show the (original, unfiltered) schematic sheet containing the object.
Un-Highlight Selection (*?)	Complementary and analogous to the Highlight Search area: selects names in the Highlighted area, based on the filter string you input here.
Jump to location	When enabled, jumps to another sheet if necessary to show target objects.
Name Space: Tech View	Searches for the specified name using the mapped (srm) database. For more information, see Using Find for Hierarchical and Restricted Searches, on page 570 of the *User Guide*.
Name Space: Netlist	Searches for the specified name using the output netlist file. For more information, see Using Find for Hierarchical and Restricted Searches, on page 570 of the *User Guide*.

Field/Option	Description
Find 200	Adds up to 200 more objects that match the filter string to the UnHighlighted list. This button becomes available after you enter a Highlight Search (*?) filter string. This button does not find objects in HDL Analyst views. It matches names of design objects against the Highlight Search (*?) filter and provides the candidates listed in the UnHighlighted list, from which you select the objects to find. Using the Enter (Return) key when the cursor is in the Highlight Search (*?) field is equivalent to clicking the Find 200 button. *Usage note:* Click Find 200 before Find All to prevent unwanted matches in case the Highlight Search (*?) string is less selective than you expect.
Find All	Places all objects that match the Highlight Search (*?) filter string in the UnHighlighted list. This button does not find objects in HDL Analyst views. It matches names of design objects against the Highlight Search (*?) filter and provides the candidates listed in the UnHighlighted list, from which you select the objects to find. (Enter a filter string before clicking this button.) See *Usage Note* for Find 200, above.

For more information on using the Object Query dialog box, see Using Find for Hierarchical and Restricted Searches, on page 570 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Find in Files Command

The Find in Files command searches the defined target for the occurrence of a specified search string. The list of files containing the string is reported in the display area at the bottom of the dialog box. For information on using this feature, see *Searching Files*, on page 284 of the *User Guide*.

Field/Option	Description
Find what	Text string object of search.
Files Contained in Project	Drop-down menu identifying the source project of the files to be searched.
Implementation Directory	Drop-down menu restricting project search to a specific implementation or all implementations.
Directory	Identifies directory for files to be searched.

Field/Option	Description
Result Window	Allows a secondary search string (Find what) to be applied to the targets reported from the initial search.
Include sub-folders for directory searches	When checked, extends the search to sub-directories of the target directory.
File filter	Excludes files from the search by filename extension.
Search Options	Standard string search options; check to enable.
Find	Initiates search.
Result Display	List of files containing search string. Status line lists the number of matches in each file and the number of files searched.

Copyright © 2013 Synopsys, Inc.

Replace Command

Use Edit->Replace to find and optionally replace text in the Text Editor.

Feature	Description
Find what	Search string matching the text to find. You can use the pull-down list to view and reuse search strings used previously in the current session.
Replace with	The text that replaces the found text. You can use the pull-down list to view and reuse replacement text used previously in the current session.
Match whole word only	Finds only occurrences of the exact string (strings longer than the Find what string are not recognized).
Match case	When enabled, searching is case sensitive.
Regular expression	When enabled, wildcard characters (* and ?) can be used in the search string: ? matches any single character; * matches any string of characters, including the empty string.
Selection	Replace All replaces only the matched occurrence.
Whole file	Replace All replaces all matching occurrences.
Find Next	Initiates a search for the search string (see Find What).
Replace	Replaces the found text with the replacement text, and locates the next match.
Replace All	Replaces all text that matches the search string.

Copyright © 2013 Synopsys, Inc.

Goto Command

Use Edit->Goto to go to a specified line number in the Text Editor.

Copyright © 2013 Synopsys, Inc.

View Menu

Use the View menu to set the display and viewing options, choose toolbars, and display result files. The commands in the View menu vary with the active view. The following tables describe the View menu commands in various views.

- View Menu Commands: All Views, on page 160
- View Menu: Zoom Commands, on page 161
- View Menu: RTL and Technology Views Commands, on page 162
- View Menu: FSM Viewer Commands, on page 162

For the Synplify Premier physical synthesis tools View menu commands, see *View Menu: Design Planner Commands*, on page 547 and *View Menu: Physical Analyst Commands*, on page 574.

View Menu Commands: All Views

Command	Description
Font Size	Changes the font size in the Project UI of the synthesis tools. You can select one of the following options: • Increase Font Size • Decrease Font Size • Reset Font Size (default size)
Toolbars	Displays the Toolbars dialog box, where you specify the toolbars to display. See *Toolbar Command*, on page 163.
Status Bar	When enabled, displays context-sensitive information in the lower-left corner of the main window as you move the mouse pointer over design elements. This information includes element identification.
Refresh	Updates the UI display of project files and folders.
Output Windows	Displays or removes the Tcl Script/Messages and Watch windows simultaneously in the Project view. Refer to the Tcl Window and Watch Window options for more information.

Copyright © 2013 Synopsys, Inc.

Command	Description
Tcl Window	(Synplify Pro and Synplify Premier tools only.) When enabled, displays the Tcl Script and Messages windows. All commands you execute in the Project view appear in the Tcl window. You can enter or paste Tcl commands and scripts in the Tcl window. Check for notes, warning, and errors in the Messages window.
Watch Window	(Synplify Pro and Synplify Premier tools only.) When enabled, displays selected information from the log file in the Watch window.
View Log File	Displays a log file report that includes compiler, mapper, and timing information on your design. See View Log File Command, on page 165.
View Result File	Displays a detailed netlist report.

View Menu: Zoom Commands

Command	Description
Zoom In Zoom Out	Lets you Zoom in or out. When selected, a Z-shaped mouse pointer (Z) appears. Zoom in or out on the view by clicking or dragging a box around (lassoing) the region. Clicking zooms in or out on the center of the view; lassoing zooms in or out on the lassoed region. Right-click to exit zooming mode. In the SCOPE spreadsheet, selecting these commands increases or decreases the view in small increments.
Pan	Lets you pan (scroll) a schematic or FSM view using the mouse. If your mouse has a wheel feature, use the wheel to pan up and down. To pan left and right, use the Shift key with the wheel.
Full View	Zooms the active view so that it shows the entire design.
Normal View	Zooms the active view to normal size and centers it where you click. If the view is already normal size, clicking centers the view.

Copyright © 2013 Synopsys, Inc.

View Menu: RTL and Technology Views Commands

These commands are available when the RTL view or Technology view is active. These commands are available in addition to the commands described in *View Menu Commands: All Views*, on page 160 and *View Menu: Zoom Commands*, on page 161.

Command	Description
Push/Pop Hierarchy	Traverses design hierarchy using the push/pop mode – see *Exploring Design Hierarchy*, on page 560 of the *User Guide*.
Previous Sheet	Displays the previous sheet of a multiple-sheet schematic.
Next Sheet	Displays the next sheet of a multiple-sheet schematic.
View Sheets	Displays the Goto Sheet dialog box where you can select a sheet to display from a list of all sheets. See *View Sheets Command*, on page 164.
Visual Properties	Toggles the display of information for nets, instances, pins, and ports in the HDL Analyst view. To customize the information that displays, set the values with Options->HDL Analyst Options->Visual Properties. See *Visual Properties Panel*, on page 351.
Back	Goes backward in the history of displayed sheets for the current HDL Analyst view.
Forward	Goes forward in the history of displayed sheets for the current HDL Analyst view.
Filter	Filters the RTL/Technology view to display only the selected objects.

View Menu: FSM Viewer Commands

The following commands are available when the FSM viewer is active (the Synplify tool does not support the FSM viewer). These commands are in addition to the common commands described in *View Menu Commands: All Views*, on page 160 and *View Menu: Zoom Commands*, on page 161.

.

Command	Description
Filter->Selected	Hides all but the selected state(s).
Filter->By output transitions	Hides all but the selected state(s), their output transitions, and the destination states of those transitions.
Filter->By input transitions	Hides all but the selected state(s), their input transitions, and the origin states of those transitions.
Filter->By any transition	Hides all but the selected state(s), their input and output transitions, and their predecessor and successor states.
Unfilter	Restores a filtered FSM diagram so that all the states and transitions are showing.
Cross Probing	Enables cross probing between FSM nodes and RTL view schematic.
Select All States	Selects all the states.
FSM Table	Toggles display of the transition table.
FSM Graph	Toggles FSM state diagram on or off.
Annotate Transitions	Toggles display of state transitions on or off on FSM state diagram.
Selection Transcription	
Tool Tips	Toggles state diagram tool tips on or off.
FSM Properties	Displays FSM Properties dialog box.
Unselect All	Unselects all states and transitions.

Toolbar Command

Select View->Toolbars to display the Toolbars dialog box, where you can:

- Choose the toolbars to display
- Customize their appearance

Copyright © 2013 Synopsys, Inc.

Feature	Description
Toolbars	Lists the available toolbars. Select the toolbars that you want to display.
Show Tooltips	When selected, a descriptive tooltip appears whenever you position the pointer over an icon.
Large Buttons	When selected, large icons are used.

View Sheets Command

Select View->View Sheets to display the Goto Sheet dialog box and select a sheet to display. The Goto Sheet dialog box is only available in an RTL or Technology view, and only when a multiple-sheet design is present.

To see if your design has multiple sheets, check the sheet count display at the top of the schematic window.

View Log File Command

View->View Log File displays the log file report for your project. The log file is available in either text (*project_name*.srr) or HTML (*project_name*_srr.htm) format. To enable or disable the HTML file format for the log file, select the View log file in HTML option in the Options->Project View Options dialog box.

When opening the log file, a table of contents appears. Selecting an item from the table of contents takes you to the corresponding HTML page. To go back, right-click on the HTML page and select Back from the menu.

Copyright © 2013 Synopsys, Inc.

You can use the search field to find an item in the table of contents. Enter all or part of the header name in the search field, then click Find. The log file displays the resulting section.

Find searches within collapsed tables. It expands the tables to show your results.

If the file changes while the search window is open, click the Refresh button to update the table of contents.

Copyright © 2013 Synopsys, Inc.

Project Menu

You use the Project menu to set implementation options, add or remove files from a project, change project filenames, create new implementations, and archive or copy the project. The Project menu commands change, depending on the view you are in. For example, the HDL Analyst RTL and Technology views only include a subset of the Project menu commands.

The Synplify Pro and Synplify Premier tools provide a graphical user interface (GUI) with views that help you manage hierarchical designs that can be synthesized independently and imported back to the top-level project in a team design flow called Hierarchical Project Management. For details about using the team design flow, see *Hierarchical Project Management Flows*, on page 86.

The following table describes the Project menu commands.

Command	Description
Implementation Options	Displays the Implementation Options dialog box, where you set options for implementing your design. See *Implementation Options Command*, on page 194.
Add Source File	Displays the Select Files to Add to Project dialog box. See *Add Source File Command*, on page 170.
Remove File From Project	Removes selected files from your project.
Change File	Replaces the selected file in your project with another that you choose. See *Change File Command*, on page 171.
Set VHDL Library	Displays the File Options dialog box, where you choose the library (Library Name) for synthesizing VHDL files. The default library is called work. See *Set VHDL Library Command*, on page 172.
Add Implementation	Creates a new implementation for a current design. Each implementation pertains to the same design, but it can have different options settings and/or constraints for synthesis runs. See *Add Implementation Command*, on page 172).

Copyright © 2013 Synopsys, Inc.

Command	Description
New Identify Implementation	Creates a new Identify implementation for a current design. To launch the Identify toolset, see the *Launch Identify Instrumentor Command*, on page 243 and *Launch Identify Debugger Command*, on page 245.
Convert Vendor Constraints	Use this vendor constraints utility to automatically convert Xilinx UCF constraints to SDC format. See *Convert Vendor Constraints Command*, on page 173.
Archive Project	Archives a design project. Use this command to archive a full or partial project, or to add files to or remove files from an archived project. See *Archive Project Command*, on page 175 for a description of the utility wizard options.
Un-Archive Project	Loads an archived project file to the specified directory. See *Un-Archive Project Command*, on page 176 for a description of the utility wizard options.
Copy Project	Creates a copy of a full or partial design project. See *Copy Project Command*, on page 179 for a description of the utility wizard options.
Hierarchical Project Options	Configures how to run synthesis for a sub-project or top-level project. See *Hierarchical Project Options Command*, on page 182. (Synplify Pro and Synplify Premier tools only.)
Add SubProject Implementation	Adds new implementations to the blocks in the top-level project. This command is only available when the top-level implementation is selected in the Project Files view. See *Working with Multiple Implementations in Hierarchical Projects*, on page 270 in the *User Guide* . (Synplify Pro and Synplify Premier tools only.)
Configure Revision Control	The synthesis tool integrates third-party Revision Control tools that allow you to edit and manage design files for increasingly complex and large designs in your project. See *Configure Revision Control Tools Command*, on page 184. (Synplify Premier tool only.)

Copyright © 2013 Synopsys, Inc.

Add Source File Command

Select Project->Add Source File to add files, such as HDL source files, to your project. This selection displays the Select Files to Add to Project dialog box.

Feature	Description
Look in	The directory of the file to add. You can use the pull-down directory list or the Up One Level button to choose the directory.
File name	The name of a file to add to the project. If you enter a name using the keyboard, then you must include the file-type extension.
Files of type	The type (extension) of files to be added to the project. Only files in the active directory that match the file type selected from the drop-down menu are displayed in the list of files. Use All Files to list all files in the directory.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Files To Add To Project	The files to add to the project. You add files to this list with the <-Add and <-Add All buttons. You remove files from this list with the Remove -> and Remove All -> buttons. For information about adding files to custom folders, see *Creating Custom Folders*, on page 220.
Use relative paths	When you add files to the project, you can specify either to use the relative path or full path names for the files.
Add files to Folders	When you add files to the project, you can specify whether or not to automatically add the files to folders. See the Folder Options described below.
Folder Options	When you add files to folders, you can specify the folder name as either the: • Operating System (OS) folder name • Parent path name from a list provided in the display

Change File Command

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

Copyright © 2013 Synopsys, Inc.

Set VHDL Library Command

Select Project->Set VHDL Library to display the File Options dialog box, where you view VHDL file properties and specify the VHDL library name. See *File Options Popup Menu Command*, on page 381. This is the same dialog box as that displayed by right-clicking a VHDL filename in the Project view and choosing File Options.

Add Implementation Command

Select Project->Add Implementation to create a new implementation for the selected project. This selection displays the Implementation Options dialog box, where you define the implementation options for the project – see *Configure Revision Control Tools Command*, on page 184. This is the same dialog box as that displayed by Project->Implementation Options, except that there is no list of Implementations to the right of the tabbed panels.

Convert Vendor Constraints Command

Use Project->Convert Vendor Constraints to automatically translate Xilinx UCF constraints to SDC format from the information specified in the dialog box. Typically, you use this command to convert a Synplify Pro logic synthesis project to a Synplify Premier project for physical synthesis. This application creates a new project that contains all supported UCF constraints converted to SDC constraints. However, your original project and project files are not altered.

See *Converting and Using Xilinx UCF Constraints*, on page 200 of the *User Guide* for details on translating the constraints.

Copyright © 2013 Synopsys, Inc.

The UCF to SDC Conversion dialog box contains the following options:

Command	Description
Current Project • Project Name • Reference Implementation	Automatically displays the current project name and implementation. You cannot edit these fields.
Project Name	Specifies a name for the new project to be created.
Project Location	Specifies a location for the project to be created.
Full Path	Displays the full path for the new project.
Constraint files with corresponding netlist files	Lists the Xilinx constraint files added to the current project. Enable the files you want to translate. You can add or remove files by double-clicking the constraint files. Note that this command does not work on `ucf` files generated by the Synopsys FPGA tools. For details, see *Supported Input Files for UCF Conversion*, on page 206 in the *User Guide*.
Run Constraints Checker after Conversion and invoke report file	Determines whether the Constraint Checker is run after translation, and whether a report file is generated.
Warning Messages	Displays messages about the status of constraint translation or constraint checker messages.
Convert	When clicked, it translates UCF constraints to SDC.

Archive Project Command

Use the Project->Archive Project command to store files for a design project into a single archive file in Synopsys Proprietary Format (`sar`). You can archive an entire project or selected files from the project.

The Archive Project command displays the Synopsys Archive Utility wizard consisting of either two (all files archived) or three (custom file selection) tabs.

Copyright © 2013 Synopsys, Inc.

Option	Description
Project Path and Filename	Path and filename of the .prj file.
Root Directory	Top-level directory that contains the project files.
Destination Directory	Pathname of the directory to store the archive .sar file.
Archive Style	The type of archive: • Create a fully self-contained copy – all project files are archived; includes project input files and result files. • If the project contains more than one implementation: - All Implementation includes all implementations in the project. - Active Implementation includes only the active implementation. • Customized file list – only project files that you select are included in the archive. • Local copy for internal network – only project input files are archived, no result files will be included.
Create Project using	If you select the Customized file list option in the wizard, you can choose one the following options on the second tab: • Source Files – Includes all design files in the archive. You cannot enable the SRS option if this option is enabled. • SRS – Includes all .srs files (RTL schematics) in the archive. You cannot enable the Source Files option when this option is enabled.
Add Extra Files	If you select the Customized file list option in the wizard, you can use this button on the second tab to add additional files to the archive.

For step-by-step details on how to use the archive utility, see *Archive Project Command*, on page 175.

Un-Archive Project Command

Use the Project->Un-Archive Project command to extract the files from an archived design project.

This command displays a Synplicity Un-Archive Utility wizard.

Option	Description
Archive Filename	Path and filename of the .prj file.
Project Name	Top-level directory that contains the project files.
Destination Directory	Pathname of the directory to store the archive .sar file.

Copyright © 2013 Synopsys, Inc.

Option	Description
Original File Reference/ Resolved File Reference	Displays the files in the archive that will be extracted. You can exclude files from the .sar by unchecking the file in the Original File Reference list. Any unchecked files are commented out in the .prj file. If there are unresolved reference files in the .sar file, you must fix (Resolve button) or uncheck them. Or, if there are files that you want the change when project files are extracted, use the Change button and select files, as appropriate. See *Resolve File Reference*, next for more details.

For step-by-step details on how to use the un-archive utility, see *Un-Archive Project Command*, on page 176.

Resolve File Reference

When you use the Un-Archive Utility wizard to extract a project, if there are unresolved file references, use the Resolve button next to the file to point to a new file location. You can also optionally replace project files in the destination directory by clicking the Change button next to the file you want to replace. The Change and Resolve buttons bring up the following dialog box:

Option	Description
Filename	Specifies the path and name of the file you want to change or resolve.
Original Directory	Specifies the location of the project at the time it was archived.
Replace directory with	Specifies the new location of the project files you want to use to replace files.
Final Filename	Specifies the path name of the directory and the file name of the replace file.
Replace buttons	• Replace – replaces only the file specified in the Filename field when the project is extracted. • Replace Unresolved – replaces any unresolved files in the project, with files of the same name from the Replace directory. • Replace All – replaces all files in the archived project with files of the same name from the Replace directory. • To undo any replace-file references, clear the Replace directory with field, then click Replace. This causes the utility to point back to the Original Directory and filenames.

Copy Project Command

Use the Project->Copy Project command to create a copy of a design project. You can copy an entire project or selected files from the project.

The Copy Project command displays the Synopsys Copy Utility wizard consisting of either two (all files copied) or three (custom file selection) tabs.

Copyright © 2013 Synopsys, Inc.

Option	Description
Project Path and Filename	Path and filename of the .prj file.
Root Directory	Top-level directory that contains the project files.
Destination Directory	Pathname of the directory to store the archive .sar file.

Option	Description
Copy Style	The type of archive: • Create a fully self-contained copy – all project files are archived; includes project input files and result files. • If the project contains more than one implementation: - All Implementation includes all implementations in the project. - Active Implementation includes only the active implementation. • Customized file list – only project files that you select are included in the archive. • Local copy for internal network – only project input files are archived, no result files will be included.
Create Project using	If you select the Customized file list option in the wizard, you can choose one the following options on the second tab: • Source Files – Includes all design files in the archive. You cannot enable the SRS option if this option is enabled. • SRS – Includes all .srs files (RTL schematics) in the archive. You cannot enable the Source Files option if this option is enabled.
Add Extra Files	If you select the Customized file list option in the wizard, you can use this button on the second tab to add additional files to the archive.

For step-by-step details on how to use the copy utility, see *Copy Project Command*, on page 179.

Copyright © 2013 Synopsys, Inc.

Hierarchical Project Options Command

Synplify Pro, Synplify Premier
Use this command to configure synthesis runs for a sub-project or top-level project when you are working with hierarchical designs. You can determine which sub-projects to run, how to run each project (top-down or bottom-up), and synchronize options across all implementations. Most of the default implementation options displayed in this dialog box are automatically filled in for the target device you specify for synthesis. For more information about using this command, see *Configuring Synthesis Runs for Hierarchical Projects, on page 278* in the *User Guide*.

	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	☐	☐	☐	☐

The following table describes the Run Configuration options for the top-level project and its sub-projects.

Edit Run Configuration Command	Description
Implementation	Selects the implementation, such as rev_1, to synthesize for the sub-project. If you do not want to run synthesis, choose <off> from the pull-down menu.
Run Type	You can choose the following run types: • top_down – the sub-project .srs files are imported back to the top-level project for final assembly. • bottom_up – design block .srd/.edif files for the sub-project are imported back to the top-level project for final assembly. This is the default.
Options	Most default options are automatically filled in with the target device you specify for synthesis from the Device tab of the Implementation Options panel.
Synchronize All Options with Top Level	Synchronizes device options for the top-level project and sub-projects so that they all match the top-level project. When you create sub-projects, the top-level and block-level device option settings can vary. Running the design with conflicting options can cause errors during synthesis. The software highlights options where there is a mismatch between the top-level and sub-project settings. Pink highlighting indicates a mismatch that must be synchronized and yellow highlighting indicates that synchronization with the top level is optional. Resolve conflicting options before synthesizing the design. See *Configuring Synthesis Runs for Hierarchical Projects*, on page 278 in the *User Guide* for details.

Insert Sub-project Command

This command lets you nest sub-projects within a sub-project hierarchy. To do this, right-click on a sub-project and select Insert SubProject from the popup menu. You can add an existing project or a new project from the Insert Project dialog box. Then, begin adding design files to your sub-project after you have created the new project.

Copyright © 2013 Synopsys, Inc.

Configure Revision Control Tools Command

Synplify Premier tools. The Configure Revision Control Tools command lets you integrate a third-party revision control tool and use its functionality to manage design files for large and complex designs in your project. The following standard source control management (SCM) functions are provided; note that these functions are available depending on the revision control tool you use:

- Add a file to the revision control tool.
- Check in a file to commit changes to the revision control tool.
- Check out a file from the revision control tool.
- Get latest revision of a file from the revision control tool.
- Discard changes and revert back to the last checked-in file.
- Refresh file information to provide SCM file status.

You must set all required environment variables before you can use the revision control tool commands with the Revision Control commands in the Project view. For information about setting up and using a revision control tool, see *Using Revision Control Tools*, on page 1149 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

The following table describes the Configure Revision Control Tools dialog box options:

Field/Option	Description
Enable Revision Control Tool Integration	Allows you to determine whether or not to integrate a revision control tool for your system. If enabled, select the revision control tool to use.
Include timestamps in messages submitted to server	When enabled, includes timestamps in messages to the server.
Current Configuration	Specify the third-party revision control tool to use. For example, you can configure the following default tools: • Perforce • CVS
Standard Commands	The standard commands include: • Command name – Read-only field. • Confirmation – When enabled, you will be asked to confirm the revision control operation. See *Confirm Revision-Control Operation Dialog Box*, on page 192. • Tcl commands to be executed from the SCM Tcl script file – Read-only field.
Edit Tcl Button	Contains the standard default commands in a SCM Tcl script file for the current configuration of the specified revision control tool. See *SCM Tcl Script File*, on page 193.
Add or Delete Button	For custom commands, you can perform the following: • Add – Adds a new custom command to the configuration. Use Edit Tcl to add the command code to the SCM Tcl file if necessary. • Delete – Select the custom command and click the Delete button.
New Configuration Button	Allows you to create a configuration for a new revision control tool. See *Create New SCM Configuration*, on page 187.

Copyright © 2013 Synopsys, Inc.

Create New SCM Configuration

Allows you can create a source control management (SCM) configuration for a new revision control tool.

Field/Option	Description
Configuration Name	Select the new revision control tool that you want to configure.
Tcl Script Path	Locate the path to the Tcl script file for the new SCM configuration for the specified revision control tool. For more information, see *Custom SCM Functionality*, on page 188.

Copyright © 2013 Synopsys, Inc.

Custom SCM Functionality

The following SCM functions are defined as standard functionality for revision control tool integration. When you create a new SCM configuration, you can implement a subset of them; including none, if you so choose. These SCM functions include:

- scm_check_in – Commits changes to the SCM database.
- scm_check_out – Checks outs one or more files from the SCM database.
- scm_add – Marks one or more files to be added to the SCM database.

  This function usually must be followed by an scm_check_in operation, so that the files are actually added to the SCM database.
- scm_revert_changes – Discards changes for one or more files and reverts their status to the last checked-in status in the SCM database.
- scm_get_latest_revision – Updates one or more files in the local workspace to the current version in the SCM database.
- scm_file_status – Indicates which icons, if any, to use for one or more files in the Project view. Usually these icons correspond to the file's current state in the SCM database; for example, checked in or marked for add.

Tcl SCM Functions

All required arguments are formatted as Tcl argument strings, for example:

```
{this is a single argument}
```

SCM Function	Function Arguments	Return Codes
scm_check_in	Requires two arguments: • First argument is a list of files to check in (use full-path names); each file is enclosed in its own set of { } braces. • Second argument is a text comment included with the check-in operation. These comments are usually a description of what changed since the previous revision. If you select the Include timestamps in messages to the server option on the Configure Revision Control dialog box, a text timestamp is appended to the second argument before the check-in operation. This may be redundant, since the revision control server keeps its own timestamps, but might be useful in some situations.	• 0 - OK status • 1 - reserved for warning, but this status is not currently used. • 2 - an error status.
scm_check_out	Requires a single argument: • A list of files to check out (use full-path names); each file is enclosed in its own set of { } braces.	• 0 - OK status • 1 - reserved for warning, but this status is not currently used. • 2 - an error status
scm_add	Requires a single argument: • A list of files to add to the SCM database (use full-path names); each file is enclosed in its own set of { } braces.	• 0 - OK status • 1 - reserved for warning, but this status is not currently used. • 2 - an error status

Copyright © 2013 Synopsys, Inc.

SCM Function	Function Arguments	Return Codes
scm_revert_changes	Requires a single argument: • A list of files for which to revert all local changes (use full-path names); each file is enclosed in its own set of { } braces.	• 0 - OK status • 1 - reserved for warning, but this status is not currently used. • 2 - an error status
scm_get_latest_revision	Requires a single argument: • A list of files for which to retrieve the latest revision from the SCM database (use full-path names); each file is enclosed in its own set of { } braces.	• 0 - OK status • 1 - reserved for warning, but this status is not currently used. • 2 - an error status
scm_file_status	Requires a single argument: • A list of files for which to retrieve the current status from the SCM database (use full-path names), each file is enclosed in its own set of { } braces.	Returns a string value.

Return Status for scm_file_status Function

The scm_file_status function returns a string value formatted as a list of name-value pairs, each enclosed in its own set of { } braces; the first and second elements are separated by a | pipe character. The first element of each pair is the file name provided in the function argument (\backslash/forward slash directory separators may be changed in either direction without harming the functionality). The second element specifies which icon to use for a particular file in the Project view.

Available icons include:

- nothing—no icon is displayed (files are not in the SCM database)
- red_check (✓)—displays a red check mark icon (indicates files are up-do-date))
- green_check (✓)—displays a green check mark icon (indicates files are modified)
- conflict(✓)—displays a conflict icon (indicates a file merge conflict)
- warning (⚠)—displays a warning icon (indicates the file is out-of-sync and requires an update)
- plus (+)—displays a red plus sign (used for files that have been added to the repository, but are not checked in)
- lock (🔒)—displays a black lock icon (indicates files are checked in and locked)

The scm_file_status command can be complicated, so an example is instructive. If you want the current status of the files c:\a\b\c.txt and d:\e\f\g.vhd, the scm_file_status command is formatted as:

```
scm_file_status {{c:\a\b\c.txt} {d:\e\f\g.vhd}}
```

The output of the scm_file_status command then looks like:

```
{{d:/e/f/g.vhd|green_check} {c:/a/b/c.txt|lock}}
```

The function itself may change the directory separators from / to \ or in the other direction; this is acceptable and does not confuse the tool. Note that the function has changed the order of the files in the list; this is also acceptable. However, all files passed in the argument MUST be listed in the string returned.

All these descriptions are the default behavior for these functions. If a different number of arguments are required for a given function (for example, Clearcase requires a second argument for check-out as well as check-in operations) you may choose not to implement the standard functionality and create a custom function with the same name using the appropriate number of arguments. For this to work correctly, one of the following two conditions must be met:

1. You name the function something other than the standard action name (see a list of standard action names below)

Copyright © 2013 Synopsys, Inc.

2. You edit the scmconfig.xml file, altering the default commandLine parameter for the command tag in question (advanced users only – editing XML is not recommended)

The second condition is only useful if you wish to use the same action name for a standard function, but with a different parameter list or a different number of parameters. Keep in mind that the label parameter of the command tag allows you to change the user-visible name of the action anyway, so it is not likely this is required.

Default action names are:

- scm_check_in (Check In)
- scm_check_out (Check Out)
- scm_add_files (Add Files to Repository)
- scm_get_latest_revision (Get Latest Revision)
- scm_revert_changes (Revert Changes – note that the visible label is Discard Changes in CVS, but internally the action name is still Revert Changes)
- scm_file_status (Refresh File Info)

Confirm Revision-Control Operation Dialog Box

When the Confirmation option is enabled for the revision control standard command selected, you will be asked to confirm this operation.

SCM Tcl Script File

When you click on the Edit Tcl button, a Tcl script file is opened that contains the standard default commands for the specified revision control tool. You can edit or add custom commands to this Tcl script files as needed. By default, the tool provides a Tcl script file for Perforce and CVS.

```
#cvs.tcl
#Implementation of SCM functionality for the CVS tool.

#check in function
#files is the list of files to be checked in
#comment is the user-supplied comment to be attached to the checkin
proc scm_check_in { files  comment } {
    set match "";
    set listsize [llength $files];
    for {set i 0} {$i<$listsize} {incr i} {
        set filename [lindex $files $i];
        set filename [string trim $filename "{}"];
```

```
#perforce.tcl
#Implementation of SCM functionality for the Perforce tool.
#For use with the 2010.2 release of Perforce.
#check in function
#files is the list of files to be checked in
#comment is the user-supplied comment to be attached to the checkin
proc scm_check_in { files  comment } {
    set match "";
    if { [llength $files]==1 } {
        set file [open "|p4 submit -d \"$comment\" $files |& more" r+];
        while {[gets $file line]>=0} {
            regexp {Change [0-9]+ submitted} $line match;
            if {$match!=""} {
                break;
            }
        }
        if {$match==""} {

            #error condition - p4 submit operation failed
            puts "Unable to submit $files: $line";

            #return code 2 to signify error condition
            return 2;
```


Copyright © 2013 Synopsys, Inc.

Implementation Options Command

You use the Implementation Options dialog box to define the implementation options for the current project. You can access this dialog box from Project->Implementation Options, by clicking the button in the Project view, or by clicking the text in the Project view that lists the current technology options.

This figure shows the Implementation Options dialog box for the Synplify Premier tool. Some of the tabs are not available in the Synplify Pro and Synplify tools.

This section describes the following:

- Device Panel, on page 195. For device-specific details of the options, refer to the appropriate vendor chapter.
- Options Panel, on page 197
- Constraints Panel, on page 200
- Implementation Results Panel, on page 203
- Timing Report Panel, on page 205
- High Reliability Panel, on page 206

Copyright © 2013 Synopsys, Inc.

- VHDL Panel, on page 207
- Verilog Panel, on page 211
- GCC Panel, on page 221 (Synplify Pro)
- GCC & Prototyping Tools Panel, on page 222 (Synplify Premier)
- Place and Route Panel, on page 223

For a description of the Design Planning panel, see *Design Planning Panel*, on page 550.

Device Panel

You use the Device panel to set mapping options for the selected technology.

The mapping options vary, depending on the technology. See *Setting Device Options*, on page 228 in the *User Guide* for a procedure, and the relevant vendor sections in this reference manual for technology-specific descriptions of the options.

Copyright © 2013 Synopsys, Inc.

The table below lists the following category of options. Not all options are available for all tools and technologies.

Option	Description
Technology Vendor	Specify the device technology you want to synthesize. You can also select the part, package, and speed grade to use. For more information, see the appropriate vendor appendix in the *Reference* manual.
Device Mapping Options	The device mapping options vary depending on the device technology you select. For more information, see the appropriate vendor appendix in the *Reference* manual.
Option Description	Click on a device mapping option to display its description in this field. Refer to the relevant vendor sections for technology-specific descriptions of the options.
System Designer Board File	If you are using a HAPS board file in your System Designer design, specify the single-FPGA board file (vb) here. The vb file is not used for synthesis; it is only used by the System Designer tool. Note that you cannot specify multi-FPGA boards. The *installPath*\lib\board directory contains supported board files. See the System Designer documentation for details about how to use the HAPS board file in your design. The equivalent Tcl command is launch_system_designer, which is described in *launch_system_designer*, on page 1658.

Options Panel

You use the Options panel of the Implementation Options dialog box to define general options for synthesis optimization. See *Setting Optimization Options*, on page 231 of the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

The following table lists the options alphabetically. Not all options are available for all tools and technologies.

Option	Description
Auto Compile Point	*Synplify Pro and Synplify Premier* Enables the automatic compile point flow, which can analyze a design and identify modules that can automatically be defined as compile points and mapped in parallel using multiprocessing. Note: You cannot use this option with the Power Optimization option. See *The Automatic Compile Point Flow*, on page 1109 in the *User Guide*.
Continue on Error	When enabled during compilation, allows the compiler to continue, if possible, after encountering a non-syntax error within a design unit and to resume compilation with the next design unit without stopping. Continuing on a compilation error is only available with the Synplify Premier products. When enabled for compile-point synthesis, allows the operation to continue on error and synthesize the remaining compile points. Continuing on a synthesis error is available with both the Synplify Pro and Synplify Premier products.
Enable 64-bit Synthesis	*Synplify Pro and Synplify Premier* Enables/disables the 64-bit mapping switch. When enabled, this switch allows you to run client programs in 64-bit mode, if available on your system. For batch mode, use this Tcl command in your project file: set_option -enable64bit 1 This option is supported on the Windows and Linux platforms.
Fast Synthesis	*Synplify Premier*)Determines whether the tool speeds up runtime in logic synthesis mode. Enabling the option reduces the number of mapper optimizations, and could result in reduced QoR. Available only when Physical Synthesis or Physical Plus is disabled. See Chapter 19, *Fast Synthesis* of the *User Guide* for information.
FSM Compiler	Determines whether the FSM Compiler is run. See *FSM Compiler*, on page 105 and *Optimizing State Machines*, on page 454 in the *User Guide*.
FSM Explorer	*Synplify Pro and Synplify Premier* Determines whether the FSM Explorer is run. See *Running the FSM Explorer*, on page 459 in the *User Guide*. **Note:** You cannot use this option with the Power Optimization option.

Option	Description
HDL Analyst Quick Load (*Beta*)	*Synplify Premier* When enabled, allows the compiler to pre-partition the design into multiple modules that are written to separate netlist files (`srs`). This feature improves memory usage in the HDL Analyst for large designs. By default, this option is disabled. For more information, see *HDL Analyst Quick Load*, on page 234.
Physical Plus	*Synplify Premier; Xilinx only* When enabled, the Physical Plus flow is a modular flow that can be run from RTL to fully optimized and placed netlist. This flow allows you to reiterate through various stages of the design process flow after you have run Synplify Premier logic synthesis with Enhanced Optimization and Xilinx P&R once. You have the flexibility to choose the part of the flow for which you can implement netlist or timing changes. Use the Process view GUI to guide you through the steps that can or must be executed based on updates to the input files. For more information, see *Xilinx Physical Plus*, on page 64 in the *User Guide*.
Physical Synthesis	*Synplify Premier; Altera only* Determines whether you run physical synthesis or logic synthesis. • Disabled Runs logic synthesis only. You can also specify fast synthesis or enhanced optimizations in this mode. See *Synplify Premier Synthesis Design Flows*, on page 39 of the *User Guide* for a list of the flows you can use in this mode. • Enabled Runs physical synthesis, which includes global placement, placement-aware optimizations, and graph-based detailed placement. Use this option to achieve the fastest timing closure and predictability. Physical synthesis can also use a design plan (sfp) file to guide its placement. See Running Physical Synthesis, on page 506 in the *User Guide*. For batch mode, use the set_option -physical_synthesis 1 Tcl command in your project file.
Pipelining	*Synplify Pro and Synplify Premier* Runs designs at a faster frequency by moving registers after the multiplier or ROM into the multiplier or ROM. See *Pipelining*, on page 426 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Option	Description
Power Optimization	*Synplify Premier; Xilinx only* Lets you reduce power for designs that contain Xilinx block RAM and DSP48 components. **Note:** You cannot use this option with the Auto Compile Point and FSM Explorer options. If either option is enabled, Power Optimization disables these options whenever they are enabled. See Power Optimization for Xilinx Designs, on page 2105.
Resource Sharing	Determines whether you optimize area by sharing resources. When enabled, this optimization technique runs during the compilation stage of synthesis. Even if it is disabled, the mapper can still flatten the netlist and re-optimize adders, multipliers as needed to improve timing, because this setting does not affect the mapper. See *Sharing Resources*, on page 448 for information for how to use this option in the *User Guide*. Enabling this option generates the resource sharing report in the log file (see *Resource Usage Report*, on page 612).
Retiming	*Synplify Pro and Synplify Premier* Determines whether the tool moves storage devices across computational elements to improve timing performance in sequential circuits. Note that the tool might retime registers associated with RAMs, DSPs, and generated clocks, regardless of the Retiming setting. See Retiming, on page 430 in the *User Guide*.

Constraints Panel

You use the Constraints panel of the Implementation Options dialog box to specify target frequency and timing constraint files for design synthesis. Depending on the synthesis tool you are using and the device you specify, the types of constraint files you can apply for the implementation may vary. See the table below for a complete list of option types you can apply.

See *Specifying Global Frequency and Constraint Files*, on page 234, in the *User Guide* for details.

Option	Description
Frequency	Sets the default global frequency. You can either set the global frequency here or in the Project view. To override the default you set here, set individual clock constraints from the SCOPE interface.
Auto Constrain	Use the Synplify Pro tool and Synplify Premier tools with physical synthesis disabled. When enabled and no clocks are defined, the software automatically constrains the design to achieve the best possible timing. It does this by reducing periods of each individual clock and the timing of any timed I/O paths in successive steps. See *Using Auto Constraints*, on page 739 in the *User Guide* for information about using this option. You can also set this option in the Project view.

Copyright © 2013 Synopsys, Inc.

Option	Description
Use clock period for unconstrained IO	Determines whether default constraints are used for I/O ports that do not have user-defined constraints. When disabled, only define_input_delay or define_output_delay constraints are considered during synthesis or forward-annotated after synthesis. When enabled, the software considers any explicit define_input_delay or define_output_delay constraints. In addition, for all ports without explicit constraints, it uses constraints based on the clock period of the attached registers. Both the explicit and implicit constraints are used for synthesis and forward-annotation. The default is off (disabled) for new designs.
Use Logical Constraints for Physical Plus	*Synplify Premier; Xilinx only* When enabled, the logical constraints provided during logical synthesis are automatically forward-annotated to physical synthesis. The file that gets forward-annotated is named *design*_synplify.sdc or *design*_synplify.fdc. This option is enabled by default.
Generate IO & Clock Generating Component Placement for Physical Plus	*Synplify Premier; Xilinx only* When enabled, lets the synthesis software automatically generate the I/O and clock component placement required for the Physical Plus feature.
Constraint Files SDC	Specifies which timing constraints (SDC) files to use for the implementation. Select the check box to select a file. For Synplify Pro and Synplify Premier tools, block-level files in the compile-point flows, the Module column shows the name of the module or compile point.
Constraint Files FDC	Specifies which timing constraints (FDC) files to use for the implementation. Select the check box to select a file. For Synplify Pro and Synplify Premier tools, block-level files in the compile-point flows, the Module column shows the name of the module or compile point.
UCF	Specifies which Xilinx user constraints (UCF) files to use for the implementation. Select the check box to select a file.
PDC	*Synplify Premier; Xilinx only* Specifies the Xilinx physical design constraints (PCF) files to use for the implementation. Select the check box to select a file.

Copyright © 2013 Synopsys, Inc.

Option	Description
NCD	*Xilinx only* Specifies the Xilinx place-and-route file (NCD) to use for the implementation. Select the check box to select a file.
Power	*Synplify Premier; Xilinx* Specifies which Activity Analysis for Power Optimization report (SAIF) files to use for the implementation. Enable the checkbox to select a file.
CDC	*Synplify Premier* Specifies which compiler directives (CDC) files to use for the implementation. Enable the checkbox to select a file.

Implementation Results Panel

You use the Implementation Results panel to specify the implementation name (default: rev_1), the results directory, and the name and format of the top-level output netlist file (Result File). You can also specify output constraint and netlist files. See *Specifying Result Options*, on page 237 of the *User Guide* for details.

The results directory is a subdirectory of the project file directory. Clicking the Browse button brings up the Select Run Directory dialog box to allow you to browse for the results directory. You can change the location of the results directory, but its name must be identical to the implementation name.

Select optional output file check boxes to generate the corresponding Verilog netlist, VHDL netlist, or vendor constraint files.

Copyright © 2013 Synopsys, Inc.

Implementation Results
Implementation name
Implementation Name:
rev_1
Results directory
Results Directory:
C:\tutorial\rev_1
Browse...
Result filename
Result Base Name:
eight_bit_uc
Result Format:
vqm
Quartus Version:
Quartus II 12.1
Result format
Optional Output File Options
Write Mapped Verilog Netlist
Write Mapped VHDL Netlist
Write Vendor Constraint File
C:\tutorial\rev_1
Browse...
Result Base Name:
eight_bit_uc
Result Format:
edif
Optional output files
Optional Output File Options
Write Mapped Verilog Netlist
Write Mapped VHDL Netlist
Write Vendor Constraint File

Option	Description
Implementation Name Results Directory Result Base Name	Displays implementation name, directory path for results, and the base name for the result files.
Result Format	Select the output that corresponds to the technology you are using. See *Generating Vendor-Specific Output*, on page 834 in the *User Guide* for a list of netlist formats.
Quartus Version	For certain Altera technologies, select the version of Quartus II place-and-route tool for the Altera library files you are using to compile your design.
Use Vivado for Place and Route	*Xilinx 7 Series (Virtex-7, Kintex-7, and Artix-7) Devices* It is recommended that you use the Vivado Integrated Design environment (IDE) tool to run place and route for these devices. You must enable this option. For more information, see *Running Xilinx Vivado Place-and-Route*, on page 852.
Write Mapped Verilog Netlist Write Mapped VHDL Netlist	Generates mapped Verilog or VHDL netlist files.
Write Vendor Constraint File	Generates a vendor-specific constraint file for forward annotation.

Timing Report Panel

Use the Timing Report panel (Implementation Options dialog box) to set criteria for the (default) output timing report. Specify the number of critical paths and the number of start and end points to appear in the timing report. See *Specifying Timing Report Output*, on page 238 in the *User Guide* for details. For a description of the report, see *Timing Reports*, on page 615.

Copyright © 2013 Synopsys, Inc.

Option	Description
Number of Critical Paths	Set the number of critical paths for the software to report.
Number of Start/End Points	Specify the number of start and end points to see reported in the critical path sections.

See also:

- Timing Reports, on page 615, for more information on the default timing report, which is affected by the Timing Report panel settings.
- Analysis Menu, on page 284, information on creating additional custom timing reports for certain device technologies.

High Reliability Panel

Use the High Reliability panel (Implementation Options dialog box) to implement safe logic for the design. For more information about high reliability support, see *Implementing High-Reliability Designs*, on page 345.

Option	Description
Preserve and Decode Unreachable States (FSM, Counters, Sequential Logic)	*Synplify Pro and Synplify Premier tools* When enabled, this option turns off sequential optimizations on all counters, FSMs, and sequential logic, to increase the reliability of the circuit. If you do not want to implement this globally, use the syn_safe_case directive (*syn_safe_case*, on page 1517) on individual FSMs.
FSM Error Correction Using Hamming Distance 3	*Synplify Premier tools* When enabled, a Hamming 3 encoding is used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers.

VHDL Panel

You use the VHDL panel in the Implementation Options dialog box to specify various language-related options. With mixed HDL designs, the VHDL and Verilog panels are both available. See *Setting Verilog and VHDL Options*, on page 239, of the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

The following table describes the options available.

Feature	Description
Top Level Entity	The name of the top-level entity of your design. If the top-level entity does not use the default work library to compile the VHDL files, you must specify the library file where the top-level entity can be found. To do this, the top-level entity name must be preceded by the VHDL library followed by a period (.). To specify VHDL library files, see *Project Menu*, on page 168 for the Set VHDL Library command, or the *File Options Popup Menu Command*, on page 381.
Default Enum Encoding	The default enumeration encoding to use. This is only for enumerated types; the FSM compiler automatically determines the state-machine encoding, or you can specify the encoding using the syn_encoding attribute.
Push Tristates	When enabled (default), tristates are pushed across process/block boundaries. For more information, see *Push Tristates Option*, on page 220.

Feature	Description
Synthesis On/Off Implemented as Translate On/Off	When enabled, the software ignores any VHDL code between synthesis_on and synthesis_off directives. It treats these third-party directives like translate_on/translate_off directives (see translate_off/translate_on, on page 1593 for details).
VHDL 2008	When enabled, allows you to use VHDL 2008 language standards.
Implicit Initial Value Support	When enabled, the compiler passes init values through a syn_init property to the mapper. For more information, see *VHDL Implicit Data-type Defaults*, on page 874.
Beta Features for VHDL	Enables use of any VHDL beta features included in the release. Enabling this checkbox is equivalent to including a set_option -hdl_define -set _BETA_FEATURES_ON_ directive in the project file.
Generics	Shows generics extracted with Extract Generic Constants. You can override the default and set a new value for the generic constant. The value is valid for the current implementation.
Extract Generic Constants	Extracts generics from the top-level entity and displays them in the table.
Design Compiler Installation Location [$SYNOPSYS=:]	*Synplify Premier tool only.* Specifies the path to the Synopsys DesignWare foundation library building blocks. Do not use this field if the $SYNOPSYS environment variable is defined.
Use DesignWare Foundation Library	*Synplify Premier tool only.* Indicates that Synopsys DesignWare foundation library building blocks are to be used in the design.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Use DesignWare MinPower Library	*Synplify Premier tool only.* Indicates that a corresponding building block from the Synopsys minPower library is to be used in place of its equivalent DesignWare foundation library building block. Selecting this option automatically selects the Use DesignWare Foundation Library checkbox.
Stop synthesis if no DesignWare license is found	*Synplify Premier tool only.* Specifies how to treat a DesignWare building block encountered in a design when no DesignWare license is available (DesignWare support is licensed separately). If checked, synthesis is stopped with an error; if unchecked (the default), the DesignWare building block is treated as a black box and synthesis is allowed to continue.

Copyright © 2013 Synopsys, Inc.

Verilog Panel

You use the Verilog panel in the Implementation Options dialog box to specify various language-related options. With mixed HDL designs, the VHDL and Verilog panels are both available. See *Setting Verilog and VHDL Options*, on page 239 of the *User Guide* for details.

Feature	Description
Top Level Module	The name of the top-level module of your design.
Compiler Directives and Parameters	Shows design parameters extracted with Extract Parameters. You can override the default and set a new value for the parameter. The value is valid for the current implementation.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Extract Parameters	Extracts design parameters from the top-level module and displays them in the table. See *Compiler Directives and Design Parameters*, on page 215.
Compiler Directives	Provides an interface where you can enter compiler directives that you would normally enter in the code with ‘ifdef and ‘define statements. See *Compiler Directives and Design Parameters*, on page 215.
Verilog Language – Verilog 2001	When enabled, the default Verilog standard for the project is Verilog 2001. When both Verilog 2001 and SysyemVerilog are disabled, the default standard is Verilog 95. For information about Verilog 2001, see *Verilog 2001 Support*, on page 642. You can override the default project standard on a per file basis by selecting the file, right-clicking, and selecting the File Options command (see *File Options Popup Menu Command*, on page 381).
Verilog Language – SystemVerilog	When enabled, the default Verilog standard for the project is SystemVerilog which is the default standard for all new projects. Enabling SystemVerilog automatically enables Verilog 2001.
Push Tristates	When enabled (default), tristates are pushed across process/block boundaries. For details, see *Push Tristates Option*, on page 220.
Allow Duplicate Modules	Allows the use of duplicate modules in your design. When enabled, the last definition of the module is used by the software and any previous definitions are ignored. You should not use duplicate module names in your Verilog design, therefore this option is disabled by default. However, if you need to, you can allow for duplicate modules by enabling this option.
Multiple File Compilation Unit	When enabled (the default), the Verilog compiler uses the compilation unit for modules defined in multiple files. See SystemVerilog *Compilation Units*, on page 809 for additional information.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Auto Infer Blackbox	Selects how undefined modules are treated in Synplify Premier. Selecting None (the default) causes the compiler to error out when an undefined module is encountered. Selecting With BIDIR ports creates a black box for the undefined module and generates a warning message, but allows the operation to continue; this mode matches the behavior of Certify.
Beta Features for Verilog	Enables use of any Verilog beta features included in the release. Enabling this checkbox is equivalent to including a set_option -hdl_define -set _BETA_FEATURES_ON_ directive in the project file.
Include Path Order	Specifies the search paths for the include commands in the Verilog design files of your project. Use the buttons in the upper right corner of the box to add, delete, or reorder the paths. The include paths are relative. See *Updating Verilog Include Paths in Older Project Files*, on page 219 in the *User Guide* for additional information.
Library Directories	Specifies all the paths to the directories which contain the Verilog library files to be included in your design for the project.
Design Compiler Installation Location [$SYNOPSYS=:]	*Synplify Premier tool only.* Specifies the path to the Synopsys DesignWare foundation library building blocks. Do not use this field if the $SYNOPSYS environment variable is defined.
Use DesignWare Foundation Library	*Synplify Premier tool only.* Indicates that Synopsys DesignWare foundation library building blocks are to be used in the design.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Use DesignWare MinPower Library	*Synplify Premier tool only.* Indicates that a corresponding building block from the Synopsys minPower library is to be used in place of its equivalent DesignWare foundation library building block. Selecting this option automatically selects the Use DesignWare Foundation Library checkbox.
Stop synthesis if no DesignWare license is found	*Synplify Premier tool only.* Specifies how to treat a DesignWare building block encountered in a design when no DesignWare license is available (DesignWare support is licensed separately). If checked, synthesis is stopped with an error; if unchecked (the default), the DesignWare building block is treated as a black box and synthesis is allowed to continue.

Copyright © 2013 Synopsys, Inc.

Compiler Directives and Design Parameters

When you click the Extract Parameters button in the Verilog panel (Implementation Options dialog box), parameter values from the top-level module are displayed in the table. You can also override the default by setting a new value for the parameter. The value is valid for the current implementation only.

The Compiler Directives field provides an interface where you can enter compiler directives that you would normally enter in the code using ‘ifdef and ‘define statements. Use spaces to separate the statements. The directives you enter are stored in the project file. For example, if you enter the directive shown below to the Compiler Directives field of the Verilog panel:

the software writes the following statement to the project file:

```
set_option -hdl_define -set "ABC=30"
```


Copyright © 2013 Synopsys, Inc.

To define multiple variables in the GUI, use a space delimiter. For example:

The software writes the following statement to the prj file:

```
set_option hdl_define -set "ABC=30 XYZ=12 vj"
```

More information is provided for the following Verilog compiler directives:

- IGNORE_VERILOG_BLACKBOX_GUTS Directive
- _BETA_FEATURES_ON_ Directive
- _SEARCHFILENAMEONLY_ Directive

IGNORE_VERILOG_BLACKBOX_GUTS Directive

When you use the syn_black_box directive, the compiler parses the contents of the black box and can determine whether illegal syntax or incorrect code is defined within it. Whenever this occurs, an error message is generated.

However, if you do not want the tool to check for illegal syntax in your black box, set the:

- Built-in compiler directive IGNORE_VERILOG_BLACKBOX_GUTS in the Compiler Directives field of the Verilog panel on the Implementation Options dialog box.

Copyright © 2013 Synopsys, Inc.

The software writes the following command to the project file:

```
set_option -hdl_define -set "IGNORE_VERILOG_BLACKBOX_GUTS"
```

- `` `define IGNORE_VERILOG_BLACKBOX_GUTS `` directive in the Verilog file.

This option is implemented globally for the project file.

Example of the IGNORE_VERILOG_BLACKBOX_GUTS Directive

Note that the IGNORE_VERILOG_BLACKBOX_GUTS directive ignores the contents of the black box. However, whenever you use this directive, you must first define the ports for the black box correctly. Otherwise, the IGNORE_VERILOG_BLACKBOX_GUTS directive generates an error. See the following valid Verilog example:

```
`define IGNORE_VERILOG_BLACKBOX_GUTS
module b1_fpga1 (A,B,C,D) /* synthesis syn_black_box */;
input B;
output A;
input [2:0] D;
output [2:0] C;
temp;
assign A = B;
assign C = D;

endmodule

module b1_fpga1_top (inout A, B, inout [2:0] C, D);
b1_fpga1 b1_fpga1_inst(A,B,C,D);
endmodule
```


Copyright © 2013 Synopsys, Inc.

_BETA_FEATURES_ON_ Directive

Beta features for the Verilog, SystemVerilog, or VHDL language must be explicity enabled. In the UI, a Beta Features checkbox is included on the VHDL or Verilog tab of the Implementations Options dialog box. A _BETA_FEATURES_ON_ compiler directive is also available. This directive is specified with a set_option -hdl_define command added to the project file as shown below:

```
set_option -hdl_define -set _BETA_FEATURES_ON_
```

The directive can also be added to the Compiler Directives field of the Verilog panel.

SEARCHFILENAMEONLY Directive

This directive provides a workaround for some known limitations of the archive utility.

If Verilog 'include files belong in any of the following categories, you may encounter problems when compiling a design after un-archiving:

1. The include paths have relative paths to the project file.

   ```
   `include "../../../defines.h"
   ```

2. The include paths have absolute paths to the project file.

   ```
   `include "c:/temp/params.h"
   ```

   ```
   `include "/remote/sbg_home/user/params.h"
   ```

3. The include paths have the same file names, but are located in different directories relative to the project file.

```
`include "../myflop.v"
…
`include "../../myflop.v"
```

Use the _SEARCHFILENAMEONLY_ directive to resolve categories 1 and 2 above. Catagory 3 is a known limitation; in this case it is recommended that you adopt standard coding practices to avoid files with the same name and different content.

When you un-archive a `sar` file that contains relative or absolute include paths for the files in the project, you can add the _SEARCHFILENAMEONLY_ compiler directive to the unarchived project; this has the compiler remove the relative/absolute paths from the `include and search only for the file names.

This directive is specified with a set_option -hdl_define command added to an implementation within the project file as shown below:

```
set_option -hdl_define -set "_SEARCHFILENAMEONLY_"
```

The directive can also be added to the Compiler Directives field of the Verilog panel as shown below.

The compiler generates the following warning message whenever it extracts include files using this directive:

```
@W: | Macro _SEARCHFILENAMEONLY_ is set: fileName not found
attempting to search for base file name fileName
```


Copyright © 2013 Synopsys, Inc.

Push Tristates Option

Pushing tristates is a synthesis optimization option you set with Project->Implementation Options->Verilog or VHDL.

Description

When the Push Tristates option is enabled, the Synopsys FPGA compiler pushes tristates through objects such as muxes, registers, latches, buffers, nets, and tristate buffers, and propagates the high-impedance state. The high-impedance states are not pushed through combinational gates such as ANDs or ORs.

Push Tristates off: tristate is not pushed through the flip-flop

Push Tristates on: tristate is pushed through the flip-flop so that the result matches RTL simulation

If there are multiple tristates, the software muxes them into one tristate and pushes it through. The software pushes tristates through loops and stores the high impedance across multiple cycles in the register.

Advantages and Disadvantages

The advantage to pushing tristates to the periphery of the design is that you get better timing results because the software uses tristate output buffers.

The Synopsys FPGA software approach to tristate inference matches the simulation approach. Simulation languages are defined to store and propagate 0, 1, and Z (high impedance) states. Like the simulation tools, the Synopsys FPGA synthesis tool propagates the high-impedance states instead of producing tristate drivers at the outputs of process (VHDL) or always (Verilog) blocks.

The disadvantage to pushing tristates is that you might use more design resources.

Effect on Other Synthesis Options

Tristate pushing has no effect on the syn_tristatetomux attribute. This is because tristate pushing is a compiler directive, while the syn_tristatetomux attribute is used during mapping.

GCC Panel

The GCC panel is only available with the Synplify Pro product and specifies how gated and generated clocks are treated during synthesis (see *GCC & Prototyping Tools Panel*, on page 222 for the Synplify Premier equivalent of this panel).

Feature	Description
Clock Conversion	Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 of the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

GCC & Prototyping Tools Panel

The GCC & Prototyping Tools panel of the Implementation Options dialog box is available with the Synplify Premier and Synplify Premier with Design Planner products and is used to specify if gated and generated clocks are to be converted during synthesis. You can also specify which netlist prototyping options are to be applied, and if a netlist restructuring or Tcl file is to be compiled with the design.

Feature	Description
Clock Conversion	Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 of the *User Guide* for details.
Force Generated Clock Conversion with Asynchronous Signals	Used with latches with asynchronous signals to force generated clock optimization when enabled. This option is automatically enabled when a HAPS technology is selected.
Convert MUX/XOR Gates Clock Tree	Performs gated-clock conversion on muxes or XOR gates in the clock tree when enabled (available only with Xilinx-based designs).
Enable prototyping tools	Enables the individual prototyping tools options to be selected and also enables the Add Prototype file button to allow a netlist restructuring file to be included in the project.
Feedthrough optimization	Prototyping tool option that eliminates unused routes through a module after it has been compiled. These optimizations are available when you floorplan the design.
Constant propagation	Prototyping tool option that eliminates the I/O pin count associated with tying a logic input signal high or low by making the connection internal to the instance rather than through the instance interface. These optimizations are available when you floorplan the design

Copyright © 2013 Synopsys, Inc.

Feature	Description
Create always/process level hierarchy	Prototyping tool option that creates an additional level of hierarchy based on the always (Verilog) or process (VHDL) blocks in the source RTL design file. Allows large modules or entities to be broken up into a set of logical submodules. These optimizations are available when you floorplan the design
Optimize Netlist	Prototyping tool option that removes unnecessary and redundant logic. These optimizations are available when you floorplan the design
Add Prototype file	Brings up an Add File dialog box to allow a prototyping tool file (bit-slice definition) or Tcl file to be specified for the project.

Place and Route Panel

The Place and Route panel allows you to run selected place-and-route jobs after design synthesis. To create a place-and-route job, see *Add P&R Implementation Popup Menu Command*, on page 386 or *Options for Place & Route Jobs Popup Menu Command*, on page 390 for details.

Copyright © 2013 Synopsys, Inc.

Import Menu

The Import menu provides support for handling various IPs in the Synopsys FPGA synthesis tools.

The following table describes the Import IP menu commands.

Command	Description
Import Xilinx EDK/ISE Project	*Synplify Pro and Synplify Premier.* Opens the ise2syn utility that lets you convert ISE or EDK projects to synthesis projects. See *Import Xilinx EDK/ISE Project Command*, on page 225.
Import Altera SOPC Project	*Synplify Pro and Synplify Premier.* Opens the sopc2syn interface that lets you include subsystems created with SOPC builder into the synthesis project. See *Import Altera SOPC Project Command*, on page 228.
Import Altera QSF Project	*Synplify Pro and Synplify Premier.* Opens the qsf2syn utility that lets you convert QSF projects to synthesis projects. See *Import Altera QSF Project Command*, on page 230.
Import Xilinx Constraints	Translates UCF and NCD constraints from ISE and DCP constraints from Vivado for placement and timing to the PDC files for Xilinx Physical Plus. See *Import Xilinx Constraints*, on page 232.
Import IP Package	Imports an IP package into your project. You need to consolidate the IP core files into a single folder or a compressed zip file before importing. See *Import IP Package Command*, on page 235.
Launch System Designer	Launches the System Designer tool, through which you can build embedded IP cores, then synthesize them in the Synopsys FPGA synthesis tools. See *Launch System Designer Command*, on page 236.

Copyright © 2013 Synopsys, Inc.

Import Xilinx EDK/ISE Project Command

Use the Import Xilinx EDK/ISE Project command to open the GUI interface to the ise2syn conversion utility. Use this utility to convert a Xilinx .ise, .xise, or .xmp project into a Synplify Pro or Synplify Premier project for synthesis. For information about using the utility to convert a Xilinx project, see *Converting Xilinx Projects with ise2syn*, on page 1051 in the *User Guide*.

For information about the equivalent Tcl command for batch mode, see *Xilinx ise2syn Utility*, on page 1017.

Copyright © 2013 Synopsys, Inc.

The following table describes the options you can set.

Field	Description
Xilinx ISE or EDK Project File	Specifies an ISE project (ise/xise) or an EDK project (xmp) to be imported.
Project Name	Specifies a name for the synthesis project file to be created. The default name for the file is *baseNameOfInputProject*/synplify. If the input file is system.xmp, the default project file name is system.prj.
Project Location	Specifies the directory for the synthesis project file. You must have write access to this directory. The default location for the file is the directory for the input project file. If the input file is E:\ise2syn\sBr2_Exbox2\system.xmp, the default project location is E:\ise2syn\sBr2_Exbox2\synplify.
Full Path	Automatically updated field that reflects the values specified in Project Name and Project Location.
Force White Box Cores	Specifies cores to be treated as white boxes. This option is only applicable to EDK designs. Use this option only if the tool errors out because it cannot read the core information because it is encrypted or in non-standard HDL. See *Force White Box Cores Option (EDK)*, on page 227 for more information about using this option. When this option is enabled, the utility uses .ngc files instead of HDL files for the core. Use this option for EDK cores and cores that use non-standard HDL. To specify multiple core names, separate the names with commas. Do not leave white space between the core name and the comma.

When you click Import, the ise2syn utility generates the following output files:

`.prj`	Synplify Pro or Synplify Premier synthesis project file
`_conv.sdc`	Synthesis constraint file, with converted ucf, ncf, or xcf constraints
`_unapplied.ucf`	Xilinx User Constraint file, that contains any unconverted constraints. This command uses the ucf2sdc utility to convert constraints. See *Constraint Translation*, on page 1058 in the *User Guide* and *Xilinx ucf2sdc Utility*, on page 1004 for details about unsupported constraints and the ucf2sdc utility, respectively.

Force White Box Cores Option (EDK)

Use this option to specify EDK cores to be white-boxed. Use this option only if the tool errors out because it cannot read the core information. By default, the tool automatically white boxes certain core configurations, and you do not have to enable the Force White Box Cores option for them. The tool automatically white boxes the following cores:

Xilinx specific core configurations	`ddr`
	`ddr2`
	`ddr2_sdram`
	`ipic_to_temac`
	`opb_ddr`
	`mch_opb_ddr`
	`plb_ddr2`
	`plb_master_llink`
	`plb_rapidio_lvds`
	`plb_temac`
	`plbv46_master`
	`plbv46_pcie`

Unsupported HDL constructs (in the Synplify Pro/Premier tool)	`clock_generator`
	`mpmc`
	`xps_mch_emc`

Encrypted cores	

Enable the Force White Box Cores option if your project has cores that are not listed in the previous table. For EDK project cores, specify any cores that are not listed in the previous table and which meet either of these criteria:

- Xilinx-specific core configurations that are not supported by the synthesis tools. For some cores, Xilinx tools configures the cores during synthesis based on some Xilinx specific directives.
- Xilinx cores that use non-standard HDL constructs and which are not supported by the synthesis tool compiler, like signal width mismatch for example.

To specify cores to be white-boxed, do the following:

1. Type the name of the core to be white-boxed in the Force White Box Cores option.
   - To locate the core name that corresponds to a HDL file, open the Synplify Pro/Premier project as text and locate the HDL file in it.

Copyright © 2013 Synopsys, Inc.

– The core name is displayed as a comment at the beginning. For example:

```
# Core Name = ppc405_virtex4, Core Instance = ppc405_0, Core Version = 2.01.b
add_file -verilog $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/verilog/mpmc2_sample_cycle.v
add_file -vhdl -lib ppc405_virtex4_v2_01_b $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/vhdl
add_file -vhdl -lib ppc405_virtex4_v2_01_b $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/vhdl
add_file -vhdl -lib ppc405_virtex4_v2_01_b $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/vhdl
add_file -vhdl -lib ppc405_virtex4_v2_01_b $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/vhdl
add_file -vhdl -lib ppc405_virtex4_v2_01_b $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/vhdl
add_file -vhdl -lib ppc405_virtex4_v2_01_b $XILINX_XMP_PRJ_DIR/pcores/ppc405_virtex4_v2_01_b/hdl/vhdl
add_file -vhdl -lib work $XILINX_XMP_PRJ_DIR/hdl/ppc405_0_wrapper.vhd
```

2. To specify a list of cores, separate the names with commas. Alternatively, specify a file that contains a comma-separated list of files to be white-boxed.

Import Altera SOPC Project Command

The Import Altera SOPC Project command allows you to include subsystems created with the SOPC Builder into the Synplify Pro or Synplify Premier project for synthesis. SOPC Builder is the Altera embedded system development tool, and is used to design embedded processor systems and subsystems in Altera FPGAs with embedded NIOS soft processor cores. The Import Altera SOPC Project command uses the sopc2syn utility, whose command line syntax is described in *Altera SOPC2Syn Utility*, on page 998.

For information about using these components and cores in your design, see *Working with SOPC Builder Components*, on page 1029 and *Including Altera Processor Cores Generated in SOPC Builder*, on page 1023 in the *User Guide*.

The following table describes the Import Altera SOPC Project menu options.

Command	Description
PTF File	Specifies the SOPC Builder plain test file (ptf) to be imported into the project.
Project Name	Specifies a name for the synthesis project.
Project Location	Specifies a location for the project.
Full Path	Specifies the full path to the project directory.
Preferred HDL Language	Sets the preferred HDL language in case of duplicate files. You can select one of the following: • VHDL • Verilog

Copyright © 2013 Synopsys, Inc.

Command	Description
Force White Box Cores	This is an advanced option that forces listed cores to be treated as white boxes. This means that the utility uses .vqm files for core definitions. This option has no default. • *string* is a comma-separated list of cores to be white-boxed. • *file* is a file containing a comma-separated list of cores to be white-boxed. See *Defining SOPC Components as Black Boxes and White Boxes*, on page 1031 of the *User Guide* for more information.
Wrap	This advanced option enables or disables the generation of missing core wrappers. • When enabled, it generates missing core wrappers. • When disabled, it does not generate missing core wrappers. This is the recommended setting and the default.
Clear Box	This option specifies whether or not to use the clear box flow for encrypted cores. This also determines if the cores are treated as black boxes. (See *Defining SOPC Components as Black Boxes and White Boxes*, on page 1031 of the *User Guide*.) • When enabled, it uses the clear box flow. The tool adds black box wrapper files for encrypted cores. It adds the HDL files for the encrypted cores to the project, but only uses them to run place-and-route. This is the default. • When disabled, it does not use the clear box flow. Black box wrapper files for the encrypted cores are added to the project and the HDL files for the encrypted cores are copied to the place-and-route directory.

Import Altera QSF Project Command

Use the Import Altera QSF Project command to converts an Altera Quartus project into a Synplify Pro or Synplify Premier project for synthesis. This command uses the qsf2syn utility, the syntax for which is described in *Altera qsf2syn Utility*, on page 1003.

For detailed information about using the command and converting Quartus projects, see *Importing Projects from Quartus*, on page 1033 in the *User Guide*.

The following table describes the Import Altera QSF Project menu options.

Command	Description
Altera QSF Project File	Specifies the .qsf file for the Quartus project to be imported. See *Imported Quartus Project Settings and Timing Constraints*, on page 1037 in the *User Guide* for a list of the supported constraints and project settings.
Project Name	Specifies a name for the synthesis project to be generated. See *Synthesis Files Generated After Importing the Quartus Project*, on page 1036 in the *User Guide* for a list of the files that are created.

Copyright © 2013 Synopsys, Inc.

Command	Description
Project Location	Specifies a location for the synthesis project.
Full Path	Echoes the full path to the synthesis project directory.
Continue Previous Translation	Lets you continue translation from the point at which you stopped to debug or fix your design. For details on using this option, see *Importing Quartus Projects*, on page 1033 in the *User Guide*.

Import Xilinx Constraints

Use the Import Xilinx Constraints command to translate UCF and NCD constraints from ISE or DCP constraints from Vivado for placement and timing information for the PDC file for Xilinx Physical Plus. For information about the Physical Plus feature, see *Xilinx Physical Plus*, on page 64.

Copyright © 2013 Synopsys, Inc.

The following table describes the options you can set:

Field	Description
ISE Place and Route	
Edif Input	Specifies the input EDIF files to be used in Xilinx Physical Plus.
Import UCF Constraints • Select UCF Files • Select PDC output for UCF constraints	Specifies the UCF constraint files to import to the Physical Plus project. • Specifies the UCF files to be imported for the specified EDIF files. • Imports the UCF design constraints to the specified physical design constraints (PDC) file. *ucfFileName*_ucf.pdc Note: Import UCF Constraints translate timing and/or placement contraints from the UCF file.
Import I/O and Clock Component Placement • Select NCD File • Select PDC output for NCD constraints • Use for Physical Plus or Use for Logical Synthesis	Imports I/O and clock component placement for Physical Plus: • Specifies the Xilinx P&R (NCD) file, from which to import I/O and clock component placement. • Imports the NCD placement constraints to the specified physical design constraints (PDC) file. *designName*_ncd.pdc • Specifies whether to use the physical (PDC) or logical (SDC) file if you are starting from the RTL level for the imported NCD constraints. Note: Physical Plus requires I/O and clock components to have placement constraints. If they are not available in the UCF or SDC file, then they can be extracted from the NCD file.
Extract RAM/DSP Placement	When enabled, imports RAM and DSP placement, as well as, I/O and clock components for Physical Plus. Note: Optionally, RAM/DSP placement can be locked from a previous implementation.
Add PDC constraints to project	Specifies the location of the project file to add the PDC constraints file.
Vivado Place and Route	
Edif Input	Specifies the input EDIF files to be used in Xilinx Physical Plus.

Copyright © 2013 Synopsys, Inc.

Select Post-Place Vivado Database (DCP)	Specifies the Vivado Design Checkpoint files (DCP) to import placement and routing information for the Physical Plus project.
Import I/O and Clock Component Placement • Specify Physical PDC or Logical (SDC) output for XDC placement constraints • Use for Physical Plus or Use for Logical Synthesis	Imports I/O and clock component placement for Physical Plus: • Imports the XDC placement constraints to the specified physical design constraints (PDC) file. *designName*_xdc_place.pdc • Specifies whether to use the physical (PDC) or logical (SDC) file if you are starting from the RTL level for the imported constraints.
Include RAM & DSP Placement	When enabled, imports RAM and DSP placement, as well as, I/O and clock components for Physical Plus.
Add constraints to project	Specifies the location of the project file to add the constraints file.

Import IP Package Command

Use the Import IP Package command to help you import IP core files to your Project view.

The following table describes the Import IP Package menu options.

Copyright © 2013 Synopsys, Inc.

Command	Description
IP Package	Location of the consolidated ZIP file containing the IP core files. The file format can be .zip or .tar. Use the Browse button to help you locate the IP package.
IP Directory	Directory location of the consolidated IP core files. Use the Browse button to help you locate the IP directory.
Package Name	The name of the top-level module for the IP core.

Launch System Designer Command

Use the Launch System Designer command to bring up the System Designer GUI. This tool helps you build embedded IP cores which you can include in your project file for synthesis.

For information about the System Designer feature:

- From the Project view of the synthesis tool, select Help->Online Documents and bring up the *User Guide* and/or *Tutorial* PDF documents.
- From the System Designer tool, select Help->Help Contents to bring up an HTML based help system for the tool.

Copyright © 2013 Synopsys, Inc.

Run Menu

You use the Run menu to perform the following tasks:

- Compile a design, without mapping it.
- Synthesize (compile and map) or resynthesize a design.
- Check design syntax and synthesis code, and check source code errors.
- Check constraint syntax and how/if constraints are applied to the design.
- Run Tcl scripts.
- Run all implementations at once.
- Check the status of the current job.

The following table describes the Run menu commands.

Command	Description
Run	Synthesizes (compiles and maps) the top-level design. For the compile point flow, this command also synthesizes any compile points whose constraints, implementation options, or source code changed since the last synthesis run (Synplify Pro and Synplify Premier tools only). You can view the result of design synthesis in the RTL and Technology views. Same as clicking the Run button in the Project view.
Resynthesize All	Resynthesizes (compiles and maps) the entire design, including the top level and *all compile points*, whether or not their constraints, implementation options, or source code changed since the last synthesis. If you do *not* want to force a *recompilation of all compile points*, then use Run->Run instead.
Compile Only	Compiles the design into technology-independent high-level structures. You can view the result in the RTL view.

Copyright © 2013 Synopsys, Inc.

Command	Description
Write Output Netlist Only	Generates an output netlist after synthesis has been run. This command generates the netlists you specify on the Implementation Results tab of the Implementation Options dialog box. You can also use this command in an incremental timing analysis flow. See Generating Custom Timing Reports with STA, on page 729 for details.
Estimate Area	For details, see *Run Menu: Design Planner Commands*, on page 551.
Compile Physical Hierarchy	For details, see *Run Menu: Design Planner Commands*, on page 551.
FSM Explorer	*Synplify Pro and Synplify Premier.* Analyzes finite state machines contained in a design, and selects the optimum encoding style. This menu command is not available in some views.
Translate Vendor IO	Displays a dialog box where you can specify an Altera PIN or Xilinx PAD file to translate into an FPGA constraint file (sdc). See *Translate Vendor IO Command*, on page 242.
Syntax Check	Runs a syntax check on design code. The status bar at the bottom of the window displays any error messages. If the active window shows an HDL file, then the command checks only that file; otherwise, it checks all project source code files.
Synthesis Check	Runs a synthesis check on your design code. This includes a syntax check and a check to see if the synthesis tool could map the design to the hardware. No optimizations are carried out. The status bar at the bottom of the window displays any error messages. If the active window shows an HDL file, then the command checks only that file; otherwise, it checks all project source code files.
Constraint Check	Checks the syntax and applicability of the timing constraints in the .sdc file for your project and generates a report (*projectName*_cck.rpt). The report contains information on the constraints that can be applied, cannot be applied because objects do not exist, and wildcard expansion on the constraints. See *Constraint Checking Report*, on page 629.

Command	Description
Arrange VHDL files	Reorders the VHDL source files for synthesis.
Launch Identify Instrumentor	Launches the Identify instrumentor. For more information, see: Working with the Identify Tools, on page 880 of the *User Guide*. To launch the Identify instrumentor in batch mode, use the set_option -identify_debug_mode 1 Tcl command.
Launch Identify Debugger	Launches the Identify debugger tool. For more information, see: Working with the Identify Tools, on page 880 of the *User Guide*. To launch the Identify debugger in batch mode, use the set_option -identify_debug_mode 1 Tcl command.
Launch SYNCore	Opens the Synopsys FPGA IP Core Wizard. This tool helps you build IP blocks such as memory or FIFO models for your design. See the *Launch SYNCore Command*, on page 245 for details.
Configure and Launch VCS Simulator	Allows you to configure and launch the VCS simulator. See *Configure and Launch VCS Simulator Command*, on page 274.
Run Tcl Script	Displays the Open dialog box, where you choose a Tcl script to run. See *Run Tcl Script Command*, on page 240.
Run All Implementations	*Synplify Pro and Synplify Premier.* Runs all selected implementations for one project at the same time. See *Run All Implementations Command*, on page 241.
Job Status	During compilation, tells you the name of the current job, and gives you the runtime and directory location of your design. This option is enabled during synthesis. See *Job Status Command*, on page 243. Clicking in the status area of the Project view is a shortcut for this command.

Copyright © 2013 Synopsys, Inc.

Command	Description
Next Error/Warning	Shows the next error or warning in your source code file.
Previous Error/Warning	Shows the previous error or warning in your source code file.
Log File Message Filter	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. For more information, see *Log File Message Controls*, on page 528.

Run Tcl Script Command

Select Run->Run Tcl Script to display the Open dialog box, where you specify the Tcl script file to execute. The File name area is filled automatically with the wildcard string "*.tcl", corresponding to Tcl files.

This dialog box is the same as that displayed with File->Open, except that no Open as read-only check box is present. See *Open Project Command*, on page 146, for an explanation of the features in the Open dialog box.

Run All Implementations Command

Select Run->Run All Implementations to run selected implementations in batch mode. To use the Batch Run Setup dialog box, check one or more implementations from the list displayed and click the Run button.

Copyright © 2013 Synopsys, Inc.

Translate Vendor IO Command

Select Run->Translate Vendor IO to translate the Xilinx PAD or Altera PIN file into an sdc constraint file. This command displays the appropriate translation dialog box for your target vendor (Translate Xilinx PAD File to Synplicity Scope (SDC) File or Translate Altera PIN File to Synplicity Scope (SDC) File), where you specify the input and output files for the translation. Enable Add to project to add the constraint file to your project.

For information on using this command, see *Translating Pin Location Files*, on page 321 in the *User Guide*.

Job Status Command

Select Run->Job Status to monitor the synthesis jobs that are running, their run times, and their associated commands. This information appears in the Job Status dialog box. This dialog box is also displayed when you click in the status area of the Project view (see The Project View, on page 62).

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

Launch Identify Instrumentor Command

The Launch Identify Instrumentor command lets you start the Identify instrumentor from within the synthesis interface. Before you can use this command, you must define an Identify implementation in the project view. For a description of the work flow using the Identify debugger, see Working with the Identify Tools, on page 880 in the *User Guide*.

Configure Identify Launch Dialog Box

The Configure Identify Launch dialog box is automatically displayed when the location of the Identify executable has not been previously defined.

Copyright © 2013 Synopsys, Inc.

Command	Description
Use current Identify installation:	A pointer to the current installation of the Identify software. Click the radio button to use the displayed version.
Locate Identify Installation (identify_instrumentor)	A pointer to the Identify install directory. Use the (...) button to navigate to the directory location.
Identify License Option	Radio buttons to select the Identify license option. Select Use current synthesis license when only a single TSL license is available; select Use separate Identify Instrumentor license when multiple licenses are available. With a single TSL license, you are prohibited from compiling or mapping in the synthesis tool while the Identify instrumentor is open.

Copyright © 2013 Synopsys, Inc.

Launch Identify Debugger Command

The Launch Identify Debugger command lets you start the Identify Debugger software from within the synthesis interface. Before you can use this command, you must have an active Identify implementation and an instrumented design. For a description of the work flow using the Identify/Identify RTL Debugger software, see Working with the Identify Tools, on page 880 in the *User Guide*.

Launch SYNCore Command

The SYNCore wizard helps you build IP cores. The wizard can compile RAM and ROM memories including a byte-enable RAM, a FIFO, an adder/subtractor, and a counter. The resulting Verilog models can be synthesized and simulated. For details about using the wizard to build these models, see the following sections in the user guide:

- *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

To start the SYNCore wizard, select Run->Launch SYNCore and:

- Select sfifo_model and click Ok to start the FIFO wizard, described in *SYNCore FIFO Wizard*, on page 247.
- Select ram_model and click Ok to start the RAM wizard, described in *SYNCore RAM Wizard*, on page 257.
- Select byte_en_ram and click Ok to start the byte-enable RAM wizard, described in *SYNCore Byte-Enable RAM Wizard*, on page 261.
- Select rom_model and click Ok to start the ROM wizard, described in *SYNCore ROM Wizard*, on page 264.
- Select addnsub_model and click Ok to start the adder/subtractor wizard, described in *SYNCore Adder/Subtractor Wizard*, on page 268.

Copyright © 2013 Synopsys, Inc.

- Select counter_model and click Ok to start the counter wizard, described in *SYNCore Counter Wizard*, on page 272.

Each SYNCore wizard has three tabs at the top, and buttons below, which are described here:

Parameters	Consists of a multiple-screen wizard that lets you set parameters for that model. See *SYNCore FIFO Wizard*, on page 247, *SYNCore RAM Wizard*, on page 257, *SYNCore Byte-Enable RAM Wizard*, on page 261, *SYNCore ROM Wizard*, on page 264, *SYNCore Adder/Subtractor Wizard*, on page 268, or *SYNCore Counter Wizard*, on page 272 for details about the options you can set.
Core Overview	Contains basic information about the kind of model you are creating.

Generate	Generates the model with the parameters you specify in the wizard.
Sync FIFO Info, RAM Info, BYTE ENABLE RAM Info, ROM Info, ADDnSUB Info, COUNTER Info	Opens a window with technical information about the corresponding model.

SYNCore FIFO Wizard

The following describe the parameters you can set in the FIFO wizard, which opens when you select sfifo_model:

- SYNCore FIFO Parameters Page 1, on page 247
- SYNCore FIFO Parameters Page 2, on page 249
- SYNCore FIFO Parameters Page 3, on page 251
- SYNCore FIFO Parameters Page 4, on page 253
- SYNCore FIFO Parameters Page 5, on page 255

SYNCore FIFO Parameters Page 1

The page 1 parameters define the FIFO. Data is written/read on the rising edge of the clock.

Copyright © 2013 Synopsys, Inc.

Parameter	Function
Component Name	Specifies a name for the FIFO. This is the name that you instantiate in your design file to create an instance of the SYNCore FIFO in your design. Do not use spaces.
Directory	Indicates the directory where the generated files will be stored. Do not use spaces.
Filename	Specifies the name of the generated file containing the HDL description of the generated FIFO. Do not use spaces.
Width	Specifies the width of the FIFO data input and output. It must be within the valid range.
Depth	Specifies the depth of the FIFO. It must be within the valid range.

Copyright © 2013 Synopsys, Inc.

SYNCore FIFO Parameters Page 2

Copyright © 2013 Synopsys, Inc.

The page 2 parameters let you specify optional handshaking flags for FIFO write operations. When you specify a flag, the symbol on the left reflects your choice. Data is written/read on the rising edge of the clock.

Parameter	Function
Full Flag	Specifies a Full signal, which is asserted when the FIFO memory queue is full and no more writes can be performed until data is read. Enabling this option makes the Active High and Active Low options (FULL_FLAG_SENSE parameter) available for the signal. See *Full/Almost Full Flags*, on page 1033 and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter.
Almost Full Flag	Specifies an Almost_full signal, which is asserted to indicate that there is one location left and the FIFO will be full after one more write operation. Enabling this option makes the Active High and Active Low options available for the signal (AFULL_FLAG_SENSE parameter). See *Full/Almost Full Flags*, on page 1033 and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter.
Overflow Flag	Specifies an Overflow signal, which is asserted to indicate that the write operation was unsuccessful because the FIFO was full. Enabling this option makes the Active High and Active Low options available for the signal (OVERFLOW_FLAG_SENSE parameter). See *Handshaking Flags*, on page 1035 f and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter.
Write Acknowledge Flag	Specifies a Write_ack signal, which is asserted at the completion of a successful write operation. Enabling this option makes the Active High and Active Low options (WACK_FLAG_SENSE parameter) available for the signal. See *Handshaking Flags*, on page 1035 and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter.
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

Copyright © 2013 Synopsys, Inc.

SYNCore FIFO Parameters Page 3

The page 3 parameters let you specify optional handshaking flags for FIFO read operations. Data is written/read on the rising edge of the clock.

Parameter	Function
Empty Flag	Specifies an Empty signal, which is asserted when the memory queue for the FIFO is empty and no more reads can be performed until data is written. Enabling this option makes the Active High and Active Low options (EMPTY_FLAG_SENSE parameter) available for the signal. See *Empty/Almost Empty Flags*, on page 1034 and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter.

Copyright © 2013 Synopsys, Inc.

Parameter	Function
Almost Empty Flag	Specifies an Almost_empty signal, which is asserted when there is only one location left to be read. The FIFO will be empty after one more read operation. Enabling this option makes the Active High and Active Low options (AEMPTY_FLAG_SENSE parameter) available for the signal. See *Empty/Almost Empty Flags, on page 1034* and *FIFO Parameters, on page 1031* for descriptions of the flag and parameter.
Underflow Flag	Specifies an Underflow signal, which is asserted to indicate that the read operation was unsuccessful because the FIFO was empty. Enabling this option makes the Active High and Active Low options (UNDRFLW_FLAG_SENSE parameter) available for the signal. See *Handshaking Flags, on page 1035* and *FIFO Parameters, on page 1031* for descriptions of the flag and parameter.
Read Acknowledge Flag	Specifies a Read_ack signal, which is asserted at the completion of a successful read operation. Enabling this option makes the Active High and Active Low options (RACK_FLAG_SENSE parameter) available for the signal. See *Handshaking Flags, on page 1035* and *FIFO Parameters, on page 1031* for descriptions of the flag and parameter.
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

SYNCore FIFO Parameters Page 4

The page 4 parameters let you specify optional handshaking flags for FIFO programmable full operations. To use these options, you must have a Full signal specified. See *FIFO Programmable Flags*, on page 1036 for details and *FIFO Parameters*, on page 1031 for a list of the FIFO parameters. Data is written/read on the rising edge of the clock.

Parameter	Function
Programmable Full Flag	Specifies a Prog_full signal, which indicates that the FIFO has reached a user-defined full threshold. You can only enable this option if you set Full Flag on page 2. When it is enabled, you can specify other options for the Prog_Full signal (PFULL_FLAG_SENSE parameter). See *Programmable Full*, on page 1037 and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter.

Copyright © 2013 Synopsys, Inc.

Parameter	Function
Single Programmable Full Threshold Constant	Specifies a Prog_full signal with a single constant defining the assertion threshold (PGM_FULL_TYPE=1 parameter). See *Programmable Full with Single Threshold Constant*, on page 1037 for details. Enabling this option makes Full Threshold Assert Constant available.
Multiple Programmable Full Threshold Constant	Specifies a Prog_full signal (PGM_FULL_TYPE=2 parameter), with multiple constants defining the assertion and de-assertion thresholds. See *Programmable Full with Multiple Threshold Constants*, on page 1038 for details. Enabling this option makes Full Threshold Assert Constant and Full Threshold Negate Constant available.
Full Threshold Assert Constant	Specifies a constant that is used as a threshold value for asserting the Prog_full signal It sets the PGM_FULL_THRESH parameter for PGM_FULL_TYPE=1 and the PGM_FULL_ATHRESH parameter for PGM_FULL_TYPE=2.
Full Threshold Negate Constant	Specifies a constant that is used as a threshold value for de-asserting the Prog_full signal (PGM_FULL_NTHRESH parameter).
Single Programmable Full Threshold Input	Specifies a Prog_full signal (PGM_FULL_TYPE=3 parameter), with a threshold value specified dynamically through a Prog_full_thresh input port during the reset state. See *Programmable Full with Single Threshold Input*, on page 1038 for details. Enabling this option adds the Prog_full_thresh input port to the FIFO.
Multiple Programmable Full Threshold Input	Specifies a Prog_full signal (PGM_FULL_TYPE=4 parameter), with threshold assertion and deassertion values specified dynamically through input ports during the reset state. See *Programmable Full with Multiple Threshold Constants*, on page 1038 for details. Enabling this option adds the Prog_full_thresh_assert and Prog_full_thresh_negate input ports to the FIFO.
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

Copyright © 2013 Synopsys, Inc.

SYNCore FIFO Parameters Page 5

These options specify optional handshaking flags for FIFO programmable empty operations. To use these options, you first specify an Empty signal on page 3. See *FIFO Programmable Flags*, on page 1036 for details and *FIFO Parameters*, on page 1031 for a list of the FIFO parameters. Data is written/read on the rising edge of the clock.

Parameter	Function
Programmable Empty Flag	Specifies a Prog_empty signal (PEMPTY_FLAG_SENSE parameter), which indicates that the FIFO has reached a user-defined empty threshold. See *Programmable Empty*, on page 1040 and *FIFO Parameters*, on page 1031 for descriptions of the flag and parameter. Enabling this option makes the other options available to specify the threshold value, either as a constant or through input ports. You can also specify single or multiple thresholds for each of these options.

Copyright © 2013 Synopsys, Inc.

Parameter	Function
Single Programmable Empty Threshold Constant	Specifies a Prog_empty signal (PGM_EMPTY_TYPE=1 parameter), with a single constant defining the assertion threshold. See *Programmable Empty with Single Threshold Input*, on page 1041 for details. Enabling this option makes Empty Threshold Assert Constant available.
Multiple Programmable Empty Threshold Constant	Specifies a Prog_empty signal (PGM_EMPTY_TYPE=2 parameter), with multiple constants defining the assertion and de-assertion thresholds. See *Programmable Empty with Multiple Threshold Constants*, on page 1041 for details. Enabling this option makes Empty Threshold Assert Constant and Empty Threshold Negate Constant available.
Empty Threshold Assert Constant	Specifies a constant that is used as a threshold value for asserting the Prog_empty signal. It sets the PGM_EMPTY_THRESH parameter for PGM_EMPTY_TYPE=1 and the PGM_EMPTY_ATHRESH parameter for PGM_EMPTY_TYPE=2.
Empty Threshold Negate Constant	Specifies a constant that is used as a threshold value for de-asserting the Prog_empty signal (PGM_EMPTY_NTHRESH parameter).
Single Programmable Empty Threshold Input	Specifies a Prog_empty signal (PGM_EMPTY_TYPE=3 parameter), with a threshold value specified dynamically through a Prog_empty_thresh input port during the reset state. See *Programmable Empty with Single Threshold Input*, on page 1041 for details. Enabling this option adds the Prog_full_thresh input port to the FIFO.
Multiple Programmable Empty Threshold Input	Specifies a Prog_empty signal (PGM_EMPTY_TYPE=4 parameter), with threshold assertion and deassertion values specified dynamically through Prog_empty_thresh_assert and Prog_empty_thresh_negate input ports during the reset state. See *Programmable Empty with Multiple Threshold Inputs*, on page 1042 for details. Enabling this option adds the input ports to the FIFO.
Active High	Sets the specified signal to active high (1).
Active Low	Sets the specified signal to active low (0).

Copyright © 2013 Synopsys, Inc.

Parameter	Function
Number of Valid Data in FIFO	Specifies the Data_cnt signal for the FIFO output. This signal contains the number of words in the FIFO in the read domain.

SYNCore RAM Wizard

The following describe the parameters you can set in the RAM wizard, which opens when you select ram_model:

- SYNCore RAM Parameters Page 1, on page 257
- SYNCore RAM Parameters Pages 2 and 3, on page 259

SYNCore RAM Parameters Page 1

Copyright © 2013 Synopsys, Inc.

Component Name	Specifies the name of the component. This is the name that you instantiate in your design file to create an instance of the SYNCore RAM in your design. Do not use spaces. For example:
	`ram101 <ComponentName> (`
	`  .PortAClk(PortAClk)`
	`  , .PortAAddr(PortAAddr)`
	`  , .PortADataIn(PortADataIn)`
	`  , .PortAWriteEnable(PortAWriteEnable)`
	`  , .PortBDataIn(PortBDataIn)`
	`  , .PortBAddr(PortBAddr)`
	`  , .PortBWriteEnable(PortBWriteEnable)`
	`  , .PortADataOut(PortADataOut)`
	`  , .PortBDataOut(PortBDataOut)`
	`);`

Directory	Specifies the directory where the generated files are stored. Do not use spaces. The following files are created: • filelist.txt – lists files written out by SYNCore • options.txt – lists the options selected in SYNCore • readme.txt – contains a brief description and known issues • syncore_ram.v – Verilog library file required to generate RAM model • testbench.v – Verilog testbench file for testing the RAM model • instantiation_file.vin – describes how to instantiate the wrapper file • *component*.v – RAM model wrapper file generated by SYNCore Note that running the Memory Compiler wizard in the same directory overwrites the existing files.
Filename	Specifies the name of the generated file containing the HDL description of the compiled RAM. Do not use spaces.
Data Width	Is the width of the data you need for the memory. The unit used is the number of bits.
Address Width	Is the address depth you need for the memory. The unit used is the number of bits.
Single Port	When enabled, generates a single-port RAM.

Dual Port	When enabled, generates a dual-port RAM.
Single Clock	When enabled, generates a RAM with a single clock for dual-port configurations.
Separate Clocks for Each Port	When enabled, generates separate clocks for each port in dual-port RAM configurations.

SYNCore RAM Parameters Pages 2 and 3

The port implementation parameters on pages 2 and 3 are identical, but page 2 applies to Port A (single- and dual-port configurations), and page 3 applies to Port B (dual-port configurations only). The following figure shows the parameters on page 2 for Port A.

Read and Write Access	Specifies that the port can be accessed by both read and write operations.
Read Only Access	Specifies that the port can only be accessed by read operations.
Write Only Access	Specifies that the port can only be accessed by write operations.
Use Write Enable	Includes write-enable control. The RAM symbol on the left reflects the selections you make.

Copyright © 2013 Synopsys, Inc.

Register Read Address	Adds registers to the read address lines. The RAM symbol on the left reflects the selections you make.
Register Outputs	Adds registers to the write address lines when you specify separate read/write addressing. The register outputs are always enabled. The RAM symbol on the left reflects the selections you make.
Synchronous Reset	Individually synchronizes the reset signal with the clock when you enable Register Outputs. The RAM symbol on the left reflects the selections you make.
Read before Write	Specifies that the read operation takes place before the write operation for port configurations with both read and write access (Read And Write Access is enabled). For a timing diagram, see *Read Before Write*, on page 1050.
Read after Write	Specifies that the read operation takes place after the write operation for port configurations with both read and write access (Read And Write Access is enabled). For a timing diagram, see *Write Before Read*, on page 1051.
No Read on Write	Specifies that no read operation takes place when there is a write operation for port configurations with both read and write access (Read And Write Access is enabled). For a timing diagram, see *No Read on Write*, on page 1052.

Copyright © 2013 Synopsys, Inc.

SYNCore Byte-Enable RAM Wizard

The following describes the parameters you can set in the byte-enable RAM wizard, which opens when you select byte_en_ram.

- SYNCore Byte-Enable RAM Parameters Page 1, on page 261
- SYNCore Byte-Enable RAM Parameters Pages 2 and 3, on page 262

SYNCore Byte-Enable RAM Parameters Page 1

Copyright © 2013 Synopsys, Inc.

Component Name	Specifies the name of the component. This is the name that you instantiate in your design file to create an instance of the SYNCore byte-enable RAM in your design. Do not use spaces.
Directory	Specifies the directory where the generated files are stored. Do not use spaces. The following files are created: • filelist.txt – lists files written out by SYNCore • options.txt – lists the options selected in SYNCore • readme.txt – contains a brief description and known issues • syncore_be_ram_sdp.v – SystemVerilog library file required to generate single or simple dual-port, byte-enable RAM model • syncore_be_ram_tdp.v – SystemVerilog library file required to generate true dual-port byte-enable RAM model • testbench.v – Verilog testbench file for testing the byte-enable RAM model • instantiation_file.vin – describes how to instantiate the wrapper file • *component*.v – Byte-enable RAM model wrapper file generated by SYNCore Note that running the byte-enable RAM wizard in the same directory overwrites the existing files.
Filename	Specifies the name of the generated file containing the HDL description of the compiled byte-enable RAM. Do not use spaces.
Address Width	Specifies the address depth for Ports A and B. The unit used is the number of bits; the default is 2.
Data Width	Specifies the width of the data for Ports A and B. The unit used is the number of bits; the default is 2.
Write Enable Width	Specifies the write enable width for Ports A and B. The unit used is the number of byte enables; the default is 2, the maximum is 4.
Single Port	When enabled, generates a single-port, byte-enable RAM (automatically enables single clock).
Dual Port	When enabled, generates a dual-port, byte-enable RAM (automatically enables separate clocks for each port).

SYNCore Byte-Enable RAM Parameters Pages 2 and 3

The port implementation parameters on pages 2 and 3 are identical, but page 2 applies to Port A (single- and dual-port configurations), and page 3 applies to Port B (dual-port configurations only). The following figure shows the parameters on page 2 for Port A.

Copyright © 2013 Synopsys, Inc.

Read and Write Access	Specifies that the port can be accessed by both read and write operations (only mode allowed for single-port configurations).
Read Only Access	Specifies that the port can only be accessed by read operations (dual-port mode only).
Write Only Access	Specifies that the port can only be accessed by write operations (dual-port mode only).
Register address bus AddrA/B	Adds registers to the read address lines.
Register output data bus RdDataA/B	Adds registers to the read data lines. By default, the read data register is enabled.

Copyright © 2013 Synopsys, Inc.

Reset for RdDataA/B	Specifies the reset type for registered read data: • Reset type is synchronous when Reset for RdDataA/B is enabled • Reset type is no reset when Reset for RdDataA/B is disabled
Specify output data on reset	Specifies reset value for registered read data (applies only when RdDataA/B is enabled): • Default value of '1' for all bits – sets read data to all 1's on reset • Specify Reset value for RdDataA/B – specifies reset value for read data; when enabled, value is entered in adjacent field
Write Enable for Port A/B	Specifies the write enable level for Port A/B. Default is Active High.

SYNCore ROM Wizard

The following describe the parameters you can set in the ROM wizard, which opens when you select rom_model:

- SYNCore ROM Parameters Page 1, on page 265
- SYNCore ROM Parameters Pages 2 and 3, on page 266
- SYNCore ROM Parameters Page 4, on page 268

SYNCore ROM Parameters Page 1

Component Name	Specifies the name of the component. This is the name that you instantiate in your design file to create an instance of the SYNCore ROM in your design. Do not use spaces.
Directory	Specifies the directory where the generated files are stored. Do not use spaces. The following files are created: filelist.txt – lists files written out by SYNCore options.txt – lists the options selected in SYNCore readme.txt – contains a brief description and known issues syncore_rom.v – Verilog library file required to generate ROM model testbench.v – Verilog testbench file for testing the ROM model instantiation_file.vin – describes how to instantiate the wrapper file *component*.v – ROM model wrapper file generated by SYNCore Note that running the ROM wizard in the same directory overwrites the existing files.
File Name	Specifies the name of the generated file containing the HDL description of the compiled ROM. Do not use spaces.

Copyright © 2013 Synopsys, Inc.

Read Data Width	Specifies the read data width of the ROM. The unit used is the number of bits and ranges from 2 to 256. Default value is 8. The read data width is common to both Port A and Port B. The corresponding file parameter is DATA_WIDTH=n.
ROM address width	Specifies the address depth for the memory. The unit used is the number of bits. Default value is 10. The corresponding file parameter is ADD_WIDTH=n.
Single Port Rom	When enabled, generates a single-port ROM. The corresponding file parameter is CONFIG_PORT="single".
Dual Port Rom	When enabled, generates a dual-port ROM. The corresponding file parameter is CONFIG_PORT="dual".

SYNCore ROM Parameters Pages 2 and 3

The port implementation parameters on pages 2 and 3 are the same; page 2 applies to Port A (single- and dual-port configurations), and page 3 applies to Port B (dual-port configurations only).

Register address bus AddrA	Used with synchronous ROM configurations to register the read address. When checked, also allows chip enable to be configured.
Register output data bus DataA	Used with synchronous ROM configurations to register the data outputs. When checked, also allows chip enable to be configured.
Asynchronous Reset	Sets the type of reset to asynchronous (Configure Reset Options must be checked). Configuring reset also allows the output data pattern on reset to be defined. The corresponding file parameter is RST_TYPE_A=1/RST_TYPE_B=1.
Synchronous Reset	Sets the type of reset to synchronous (Configure Reset Options must be checked). Configuring reset also allows the output data pattern on reset to be defined. The corresponding file parameter is RST_TYPE_A=0/RST_TYPE_B=0.
Active High Enable	Sets the level of the chip enable to high for synchronous ROM configurations. The corresponding file parameter is EN_SENSE_A=1/EN_SENSE_B=1.
Active Low Enable	Sets the level of the chip enable to low for synchronous ROM configurations. The corresponding file parameter is EN_SENSE_A=0/EN_SENSE_B=0.
Default value of '1' for all bits	Specifies an output data pattern of all 1's on reset. The corresponding file parameter is RST_DATA_A={n{1'b1} }/RST_DATA_B={n{1'b1} }.
Specify reset value for DataA/DataB	Specifies a user-defined output data pattern on reset. The pattern is defined in the adjacent field. The corresponding file parameter is RST_TYPE_A=pattern/RST_TYPE_B=pattern.

Copyright © 2013 Synopsys, Inc.

SYNCore ROM Parameters Page 4

Binary	Specifies binary-formatted initialization file.
Hexadecimal	Specifies hexadecimal-formatted initial file.
Initialization File	Specifies path and filename of initialization file. The corresponding file parameter is INIT_FILE="*filename*".

SYNCore Adder/Subtractor Wizard

The following describe the parameters you can set in the adder/subtractor wizard, which opens when you select addnsub_model:

- SYNCore Adder/Subtractor Parameters Page 1, on page 269
- SYNCore Adder/Subtractor Parameters Page 2, on page 270

SYNCore Adder/Subtractor Parameters Page 1

Component Name	Specifies a name for the adder/subtractor. This is the name that you instantiate in your design file to create an instance of the SYNCore adder/subtractor in your design. Do not use spaces.
Directory	Indicates the directory where the generated files will be stored. Do not use spaces. The following files are created: • filelist.txt – lists files written out by SYNCore • options.txt – lists the options selected in SYNCore • readme.txt – contains a brief description and known issues • syncore_ADDnSUB.v – Verilog library file required to generate adder/subtractor model • testbench.v – Verilog testbench file for testing the adder/subtractor model • instantiation_file.vin – describes how to instantiate the wrapper file • *component*.v – adder/subtractor model wrapper file generated by SYNCore Note that running the wizard in the same directory overwrites any existing files.
Filename	Specifies the name of the generated file containing the HDL description of the generated adder/subtractor. Do not use spaces.

Copyright © 2013 Synopsys, Inc.

Adder	When enabled, generates an adder (the corresponding file parameter is ADD_N_SUB ="ADD").
Subtractor	When enabled, generates a subtractor (the corresponding file parameter is ADD_N_SUB ="SUB").
Adder/Subtractor	When enabled, generates a dynamic adder/subtractor (the corresponding file parameter is ADD_N_SUB ="DYNAMIC").

SYNCore Adder/Subtractor Parameters Page 2

Port A Width	Specifies the width of port A (the corresponding file parameter is PORT_A_WIDTH=n).
Register Input A	Used with synchronous adder/subtractor configurations to register port A. When checked, also allows clock enable and reset to be configured (the corresponding file parameter is PORTA_PIPELINE_STAGE='0' or '1').
Clock Enable for Register A	Specifies the enable for port A register.
Reset for Register A	Specifies the reset for port A register.
Constant Value Input	Specifies port B as a constant input when checked and allows you to enter a constant value in the Constant Value/Port B Width field (the corresponding file parameter is CONSTANT_PORT ='0').
Enable Port B	Specifies port B as an input when checked and allows you to enter a port B width in the Constant Value/Port B Width field (the corresponding file parameter is CONSTANT_PORT ='1').
Constant Value/Port B Width	Specifies either a constant value or port B width depending on Constant Value Input and Enable Port B selection (the corresponding file parameters are CONSTANT_VALUE= n or PORT_B_WIDTH=n).
Register Input B	Used with synchronous adder/subtractor configurations to register port B. When checked, also allows clock enable and reset to be configured (the corresponding file parameter is PORTB_PIPELINE_STAGE='0' or '1').
Clock Enable for Register B	Specifies the enable for the port B register.
Reset for Register B	Specifies the reset for the port B register.
Output port Width	Specifies the width of the output port (the corresponding file parameter is PORT_OUT_WIDTH=n).
Register output PortOut	Used with synchronous adder/subtractor configurations to register the output port. When checked, also allows clock enable and reset to be configured (the corresponding file parameter is PORTOUT_PIPELINE_STAGE='0' or '1'.
Clock Enable for Register PortOut	Specifies the enable for the output port register.

Copyright © 2013 Synopsys, Inc.

Reset for Register PortOut	Specifies the reset for the output port register.
Synchronous Reset	Sets the type of reset to synchronous (the corresponding file parameter is RESET_TYPE='0').
Asynchronous Reset	Sets the type of reset to asynchronous (the corresponding file parameter is RESET_TYPE='1').

SYNCore Counter Wizard

The following describe the parameters you can set in the ROM wizard, which opens when you select counter_model:

- SYNCore Counter Parameters Page 1, on page 272
- SYNCore Counter Parameters Page 2, on page 274

SYNCore Counter Parameters Page 1

Component Name	Specifies a name for the counter. This is the name that you instantiate in your design file to create an instance of the SYNCore counter in your design. Do not use spaces.
Directory	Indicates the directory where the generated files will be stored. Do not use spaces. The following files are created: • filelist.txt – lists files written out by SYNCore • options.txt – lists the options selected in SYNCore • readme.txt – contains a brief description and known issues • syncore_counter.v – Verilog library file required to generate counter model • testbench.v – Verilog testbench file for testing the counter model • instantiation_file.vin – describes how to instantiate the wrapper file • *component*.v – counter model wrapper file generated by SYNCore Note that running the wizard in the same directory overwrites any existing files.
Filename	Specifies the name of the generated file containing the HDL description of the generated counter. Do not use spaces.
Width of Counter	Determines the counter width (the corresponding file parameter is COUNT_WIDTH=n).
Counter Step Value	Determines the counter step value (the corresponding file parameter is STEP=n).
Up Counter	Specifies an up counter (the default) configuration (the corresponding file parameter is MODE=Up).
Down Counter	Specifies an down counter configuration (the corresponding file parameter is MODE=Down).
UpDown Counter	Specifies a dynamic up/down counter configuration (the corresponding file parameter is MODE=Dynamic).

Copyright © 2013 Synopsys, Inc.

SYNCore Counter Parameters Page 2

Enable Load option	Enables the load options
Load Constant Value	Load the constant value specified in the Load Value for constant load option field; (the corresponding file parameter is LOAD=1).
Load Value for constant load option	The constant value to be loaded.
Use the port PortLoadValue to load Value	Loads variable value from PortLoadValue (the corresponding file parameter is LOAD=2).
Synchronous Reset	Specifies a synchronous (the default) reset input (the corresponding file parameter is MODE=0).
Asynchronous Reset	Specifies an asynchronous reset input (the corresponding file parameter is MODE=1).

Configure and Launch VCS Simulator Command

The Configure and Launch VCS Simulator command enables you to launch VCS simulation from within the Synopsys FPGA synthesis tools. Additionally, configuration information, such as libraries and options can be specified on the Run VCS Simulator dialog box before running VCS simulation. You can launch this simulation tool from the synthesis tools on Linux platforms only.

Copyright © 2013 Synopsys, Inc.

For a step-by-step procedure on setting up and launching this tool, see Simulating with the VCS Tool, on page 899 in the *User Guide*.

The Run VCS *SimulationType* Simulation dialog box contains unique pages for specific tasks, such as specifying simulation type, VCS options, and libraries or test bench files. From this dialog box:

- Choose a category, which simplifies the data input for each task.
- A task marked with (✔) means that data has automatically been filled in; however, an (⊗) requires that data must be filled in.
- You are prompted to save, after canceling changes made in the dialog box.

Simulation Type

The following dialog box displays the Simulation Type task.

Copyright © 2013 Synopsys, Inc.

The Run VCS Simulator dialog box contains the following options:

Command	Description
Choose a Category Simulation Type	Select Simulation Type and choose the type of simulation to run: • Pre-synthesis – RTL simulation • Post-synthesis – Post-synthesis netlist simulation • Post-P&R – Post-P&R netlist simulation See *Simulation Type*, on page 275 to view the dialog box.
Choose a Category Top Level Module	Select Top Level Module and specify the top-level VCS module or modules for simulation. You can use any combination of the semi-colon (;), comma (,), or a space to separate multiple top-level modules. See *Top Level Module*, on page 278 to view the dialog box.
Choose a Category VCS Options	Select VCS Options and specify options for each VCS step: • Verilog compiler – VLOGAN command options for compiling and analyzing Verilog, such as the -q option. • VHDL compiler – VHDLAN options for compiling and analyzing VHDL. • Elaboration – VCS command options. The default setting is -debug_all. • Simulation – SIMV command options. The default setting is -gui. The default settings use the FPGA version of VCS and open the VCS GUI for the debugger (DBE) and the waveform viewer. See *VCS Options*, on page 278 to view the dialog box.
Choose a Category Vendor Version	For Xilinx devices, you can select Vendor Version and specify the Xilinx ISE version (10.3 or 11.2) to use. See *Vendor Version*, on page 279 to view the dialog box.
Choose a Category Libraries	Select Libraries and specify library files typically used for Post-synthesis or Post-P&R simulation. These library files are automatically populated in the display window. You can choose to: • Add a library • Edit the selected library • Remove the selected library See *Libraries*, on page 279 and *Changing Library and Test Bench Files*, on page 281 for more information.

Command	Description
Choose a Category Test Bench Files	Select Test Bench Files and specify the test bench files typically used for Post-synthesis or Post-P&R simulation. These test bench files are automatically populated in the display window. You can choose to: • Add a test bench file • Edit the selected test bench file • Remove the selected test bench file See *Test Bench Files*, on page 280 and *Changing Library and Test Bench Files*, on page 281 for more information.
Choose a Category Run Directory	Select Run Directory and specify the results directory to run the VCS simulation. See *Run Directory*, on page 280 to view the dialog box.
Choose a Category Post P&R Netlist	Select Post P&R Netlist and specify the post place-and-route netlist to run the VCS simulation. See *Post P&R Netlist*, on page 281 to view the dialog box.
Run	Runs VCS simulation.
View Script	View the script file with the specified VCS commands and options before generating it. For an example, see *VCS Script File*, on page 283.
Load From	Use this option to load an existing VCS script.
Save As	Generates the VCS script. The tool generates the XML script in the directory specified.
Restore Defaults	Restores all the default VCS settings.

Copyright © 2013 Synopsys, Inc.

Top Level Module

The following dialog box displays the Top Level Module task.

VCS Options

The following dialog box displays the VCS Options task.

Vendor Version

The following dialog box displays the Vendor Versions task.

Libraries

The following dialog box displays the Libraries task.

Copyright © 2013 Synopsys, Inc.

Test Bench Files

The following dialog box displays the Test Bench Files task.

Run Directory

The following dialog box displays the Run Directory task.

Post P&R Netlist

The following dialog box displays the Post P&R Netlist task.

Changing Library and Test Bench Files

You can add Post-synthesis or Post place-and-route library files and test bench files before you launch the VCS simulator. For example, specify options on the following dialog box.

Copyright © 2013 Synopsys, Inc.

You can also edit library files and test bench files before you launch the VCS simulator. For example: specify options on the following dialog box.

VCS Script File

When you select the VCS Script button on the Run VCS Simulator dialog box, you can view the VCS script generated by the synthesis software for this VCS run. You can also save this VCS script to a file by clicking on Save a Copy.

Copyright © 2013 Synopsys, Inc.

Analysis Menu

When you synthesize a design, a default timing report is automatically written to the log file (*projectName*.srr), located in the results directory. This report provides a clock summary, I/O timing summary, and detailed critical path information for the design. However, you can also generate a custom timing report that provides more information than the default report (specific paths or more than five paths) or one that provides timing based on additional analysis constraint files without rerunning synthesis.

Command	Description
Timing Report View	The Timing Report View displays the synthesis timing report summary and lets you compare synthesis timing results with place-and-route (P&R) Static Timing Analysis (STA) results and determine if paths for timing end points, start points, and requested periods match. See *Timing Report View*, on page 285.
Timing Analyst	*Synplify Pro and Synplify Premier.* Displays the Timing Report Generation dialog box to specify parameters for a stand-alone customized report. See *Timing Report Generation Parameters*, on page 291 for information on setting these options, and *Analyzing Timing in Schematic Views*, on page 722 in the *User Guide* for more information. If you click OK in the dialog box, the specified parameters are saved to a file. To run the report, click Generate. The report is created using your specified parameters.
Generate Timing	Generates and displays a report using the timing option parameters specified above. See the following: • Generating Custom Timing Reports with STA, on page 729 for specifics on how to run this report. • *Timing Report Generation Parameters*, on page 291 for information on setting parameters for the report. This includes information on filtering and options for running backannotation data and power consumption reports.
Activity Analysis Generation	Generates the activity analysis report. See *Activity Analysis Generation*, on page 301.
Generate Congestion Analysis	Generates a congestion report for the active implementation. See *Generate Congestion Analysis*, on page 303.

Copyright © 2013 Synopsys, Inc.

Timing Report View

Use the Analysis->Timing Report View command or its associated icon (T) to display the Timing Report View tool. The Timing Report View displays the synthesis timing report summary and the P&R timing and correlation report. This tool lets you compare synthesis timing results with place-and-route (P&R) Static Timing Analysis (STA) results and determine if paths for timing end points, start points, and requested periods match.

For more information, see *Using the Timing Report View*, on page 744.

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

Synthesis Timing Tab

The Synthesis Timing tab allows you to view and manipulate timing results after synthesis. The following table describes the options you can set on the Synthesis Timing tab.

The Synthesis Timing and P&R and Correlation tabs are mutually exclusive. The panel in front is active and uses the selected paths to run the options for the synthesis timing report or the P&R timing correlation report.

Copyright © 2013 Synopsys, Inc.

Field	Description
Synthesis Timing Options	
Logic Synthesis	Specify displaying logical timing results.
Physical Plus Synthesis	*Synplify Premier Only* Specify displaying physical synthesis timing results.
Generate • Synthesis Schematic Button • Synthesis Report Button	• Generate the selected and filtered HDL Analyst view for the path. • Generate the synthesis timing report for the selected path.
Find	Locate clock and instance names in this view.
Expand All	Expand all paths for the specified clock or instance.
Collapse All	Collapse all paths for the specified clock or instance.
Options	Specify the number of P&R paths per end point to include in the synthesis timing report. The default is 1. For details, see *Synthesis Timing Options*, on page 289.
Synthesis Timing Report	
Clock Name / End Instance	Clock name or name of end point instance
End Type	Type of end point instance from the P&R tool that can be specified as • reg • port
Slack	Slack value
Start Instance	Name of start point instance
Start Type	Type of start point instance from the P&R tool that can be specified as • reg • port

Copyright © 2013 Synopsys, Inc.

Field	Description
End Clock	End rising/falling clock
Start Clock	Start rising/falling clock
Clock Skew	Clock skew value
Required Period	Requested period

P&R Timing Correlation Tab

The P&R Timing Correlation tab lets you set options for timing.

The following table describes the options you can set on the P&R Timing and Correlation tab.

The Synthesis Timing and P&R and Correlation tabs are mutually exclusive. The panel in front is active and uses the selected paths to run the options for the synthesis timing report or the P&R timing correlation report.

Field	Description
P&R Timing Options	
P&R Directory	Select the P&R directory for an implementation or other project.
Logic Synthesis	Specify displaying logical timing results.

Copyright © 2013 Synopsys, Inc.

Physical Plus Synthesis	*Synplify Premier Only* Specify displaying physical synthesis timing results.
Correlate • All Paths Button • Selected Path Button	Run the timing correlation report for all paths or only selected path(s).
Generate • Synthesis Schematic Button • Synthesis Report Button • P&R Report Button	• Generate the selected and filtered HDL Analyst view for the path. • Generate the synthesis timing report for the selected path. • Display the P&R report file (`twr`) for the selected path.
Clock Mappings	The Clock Correlation table links clock names in synthesis to clock names in P&R. The table is populated after Clock Rules is run. The synthesis clock is a clock alias name that appears in the synthesis log file (`srr`) or timing report file (`ta`).
Find	Locate clock and instance names in this view.
Expand All	Expand all paths for the specified clock or instance.
Collapse All	Collapse all paths for the specified clock or instance.
Options	Set P&R timing correlation options. For more information, see *P&R Timing and Correlation Options*, on page 290.
P&R Timing Report	
UCF Constraint / End Instance	Paths listed by end point for each UCF Constraint.
End Type	Type of end point instance from the P&R tool that can be specified as: • reg • port
Slack	Slack value

Correlated Status	Status of timing report paths: • Green Check Mark- Paths passed correlation • Red X Mark- Paths failed correlation - Float over the cell for a tool tip describing the error. • Gray - Correlation not run for paths
Start Instance	Name of the start point instance
Start Type	Type of start point instance from the P&R tool that can be specified as: • reg • port
End Clock	End rising/falling clock
Start Clock	Start rising/falling clock
Clock Skew	Clock skew value
Required Period	Requested period

Synthesis Timing Options

The Synthesis Timing Options dialog box lets you specify the number of paths per end point to include in the synthesis timing report.

Copyright © 2013 Synopsys, Inc.

P&R Timing and Correlation Options

The P&R End Point Correlation Options dialog box lets you specify the options to narrow the scope of paths included in the timing correlation report.

Enable any of the options in the following table:

Options	**Description**
Paths per Endpoint	Specify the number of P&R paths per end point to include in the synthesis timing report. The default is 1.
Slack Margin	This option directs the timing comparison routines to compare the reported slack for a P&R path to the same path in synthesis, if found. When a P&R path successfully correlates to a synthesis path, this option further checks that reported slacks are within the given range (in nanoseconds) of each other. When enabled, slacks that are within the given range show up as green and slacks out of range show up as red. Use the tool tip to fly over red slacks with the cursor to see a report of the slack found in synthesis.
For the following *Xilinx only* options, if none of them are selected, all options are active. Select them individually to narrow the scope of paths (based on UCF constraint) included in the End Point/Start Point Table.	
Clock	Include TIMESPEC...PERIOD constraints in the table.
Offset	Include OFFSET constraints in the table.

Default STA	Use the default timing report from P&R (*top*.twr) to populate the End Point/Start Point Table. Use the option early on to verify the P&R critical paths in a design. This is the default.
TIG	Include TIMESPEC...TIG constraints in the table.
Delay	Include TIMESPEC...*delay* constraints in the table.

Timing Report Generation Parameters

Synplify Pro and Synplify Premier.

You can use the Analysis->Timing Analyst command to specify parameters for a stand-alone timing report. See *Timing Reports*, on page 615 for information on the file contents.

Copyright © 2013 Synopsys, Inc.

The following table provides brief descriptions of the parameters for running a stand-alone timing report.

Timing Report Option	Description
From or To	Specifies the starting (From) or ending (To) point of the path for one or more objects. It must be a timing start point (From) or end (To) point for each object. Use this option in combination with the others in the Filters section of the dialog box. See *Combining Path Filters for the Timing Analyzer*, on page 295 for examples of using filters.
Through	Reports all paths through the specified point or list of objects. See for more information on using this filter. Use this option in combination with the others in the Filters section of the dialog box. See the following for additional information: • Timing Analyzer Through Points, on page 293 • Combining Path Filters for the Timing Analyzer, on page 295.
Generate Asynchronous Clock Report	Generates a report for paths that cross between clock groups. Generally paths in different clock groups are automatically handled as false paths. This option provides a file that contains information on each of the paths and can be viewed in a spreadsheet. This file is in the results directory (*projectName*_async_clk.rpt.csv). For details on the report, see *Asynchronous Clock Report*, on page 622.
Limit Number of Critical Start/End points	Specifies the maximum number of start/end paths to display for critical paths in the design. The default is 5. Use this option in combination with the others in the Filters section of the dialog box.
Limit Number of Paths to	Specifies the maximum number of paths to report. The default is 5. If you leave this field blank, all paths in the design are reported. Use this option in combination with the others in the Filters section of the dialog box.
Enable Slack Margin (ns)	Limits the report to paths within the specified distance of the critical path. Use this option in combination with the others in the Filters section of the dialog box.
Open Report	When enabled, clicking the Generate button opens the Text Editor on the generated custom timing report specified in the timing report file (ta).

Copyright © 2013 Synopsys, Inc.

Timing Report Option	Description
Open Schematic	When enabled, clicking the Generate button opens a Technology view showing the netlist specified in the timing report netlist file (srm).
Output Files	Displays the name of the generated report: • Async Clock Report File contains the spreadsheet data for the asynchronous clock report. This file is not automatically opened when report generation is complete. You can locate this file in the results directory. Default name is *projectName*_async_clk.rpt.csv (name cannot be changed). • Timing Analyst Results File is the standard timing report file, located in the Implementation Results directory. The file is also listed in the Project view. Default filename is *projectName*.ta. • SRM File updates the Technology view so that you can display the results of the timing updates in the HDL and Physical Analyst tools. The file is also listed in the Project view. For more details on any of these reports, see *Timing Reports*, on page 615.
Constraint Files	Enables analysis design constraint files (adc) to be used for stand-alone timing analysis only. See *Input Files*, on page 594 for information on this file.
Generate	Clicking this button generates the specified timing report file and timing view netlist file (srm) if requested, saves the current dialog box entries for subsequent use, then closes the dialog box.

Timing Analyzer Through Points

You can specify through points for nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). You can specify the through points in two ways:

Copyright © 2013 Synopsys, Inc.

OR list	Enter the points as a space-separated list. The points are treated as an OR list and paths are reported if they crosses any of the points in the list. For example, when you type the following, the tool reports paths that pass through points b or c: `{n:b n:c}` See *Filtering Points: OR List of Through Points*, on page 294.

AND list	Enter the points in a product of sums (POS) format. The tool treats them as an AND list, and only reports the path if it passes through all the points in the list. The POS format for the timing report is the same as for timing constraints. The POS format is as follows:
	`{n:b n:c},{n:d n:e}`
	This constraint translates as follows:
	`b AND d`
	`OR b AND e`
	`OR c AND d`
	`OR c AND e`
	See *Filtering Points: AND List of Through Points*, on page 295.

See *Defining From/To/Through Points for Timing Exceptions*, on page 146 in the *User Guide* for more information about specifying through points.

Filtering Points: OR List of Through Points

This example reports the five worst paths through port bdpol or net aluout. You can enter the through points as a space-separated list (enclosing the list in braces is optional.)

Filtering Points: AND List of Through Points

This example reports the five worst paths passing through port bdpol and net aluout. Enclose each list in braces { } and separate the lists with a comma.

Combining Path Filters for the Timing Analyzer

This section describes how to use a combination of path filters to specify what you need and how to specify start and end points for path filtering.

Number and Slack Path Filters

The Limit Number of Paths To option specifies the maximum number of paths to report and the Enable Slack Margin option limits the report to output only paths that have a slack value that is within the specified value. When you use these

Copyright © 2013 Synopsys, Inc.

two options together, the tighter constraint applies, so that the actual number of paths reported is the minimum of the option with the smallest value. For example, if you set the number of paths to report to 10 and the slack margin for 1 ns, if the design has only five paths within 1 ns of critical, then only five paths are reported (not the 10 worst paths). But if, for example, the design has 15 paths within a 1 ns of critical, only the first 10 are reported.

From/To/Through Filters

You can specify the from/to points for a path. You can also specify just a from point or just a to point. The from and to points are one or more hierarchical names that specify a port, register, pin on a register, or clock as object (clock alias). Ports and instances can have the same names, so prefix the name with p: for top-level port, i: for instance, or t: for hierarchical port or instance pin. However, the c: prefix for clocks is required for paths to be reported.

The timing analyst searches for the from/to objects in the following order: clock, port, bit port, cell (instance), net, and pin. Always use the prefix qualifier to ensure that all expected paths are reported. Remember that the timing analyst stops at the first occurrence of an object match. For buses, all possible paths from the specified start to end points are considered.

You can specify through points for nets, cell pins, or hierarchical ports.

You can simply type in from/to or through points. You can also cut-and-paste or drag-and-drop valid objects from the RTL or Technology views into the appropriate fields on the Timing Report Generation dialog box. Timing analysis requires that constraints use the Tech View name space. Therefore, it is recommended that you cut-and-paste or drag-and-drop objects from the Technology view rather than the RTL view.

Copyright © 2013 Synopsys, Inc.

This following examples show how to specify start, end, or through point combinations for path filtering.

Filtering Points: Single Register to Single Register

Filtering Points: Clock Object to Single Register

Filtering Points: Single Bit of a Bus to Single Register

Copyright © 2013 Synopsys, Inc.

Filtering Points: Single Bit of a Bus to Single Bit of a Bus

Filtering Points: Multiple Bits of a Bus to Multiple Bits of a Bus

Filtering Points: With Hierarchy

This example reports the five worst paths for the net foo:

Copyright © 2013 Synopsys, Inc.

Filtering Points: Through Point for a Net

Filtering Points: Through Point for a Hierarchical Port

This example reports the five worst paths for the hierarchical port bdpol:

Examples Using Wildcards

You can use the question mark (?) or asterisk (*) wildcard characters for object searching and name substitution. These characters work the same way in the synthesis tool environment as in the Linux environment.

Copyright © 2013 Synopsys, Inc.

The ? Wildcard

The ? matches single characters. If a design has buses op_a[7:0], op_b[7:0], and op_c[7:0], and you want to filter the paths starting at each of these buses, specify the start points as op_?[7:0]. See *Example: ? Wildcard in the Name*, on page 300 for another example.

The * Wildcard

The * matches a string of characters. In a design with buses op_a2[7:0], op_b2[7:0], and op_c2[7:0], where you want to filter the paths starting at each of these objects, specify the start points as op_*[*]. The report shows all paths beginning at each of these buses and for all of the bits of each bus. See *Example: * Wildcard in the Name (With Hierarchy)*, on page 301 and *Example: * Wildcard in the Bus Index*, on page 301 for more examples.

Example: ? Wildcard in the Name

The ? is not supported in bus indices.

Example: * Wildcard in the Name (With Hierarchy)

This example reports the five worst paths, starting at block rxu_fifo and ending at block rxu_channel within module nac_core. Each register in the design has the characters reg in the name.

Example: * Wildcard in the Bus Index

This example reports the five worst paths, starting at op_b, and ending at d_out, taking into account all bits on these buses.

Activity Analysis Generation

Synplify Premier only; Xilinx only. The Activity Analysis Generation command captures the switching activity data in a report file to use for analyzing and optimizing design power. After running synthesis, you input the netlist and analysis design constraints into the Synplify Premier tool and rerun synthesis to generate the report which is written to a SAIF file. This 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 routes.

Copyright © 2013 Synopsys, Inc.

For details on complete support, see Chapter 20, *Activity Analysis for Power Optimization*.

The Activity Analysis Generation dialog box contains the following options:

Command	Description
Output Files SAIF File	The Synopsys switching activity interchange format file 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.
Constraint Files	You can override the defaults for net 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.

Generate Congestion Analysis

Synplify Premier only; Xilinx only

The Generate Congestion Analysis command lets you generate a congestion report for the active implementation. 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 improved routability for the design.

For more information, see *Congestion Analysis After Logic Synthesis*, on page 538.

The Generate Congestion Analysis Report dialog box contains the following options:

Command	Description
Implementation	Displays the current implementation.
P&R NCD Input	Specifies the mapped NCD file from Xilinx place and route. You can use the pre-populated congestion report output file: *designName*_congestion_est.rpt
Congestion Report Output	Specifies the output congestion report destination location.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu

In the Project View, the HDL Analyst menu contains commands that provide project analysis in the following views:

- RTL View
- Technology View
- Physical Analyst (see *Physical Analyst*, on page 579).

This section describes the HDL Analyst menu commands for the RTL and Technology views. Commands may be disabled (grayed out), depending on the current context. Generally, the commands enabled in any context reflect those available in the corresponding popup menus. The descriptions in the table indicate when commands are context-dependent. For explanations about the terms used in the table, such as filtered and unfiltered, transparent and opaque, see *Filtered and Unfiltered Schematic Views*, on page 416 and *Transparent and Opaque Display of Hierarchical Instances*, on page 421. For procedures on using the HDL Analyst tool, see *Analyzing With the HDL Analyst Tool*, on page 589 of the *User Guide.*

For ease of use, the commands have been divided into sections that correspond to the divisions in the HDL Analyst menu.

- HDL Analyst Menu->RTL and Technology View Submenus, on page 305
- HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 306
- HDL Analyst Menu: Filtering and Flattening Commands, on page 308
- HDL Analyst Menu: Timing Commands, on page 311
- HDL Analyst Menu: Analysis Commands, on page 311
- HDL Analyst Menu: Selection Commands, on page 315
- HDL Analyst Menu: FSM Commands, on page 315
- HDL Analyst Menu: VCD Commands, on page 316

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu->RTL and Technology View Submenus

This table describes the commands that appear on the HDL Analyst->RTL and HDL Analyst->Technology submenus when the RTL or Technology View is active. For procedures on using these commands, see *Analyzing With the HDL Analyst Tool*, on page 589 of the *User Guide*.

HDL Analyst Command	Description
RTL->Hierarchical View	Opens a new, hierarchical RTL view. The schematic is unfiltered.
RTL->Flattened View	Opens a new RTL view of your entire design, with a flattened, unfiltered schematic at the level of generic logic cells. See *Usage Notes for Flattening*, on page 309 for some usage tips.
RTL->Design Plan View	Displays an RTL schematic of the device and its design-planned regions in the Synplify Premier tool with Design Planner. Same as double-clicking the partitioned RTL view (srp) file. See *Working in the Schematic Views*, on page 546 of the *User Guide*.
Technology->Hierarchical View	Opens a new, hierarchical Technology view. The schematic is unfiltered.
Technology->Flattened View	Creates a new Technology view of your entire design, with a flattened, unfiltered schematic at the level of technology cells. See *Usage Notes for Flattening*, on page 309 for tips about flattening.
Technology->Flattened to Gates View	Creates a new Technology view of your entire design, with a flattened, unfiltered schematic at the level of Boolean logic gates. See *Usage Notes for Flattening*, on page 309 for tips about flattening.
Technology->Hierarchical Critical Path	Creates a new Technology view of your design, with a hierarchical, *filtered* schematic showing only the instances and paths whose slack times are within the slack margin you specified in the Slack Margin dialog. This command automatically enables HDL Analyst->Show Timing Information.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command	Description
Technology->Flattened Critical Path	Creates a new Technology view of your design, with a flattened, *filtered* schematic showing only the instances and paths whose slack times are within the slack margin you specified in the Slack Margin dialog. This command automatically enables HDL Analyst->Show Timing Information. See *Usage Notes for Flattening*, on page 309 for tips about flattening.

HDL Analyst Menu: Hierarchical and Current Level Submenus

This table describes the commands on the HDL Analyst->Hierarchical and HDL Analyst->Current Level submenus. For procedures on using these commands, see *Analyzing With the HDL Analyst Tool*, on page 589 of the *User Guide.*

HDL Analyst Command	Description
Hierarchical->Expand	Expands paths from selected pins and/or ports up to the nearest objects on any hierarchical level, according to pin/port directions. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level. Successive Expand commands expand the paths further, based on the new current selection.
Hierarchical->Expand to Register/Port	Expands paths from selected pins and/or ports, in the port/pin direction, up to the next register, port, or black box. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Expand Paths	Shows all logic, on any hierarchical level, between two or more selected instances, pins, or ports. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Expand Inwards	Expands within the hierarchy of an instance, from the lower-level ports that correspond to the selected pins, to the nearest objects and no further. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command	Description
Hierarchical->Goto Net Driver	Displays the unfiltered schematic sheet that contains the net driver for the selected net. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Select Net Driver	Selects the driver for the selected net. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Select Net Instances	Selects instances connected to the selected net. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Current Level->Expand	Expands paths from selected pins and/or ports up to the nearest objects on the current level, according to pin/port directions. The result is a *filtered* schematic. Limited to all sheets on the current schematic level. This command is only available if a HDL Analyst view is open. Successive Expand commands expand the paths further, based on the new current selection.
Current Level->Expand to Register/Port	Expands paths from selected pins and/or ports, according to the pin/port direction, up to the next register, ports, or black box on the current level. The result is a *filtered* schematic. Limited to all sheets on the current schematic level.
Current Level->Expand Paths	Shows all logic on the current level between two or more selected instances, pins, or ports. The result is a *filtered* schematic. Limited to the current schematic level (all sheets).
Current Level->Goto Net Driver	Displays the unfiltered schematic sheet that contains the net driver for the selected net. Limited to all sheets on the current schematic level.
Current Level->Select Net Driver	Selects the driver for the selected net. The result is a *filtered* schematic. Limited to all sheets on the current schematic level.
Current Level->Select Net Instances	Selects instances on the current level that are connected to the selected net. The result is a *filtered* schematic. Limited to all sheets on the current schematic level.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu: Filtering and Flattening Commands

This table describes the filtering and flattening commands on the HDL Analyst menu. For procedures on filtering and flattening, see *Analyzing With the HDL Analyst Tool*, on page 589 of the *User Guide*.

HDL Analyst Command	Description
Filter Schematic	Filters your entire design to show only the selected objects. The result is a *filtered* schematic. For more information about using this command, see *Filtering Schematics*, on page 593 of the *User Guide*. This command is only available with an open HDL Analyst view.
Flatten Current Schematic (Unfiltered Schematic)	In an unfiltered schematic, the command flattens the current schematic, at the current level and all levels below. In an RTL view, the result is at the generic logic level. In a Technology view, the result is at the technology-cell level. See the next table entry for information about flattening a filtered schematic. This command does not do the following: • Flatten your entire design (unless the current level is the top level) • Open a new view window • Take into account the number of Dissolve Levels defined in the Schematic Options dialog box See *Usage Notes for Flattening*, on page 309 for tips.

HDL Analyst Command	Description
Flatten Current Schematic (Filtered Schematic)	In a filtered schematic, flattening is a two-step process: • Only unhidden transparent instances (including nested ones) are flattened in place, in the context of the entire design. Opaque and hidden hierarchical instances remain hierarchical. The effect of this command is that all hollow boxes with pale yellow borders are removed from the schematic, leaving only what was displayed inside them. • The original filtering is restored. In an RTL view, the result is at the generic logic level. In a Technology view, the result is at the technology-cell level. This command does not do the following: • Flatten everything inside a transparent instance. It only flattens transparent instances and any nested transparent instances they contain. • Open a new view window. • Take into account the number of Dissolve Levels defined in the Schematic Options dialog box. See *Usage Notes for Flattening*, on page 309 for usage tips.
Unflatten Current Schematic	Undoes any flattening operations and returns you to the original schematic, as it was before flattening and any filtering. This command is available only if you have explicitly flattened a hierarchical schematic using HDL Analyst->Flatten Current Schematic, for example. It is not available for flattened schematics created directly with the RTL and Technology submenus of the HDL Analyst menu.

Usage Notes for Flattening

It is usually more memory-efficient to flatten only parts of your design, as needed. The following are a few tips for flattening designs with different commands. For detailed procedures, see *Flattening Schematic Hierarchy*, on page 600 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

RTL/Technology->Flattened View Commands

- Use Flatten Current Schematic to flatten only the current hierarchical level and below.
- Flatten selected hierarchical instances with Dissolve Instances (followed by Flatten Current Schematic, if the schematic is filtered).
- To make hierarchical instances transparent without flattening them, use Dissolve Instances in a filtered schematic. This shows their details nested inside the instances.

Flatten Current Schematic Command (Unfiltered View)

- Flatten selected hierarchical instances with Dissolve Instances.
- To see the lower-level logic inside a hierarchical instance, push into it instead of flattening.
- Selectively flatten your design by hiding the instances you do not need, flattening, and then unhiding the instances.
- Flattening erases the history of displayed sheets for the current view. You can no longer use View->Back. You can, however, use UnFlatten Schematic to get an unflattened view of the design.

Flatten Current Schematic Command (Filtered View)

- Flatten selected hierarchical instances with Dissolve Instances, followed by Flatten Current Schematic.
- Selectively flatten your design by hiding the instances you do not need, flattening, and then unhiding the instances.
- Flattening erases the history of displayed sheets for the current view. You can no longer use View->Back. You can do the following:
- Use View->Back for a view of the transparent instance flattened in the context of the entire design. This is the view generated after step 1 of the two-step flattening process described above. Use UnFlatten Schematic to get an unflattened view of the design.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu: Timing Commands

This table describes the timing commands on the HDL Analyst menu. For procedures on using the timing commands, see *Analyzing With the HDL Analyst Tool*, on page 589 of the *User Guide*.

HDL Analyst Command	Description
Set Slack Margin	Displays the Slack Margin dialog box, where you set the slack margin. HDL Analyst->Show Critical Path displays only those instances whose slack times are worse than the limit set here. Available only in a Technology view.
Show Critical Path	Filters your entire design to show only the instances and paths whose slack times exceed the slack margin set with Set Slack Margin, above. The result is flat if the entire design was already flat. This command also enables Show Timing Information (see below). Available only in a Technology view.
Show Timing Information	When enabled, Technology view schematics are annotated with timing numbers above each instance. The first number is the cumulative path delay; the second is the slack time of the worst path through the instance. Negative slack indicates that timing has not met requirements. Available only in a Technology view. For more information, see *Viewing Timing Information*, on page 722 on the *User Guide*.

HDL Analyst Menu: Analysis Commands

This table describes the analysis commands on the HDL Analyst menu. For procedures on using the analysis commands, see *Analyzing With the HDL Analyst Tool*, on page 589 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command	Description
Isolate Paths	Filters the current schematic to display only paths associated with all the pins of the selected instances. The paths follow the pin direction (from output to input pins), up to the next register, black box, port, or hierarchical instance. If the selected objects include ports and/or pins on unselected instances, the result also includes paths associated with those selected objects. The range of the operation is all sheets of a filtered schematic or just the current sheet of an unfiltered schematic. The result is always a filtered schematic. In contrast to the Expand operations, which add to what you see, Isolate Paths can only remove objects from the display. While Isolate Paths is similar to Expand to Register/Port, Isolate Paths reduces the display while Expand to Register/Port augments it.
Show Context	Shows the original, unfiltered schematic sheet that contains the selected instance. Available only in a filtered schematic.
Hide Instances	Hides the logic inside the selected hierarchical (non-primitive) instances. This affects only the active HDL Analyst view; the instances are not hidden in other HDL Analyst views. The logic inside hidden instances is not loaded (saving dynamic memory), and it is unrecognized by searching, dissolving, flattening, expansion, and push/pop operations. (Crossprobing does recognize logic inside hidden instances, however.) See *Usage Notes for Hiding Instances*, on page 314 for tips.
Unhide Instances	Undoes the effect of Hide Instances: the selected hidden hierarchical instances become visible (susceptible to loading, searching, dissolving, flattening, expansion, and push/pop operations). This affects only the current HDL Analyst view; the instances are not hidden in other HDL Analyst views.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command	Description
Show All Hier Pins	Shows all pins on the selected transparent, non-primitive instances. Available only in a filtered schematic. Normally, transparent instance pins that are connected to logic that has been filtered out are not displayed. This command lets you display these pins that connected to logic that has been filtered out. Pins on primitives are always shown.
Dissolve Instances	Shows the lower-level details of the selected non-hidden hierarchical instances. The number of levels dissolved is determined by the Dissolve Levels value in the HDL Analyst Options dialog box (*HDL Analyst Options Command*, on page 345). For usage tips, see *Usage Notes for Dissolving Instances*, on page 314.
Dissolve to Gates	Dissolves the selected instances by flattening them to the gate level. This command displays the lower-level hierarchy of selected instances, but it dissolves technology primitives as well as hierarchical instances. Technology primitives are dissolved to generic synthesis symbols. The command is only available in the Technology view. The number of levels dissolved is determined by the Dissolve Levels value in the HDL Analyst Options dialog box (*HDL Analyst Options Command*, on page 345). Dissolving an instance one level redraws the current sheet, replacing the hierarchical dissolved instance with the logic you would see if you pushed into it using Push/pop mode. Unselected objects or selected hidden instances are not dissolved. The effect of the command varies: • In an unfiltered schematic, this command *flattens* the selected instances. This means the history of displayed sheets is removed. The resulting schematic is unfiltered. • In a filtered schematic, this command makes the selected instances *transparent*, displaying their internal, lower-level logic inside hollow boxes. History is retained. You can use Flatten Schematic to flatten the transparent instances, if necessary. The resulting schematic if filtered.

Copyright © 2013 Synopsys, Inc.

Usage Notes for Hiding Instances

The following are a few tips for hiding instances. For detailed procedures, see *Flattening Schematic Hierarchy*, on page 600 of the *User Guide*.

- Hiding hierarchical instances soon after startup can often save memory. After the interior of an instance has been examined (by searching or displaying), it is too late for this savings.
- You can save memory by creating small, temporary working files: File->Save As .srs or .srm files does not save the hidden logic (hidden instances are saved as black boxes). Restarting the synthesis tool and loading such a saved file can often result in significant memory savings.
- You can selectively flatten instances by temporarily hiding all the others, flattening, then unhiding.
- You can limit the range of Edit->Find (see *Find Command (HDL Analyst)*, on page 152) to prevent it looking inside given instances, by temporarily hiding them.

Usage Notes for Dissolving Instances

Dissolving an instance one level redraws the current sheet, replacing the hierarchical dissolved instance with the logic you would see if you pushed into it using Push/pop mode. Unselected objects or selected hidden instances are not dissolved. For additional information about dissolving instances, see *Flattening Schematic Hierarchy*, on page 600 of the *User Guide*.

The type (filtered or unfiltered) of the resulting schematic is unchanged from that of the current schematic. However, the effect of the command is different in filtered and unfiltered schematics:

- In an unfiltered schematic, this command flattens the selected instances. This means the history of displayed sheets is removed.
- In a filtered schematic, this command makes the selected instances transparent, displaying their internal, lower-level logic inside hollow boxes. History is retained. You can use Flatten Schematic to flatten the transparent instances, if necessary. This command is only available if an HDL Analyst view is open.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu: Selection Commands

This table describes the selection commands on the HDL Analyst menu.

HDL Analyst Command	Description
Select All Schematic ->Instances ->Ports	Selects all Instances or Ports, respectively, on all sheets of the current schematic. All other objects are unselected. This does not select objects on other schematics.
Select All Sheet ->Instances ->Ports	Selects all Instances or Ports, respectively, on the current schematic sheet. All other objects are unselected.
Unselect All	Unselects all objects in all HDL Analyst views.

HDL Analyst Menu: FSM Commands

This table describes the FSM commands on the HDL Analyst menu.

HDL Analyst Command	Description
View FSM	*Synplify Pro and Synplify Premier only.* Displays the selected finite state machine in the FSM Viewer. Available only in an RTL view.
View FSM Info File	Displays information about the selected finite state machine module, including the number of states, the number of inputs, and a table of the states and transitions. Available only in an RTL view.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu: VCD Commands

This table describes the VCD/Identify-HDL Analyst commands for the VCD simulation or Identify Analysis tool on the HDL Analyst menu. Select HDL Analyst->VCD from the menu.

For more information, see *Using the VCD/Identify-Analyst Integration Tool*, on page 904.

HDL Analyst Command: VCD Commands	Description
VCD Panel	Toggles the VCD Control Panel to be displayed in the HDL Analyst views. See the *Simulation VCD Control Panel*, on page 317 and *Identify VCD Control Panel*, on page 320.
Annotate Values	Enable this option to display the net values in the HDL Analyst view.
Load VCD File	Loads the simulation or Identify file (vcd). The Control Panel changes to a gradient blue color and indicates an active session when the VCD file is loaded. See the *Load Simulation VCD File*, on page 317 and *Load Identify VCD File*, on page 321.
Reload VCD File	Reloads the simulation or Identify file (vcd). The Control Panel changes to a gradient blue color and indicates an active session when the VCD file is loaded.
Unload the VCD File	Unloads the simulation or Identify file (vcd) to free up memory used by the simulation data without having to close and re-open the Analyst view.
Watch Sheet	Watch all nets on the current sheet.
Un-Watch Sheet	Remove watching all nets on the current sheet.
Automatically Watch Sheets	While navigating to a sheet, all nets on the sheet are added to the watch list.
Next Transition	You can view the next transition for an object being watched in the Waveform view.
Previous Transition	You can view the previous transition for an object being watched in the Waveform view.

Copyright © 2013 Synopsys, Inc.

DVE	You can perform the following tasks: • Launch DVE • Configure DVE • Import Watched Signals from DVE • Export Watched Signals to DVE See *DVE*, on page 322.
VCD Properties	Displays properties for the Waveform viewer global data and parameters. See *VCD Properties*, on page 328.

Simulation VCD Control Panel

The Control Panel lets you use the VCD/Identify-Analyst feature, which enhances the interaction between the VCS Simulator and the HDL Analyst tools by providing a way to watch signals annotated in the HDL Analyst from the Control Panel watch list. For more information, see *Using the VCD/Identify-Analyst Integration Tool*, on page 904.

Load Simulation VCD File

To load the simulation file, click the Open a VCD File icon ( ) or select HDL-Analyst->VCD->Load VCD File from the Project menu.

Copyright © 2013 Synopsys, Inc.

This table describes the Load Simulation VCD File dialog box commands.

HDL Analyst Command: VCD Commands	Description
VCD File	Specify the simulation file (vcd) to load. You can enable the Validate VCD File with Netlist option. If you do so, you can use the Show Mismatches button to display any mismatches.
Top Level Module and Paths	Specify the path through the test bench to the root module displayed by the Analyst. Use the drop-down menu to select the top level module of the VCD file. When Limit depth is enabled, the hierarchy level for the Top Level Scope is set to 3 levels of hierarchy.
Locate Top Level Scope	Specify the top-level scope path through the test bench to the top-level module corresponding to the HDL Analyst design. See *Select Top Level Module Path Dialog Box*, on page 319.

Prune Scope Tree	Prune the scope tree for the root scope paths to be included in the hierarchy being loaded. See *Prune Scope Hierarchy Dialog Box*, on page 319.
Time	You can set the time to Full Range or Limited Range determined from the simulation file to be loaded and choose a desired unit of measurement.
Load button	Click the Load button to load the simulation VCD file.

Select Top Level Module Path Dialog Box

Select the path to the top-level module of the design.

Prune Scope Hierarchy Dialog Box

Prune the scope tree for the root scope paths to be included in the hierarchy being loaded. 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.

Copyright © 2013 Synopsys, Inc.

Identify VCD Control Panel

The Control Panel lets you use the VCD/Identify-Analyst feature, which enhances the interaction between the Identify VCD and the HDL Analyst tools by providing a way to watch Identify instrumented signals annotated in the HDL Analyst from the Control Panel watch list. For more information, see *Using the VCD/Identify-Analyst Integration Tool*, on page 904.

Load Identify VCD File

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.

This table describes the Load Simulation VCD File dialog box commands.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command: VCD Commands	Description
VCD File	Specify the Identify file (vcd) to load. You can enable the Validate VCD File with Netlist option. If you do so, you can use the Show Mismatches button to display any mismatches.
Time	You can set the time to Full Range or Limited Range determined from the Identify file to be loaded and choose a desired unit of measurement.
Load button	Click the Load button to load the Identify VCD file.

DVE

DVE (Discovery Virtual Environment) is an interactive waveform viewer that you can launch and configure within the VCD-Analyst Integration tool. Select HDL Analyst->VCD->DVE->Launch DVE from the Project menu. You can:

- Configure DVE. See *Configure DVE Launch Dialog Box*, on page 323.
- Import watched signals from DVE to the VCD-Analyst Integration tool. See the *Import Signals Dialog Box*, on page 325.
- Export watched signals from the VCD-Analyst Integration tool to DVE. See the *Export Signals Dialog Box*, on page 327.

Note: You must have a VCS license and run the DVE tool on a Linux platform.

Launching DVE

To launch DVE, select HDL Analyst->VCD->DVE->Launch DVE from the Project menu.

This is the same as using the ( ) icon.

Configure DVE Launch Dialog Box

To configure DVE options, select HDL Analyst->VCD->DVE->Configure DVE from the Project menu.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command: Simulation Commands	Description
Data Base	Select the following: • None or Analyst Sim File—no signals or signals from the HDL Analyst simulation file are selected. • Other VCD or VPD—specify the path to another simulation file. • Session—you can re-load a saved DVE session Tcl file.
Setup	Select the following: • Transfer Signals Group—transfer specified simulation file signals to a user-specified group. • Open a new wave window (when not opening session)—if enabled, opens a new waveform window.

Copyright © 2013 Synopsys, Inc.

Arguments	Specify any arguments to pass on to DVE.
VCS_HOME	Read-only field that specifies the VCS_HOME/bin installation directory.

Import Signals Dialog Box

Import watched signals from DVE to the VCD-Analyst Integration tool. Select HDL Analyst->VCD->DVE->Import Watched Signals from DVE on the Project menu.

Copyright © 2013 Synopsys, Inc.

This table describes the Import Signals dialog box commands.

HDL Analyst Command	Description
Session File	Specify the DVE session file path containing the source input signals to be imported.

Group	You can search for a specific DVE signal group.
Sync watched signals	Synchronize signals to be the same signals being watched in the Simulation Panel.
Get button	Click the Get button to extract the signals to watch from the import source file.

Export Signals Dialog Box

Export watched signals from the VCD-Analyst Integration tool to DVE. Select HDL Analyst->VCD->DVE->Export Watched Signals to DVE from the Project menu.

This table describes the Export Signals dialog box commands.

Copyright © 2013 Synopsys, Inc.

HDL Analyst Command	Description
Export path	Specify the DVE script file to which exported signals are written, then source the script in a DVE session.
DVE Signal Group	Specify a specific signal group to export. To synchronize the signal group to match the signals to be watched in the Simulation Panel, enable Sync Group.
Export All Watched Nets	When Export All Watched Nets is enabled, this option extracts all the signals to watch from the import source file. When you disable this option, then select: • All - all signals are checked. • None - all signals are unchecked. Then, you can select specific nets to export.

VCD Properties

Displays properties for the Waveform viewer global data and parameters. Select HDL-Analyst->VCD->VCD Properties from the Project menu.

To display signal properties for a watched net in the Control Panel, do the following:

- Be sure at least one net is being watched.
- Right-click on the net and select Properties from the popup menu.

Tools Menu

For more information about the Tools menu, see *Design Planner Tools Menu, on page 552*.

Copyright © 2013 Synopsys, Inc.

Options Menu

Use the Options menu to configure the VHDL and Verilog compilers, customize toolbars, and set options for the Project view, Text Editor, and HDL Analyst schematics. When using certain technologies, additional menu commands let you run technology-vendor software from this menu.

The following table describes the Options menu commands.

Command	Description
Basic Options Menu Commands for all Views	
Configure VHDL Compiler	Opens the Implementation Options dialog box where you can set the top-level entity and the encoding method for enumerated types (Synplify Pro and Synplify Premier tools only). State-machine encoding is automatically determined by the FSM compiler or FSM explorer, or you can specify it explicitly using the syn_encoding attribute. See *Implementation Options Command*, on page 194.
Configure Verilog Compiler	Opens the Implementation Options dialog box where you can specify the top-level module and the include search path. See *Implementation Options Command*, on page 194.
Configure Compile Point Process	Lets you specify the maximum number of parallel synthesis jobs that can be run and how errors in compile points are treated. See *Configure Compile Point Process Command*, on page 332.
Toolbars	Lets you customize the toolbars.
Project View Options	Sets options for organizing files in the Project view. See *Project View Options Command*, on page 335.
Editor Options	Sets your Text Editor syntax coloring, font, and tabs. See *Editor Options Command*, on page 340.
P&R Environment Options	Displays the environmental variable options set for the place-and-route tool. See *Place and Route Environment Options Command*, on page 343.

Copyright © 2013 Synopsys, Inc.

Command	Description
Configure 3rd Party Tool Options	Lets you invoke third-party tools, for example to modify the files generated or debug problems from the System Designer or EDK tools within the FPGA synthesis products. See *Configure 3rd Party Tools Options Command*, on page 344.
HDL Analyst Options	Sets display preferences for HDL Analyst schematics (RTL and Technology views). See *HDL Analyst Options Command*, on page 345.
Configure External Programs	Lets you set browser and Acrobat Reader options on Linux platforms. See *Configure External Programs Command*, on page 352.
Options Menu Commands Specifically for the Project View	
Configure Identify Launch	If Identify software is not properly installed, you might run into problems when you try to launch it from the synthesis tools. Use the Configure Identify Launch dialog box to help you resolve these issues. For guidelines to follow, see *Handling Problems with Launching Identify*, on page 884 in the *User Guide*.
Quartus II (Altera only) ->Launch Quartus ->Run Background Compile	These commands are only for Altera designs. They determine whether you run placement and routing interactively (Launch Quartus) or in batch mode (Run Background Compile).
Xilinx (Xilinx only) ->Start Design Manager ->Start Floorplanner ->Start ISE Project Navigator	These commands are only available for Xilinx designs. They let you start Design Manager, the Xilinx Floorplanner, or ISE Project Navigator from the synthesis tool.
Options Menu Commands Specifically for the Physical Analyst View	
Physical Analyst Control Panel	For details, see *Options Menu: Physical Analyst Commands*, on page 579.
Physical Analyst Color Schemes	For details, see *Options Menu: Physical Analyst Commands*, on page 579.

Copyright © 2013 Synopsys, Inc.

Configure Compile Point Process Command

Use the Configure Compile Point Process command to let you 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 compile point synthesis flow.

This feature is supported on Windows and Linux for certain technologies only. This command is grayed out for technologies that are not supported.

Field/Option	Description
Maximum Number of Parallel Synthesis Jobs	Sets the maximum number of synthesis jobs that can run in parallel. It displays the current value from the `ini` file, and allows you to reset it. Use this option for multiprocessing by running compile point jobs in parallel. Set a value based on the number of available licenses. Note that one license is used for each job. See *License Utilization for Multiprocessing*, on page 334. When you set this option, it resets the MaxParallelJobs value in the `.ini` file. See *Maximum Parallel Jobs*, on page 334 for other ways to specify this value.
Continue on Error	*Synplify Pro and Synplify Premier tools* Allows the software to continue on an error in a compile point and synthesize the rest of the design, even when there might be problems with a portion of the design. The Continue on Error mode automatically enables the MultiProcessing option to run with compile points using a single license; this is the default. For additional runtime improvements, you can specify multiple synthesis jobs that run in parallel. See Chapter 23, *Using Multiprocessing* for details. For more information about Continue on Error mode during compile-point synthesis, see *Continue on Error Mode*, on page 1106 in the *User Guide*. The Continue on Error mode also allows design compilation in the Synplify Premier tools to continue for certain types of compiler errors. For more information about Continue on Error mode during compilation, see *Continue on Error*, on page 233 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Maximum Parallel Jobs

There are three ways to specify the maximum number of parallel jobs:

`ini` File	Set this variable in the MaxParallelJobs variable in the product `ini` file:
	`[JobSetting]`
	`MaxParallelJobs=<n>`
	This value is used by the UI as well as in batch mode, and is effective until you specify a new value. You can change it with the Options->Configure Compile Point Process command.

Tcl Variable	Set the following variable in a Tcl file, the project files, or from the Tcl window: `set_option -max_parallel_jobs=<n>` This is a global option that is applied to all project files and their implementations. This value takes effect immediately. If you set it in the Tcl file or project file, it remains in effect until you specify a new value. If you set it from the Tcl window, the max_parallel_jobs value is only effective for the session and will be lost when you exit the application.
Configure Compile Point Process Command	The Maximum Number of Parallel Synthesis Jobs option displays the current ini file value and allows you to reset it.

License Utilization for Multiprocessing

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 following:

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.

Note that factors 1 and 3 above can change during a single synthesis run. The number of jobs equals the number of licenses, which then equates the lowest value of these three factors.

Project View Options Command

Select Options->Project View Options to display the Project View Options dialog box, where you define how projects appear and are organized in the Project view.

Synplify Product Only

Synplify Pro/Synplify Premier Products

The following table describes the Project View Options dialog box features.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
Show Project File Library	When enabled, displays the corresponding VHDL library next to each source VHDL filename, in the Project Tree view of the Project view. For example, with library dune, file pc.vhd is listed as [dune] pc.vhd if this option is enabled, and as pc.vhd if it is disabled. See also *Set VHDL Library Command*, on page 172, for how to change the library of a file.
Beep when a job completes	When enabled, sounds an audible signal whenever a project finishes running.
View Project Files in Type Folders	When enabled, organizes project files into separate folders by type. See *View Project Files in Type Folders Option*, on page 337 and add_file, on page 1639.
View Project Files in Custom Folders	When enabled, allows you to view files contained within the custom folders created for the project. See *View Project Files in Custom Folders Option*, on page 338.
Order files alphabetically	When enabled, the software orders the files within folders alphabetically instead of in project order. You can also use the Sort Files option in the Project view.
Autoload projects from previous session	Enable/Disable automatically loading projects from the previous session. Otherwise, projects will not be loaded automatically. This option is enabled by default. See *Loading Projects With the Run Command*, on page 338.
Auto-save project on Run	Enable/Disable automatically saving projects when the Run button is selected. See *Automatically Save Project on Run*, on page 339.
Open Log file following Run	Enable/Disable automatically opening and displaying log file after a synthesis run.
Show all files in results directory	*Synplify Pro and Synplify Premier.* When enabled, shows all files in the Implementation Results view. When disabled, the results directory shows only files generated by the synthesis tool itself.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
Allow multiple projects to be opened	*Synplify Pro and Synplify Premier.* When enabled, multiple projects are displayed at the same time. See *Allow Multiple Projects to be Opened Option*, on page 338.
View log file in HTML	Enable/Disable viewing of log file report in HTML format versus text format. See *Log File*, on page 608.
Project file name display	From the drop-down menu, select one the following ways to display project files: • File name only • Relative file path • Full file path
Use links in SRR log file to individual job logs	Determines if individual job logs use links in the `srr` log file. You can select: • off—appends individual job logs to the `srr` log file. • on—always link to individual job logs. • if_up_to_date—only link to indivdual job logs if the module is up-to-date.

View Project Files in Type Folders Option

View project files in type folders *enabled*

View project files in type folders *disabled*

Copyright © 2013 Synopsys, Inc.

View Project Files in Custom Folders Option

Selecting this option enables you to view user-defined custom folders that contain a predefined subset of project files in various hierarchy groupings or organizational structures. Custom folders are distinguished by their blue color. For information on creating custom folders, see *Creating Custom Folders*, on page 220 in the *User Guide*.

Allow Multiple Projects to be Opened Option

This feature is not supported in the Synplify tool.

Loading Projects With the Run Command

When you load a project that includes the project -run command, a dialog box appears in the Project view with the following message:

```
Project run command encountered during project load. Are you sure
you want to run?
```

You can reply with either yes or no.

Automatically Save Project on Run

If you have modified your project on the disk directory since being loaded into the Project view and you run your design, a message is generated that infers the UI is out-of-date.

The following dialog box appears with a message to which you must reply.

You can specify one of the following:

- Yes — The Auto-save project on Run switch on the Project View Options dialog box is automatically enabled, and then your design is run.
- No — The Auto-save project on Run switch on the Project View Options dialog box is not enabled, but your design is run.
- Cancel — Closes this message dialog box and does not run your design.

Copyright © 2013 Synopsys, Inc.

Editor Options Command

Select Options->Editor Options to display the Editor Options dialog box, where you select either the internal text editor or an external text editor.

The following table describes the Editor Options dialog box features.

Feature	Description
Select Editor	Select an internal or external editor.
• Synopsys Editor	Sets the Synopsys text editor as the default text editor.
• External Editor	Uses the specified external text editor program to view text files from within the Synopsys FPGA tool. The executable specified must open its own window for text editing. See *Using an External Text Editor*, on page 116 of the *User Guide* for a procedure. *Note:* Files opened with an external editor cannot be crossprobed.
Options	Set text editing preferences.

Feature	Description
• File Type	You can define text editor preferences for the following file types: project files, HDL files, log files, constraint files, and default files.
• Font	Lets you define fonts to use with the text editor.
• Font Size	Lets you define font size to use with the text editor.
• Keep Tabs • Tab Size	Lets you define whether to use tab settings with the text editor.
• Syntax Coloring	Lets you define foreground or background syntax coloring to use with the text editor. See *Color Options, on page 341*.

Color Options

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

Copyright © 2013 Synopsys, Inc.

You can set syntax colors for some common syntax options listed in the following table.

Syntax	Description
Comment	Comment strings contained in all file types.
Error	Error messages contained in the log file.
Gates	Gates contained in HDL source files.
Info	Informational messages contained in the log file.
Keywords	Generic keywords contained in the project, HDL source, constraint, and log files.
Line Comment	Line comments contained in the HDL source, C, C++, and log files.
Note	Notes contained in the log file.
SDCKeyword	Constraint-specific keywords contained in the .sdc file.
Strength	Strength values contained in HDL source files.
String DQ	String values within double quotes contained in the project, HDL source, constraint, C, C++, and log files.
String SQ	String values within single quotes contained in the project, HDL source, constraint, C, C++, and log files.
SVKeyword	SystemVerilog keywords contained in the Verilog file.
Types	Type values contained in HDL source files.
Warning	Warning messages contained in the log file.

Copyright © 2013 Synopsys, Inc.

Place and Route Environment Options Command

Select Options->P&R Environment Options to display the environment variable options set for the place-and-route tool. This option allows you to change the specified location of the selected place-and-route tool set on your system; the software locates and runs this updated version of the P&R tool for the current session of the synthesis tool.

Copyright © 2013 Synopsys, Inc.

Configure 3rd Party Tools Options Command

Use the Configure 3rd Party Tools Option command to invoke third-party tools, such as the System Designer or Embedded Development Kit (EDK) from within the Synopsys FPGA products. This allows you to modify source files or libraries added to your synthesis projects from within the third-party tool directly. Use the following dialog box to configure the location and common arguments for the tools.

For more information, see *Invoking Third-Party Vendor Tools*, on page 836 in the *User Guide*.

The 3rd Party Tool Configuration dialog box includes the following options:

Feature	Description
Application Tag Name	Specifies an application or Tcl procedure name. Type in the name or select a preconfigured application from the list.
Direct Execution	Sets up the direct invocation of a third-party tool from within the FPGA synthesis tool, using the path defined for the executable in Application Name with Path.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Tcl Mode	Sets up the tool to execute the Tcl procedure from within the FPGA synthesis tool, using the path defined for the procedure in Tcl Procedure Name. You must execute this Tcl script from the Tcl window to register the invocation procedure for the third-party tool.
Application Name with Path	When using direct execution, specifies the path to the executable for the application.
Tcl Procedure Name	When using Tcl mode, specifies the Tcl procedure name.
Command Argument if any	Defines any additional arguments for the third-party application. You can select arguments from the drop-down list or type them. Note: For internal Synopsys tools, such as the System Designer, you must include the $Syncode parameter.
Procedure Arguments if any	Defines additional arguments for the Tcl procedure. You can select arguments from the drop-down list or type them.

HDL Analyst Options Command

Select Options->HDL Analyst Options to display the HDL Analyst Options dialog box, where you define preferences for the HDL Analyst schematic views (RTL and Technology views). Some preferences take effect immediately, others only take effect in the next view that you open. For details, see *Setting Schematic View Preferences*, on page 557 in the *User Guide*.

For information about the options, see the following, which correspond to the tabs on the dialog box:

- Text Panel, on page 346
- General Panel, on page 347
- Sheet Size Panel, on page 349
- Visual Properties Panel, on page 351

Copyright © 2013 Synopsys, Inc.

Text Panel

The following options are in the Text panel.

Field/Option	Description
Show text	Enables the selective display of schematic labels. Which labels are displayed is governed by the other Show * features and Instance name, described below.
Show port name	When enabled, port names are displayed.
Show symbol name	When enabled, symbol names are displayed.
Show pin name	When enabled, pin names are displayed.
Show bus width	When enabled, connectivity bit ranges are displayed near pins (in square brackets: [ ]), indicating the bits used for each bus connection.
Instance name	Determines how to display instance names: • Show instance name • Show short instance name • No instance name
Set Defaults	Set the dialog box to display the default values.

Copyright © 2013 Synopsys, Inc.

General Panel

The following options are in the General panel.

Field/Option	Description
Show hierarchy browser	When enabled, a hierarchy browser is present as the left pane of RTL and Technology views.
Show tooltip in schematic	When enabled, displays tooltips that hover objects as you move over them in the RTL and Technology schematic views.
Compact symbols	When enabled, symbols are displayed in a slightly more compact manner, to save space in schematics. When this is enabled, Show cell interior is disabled.
Show cell interior	When enabled, the internal logic of cells that are technology-specific primitives (such as LUTs) is shown in Technology views. This is not available if Compact symbols is enabled.
Show sheet connector index	When enabled, sheet connectors show connecting sheet numbers – see *Sheet Connectors, on page 419*.
Compress buses	When enabled, buses having the same source and destination instances are displayed as bundles, to reduce clutter. A single bundle can connect to more than one pin on a given instance. The display of a bundle of buses is similar to that of a single bus.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
No buses in technology view	When enabled, buses are not displayed, they are only indicated as bits in a Technology View. This applies only to flattened views created by HDL Analyst->Technology->Flattened View (or Flattened to Gates View), not to hierarchical views that you have flattened (using, for example, HDL Analyst->Flatten Current Schematic).
Display color-coded clock nets	Displays clock nets in the HDL Analyst View with the color green.
Dissolve levels	The number of levels to dissolve, during HDL Analyst->Dissolve Instances. See *Dissolve Instances*, on page 313
Instances added for expansion	The maximum number of instances to add during any operation (such as HDL Analyst->Hierarchical->Expand) that results in a *filtered* schematic. When this limit is reached, you are prompted to continue adding more instances.

Color-coded Clock Nets

Clock nets are displayed with the color green in the RTL and Technology views.

Sheet Size Panel

Copyright © 2013 Synopsys, Inc.

The following options are in the Sheet Size panel.

Maximum instances	Defines the maximum number of instances to display on a single sheet of an unfiltered schematic. If a given hierarchical level has more than this number of instances, then it will be partitioned into multiple sheets. See *Multiple-sheet Schematics*, on page 432.
Maximum filtered instances	Defines the maximum number of instances to display on a filtered schematic sheet, at any visible hierarchical level. This limit is applied recursively, at each visible *level*, when • the sheet itself is a level, and • each transparent instance is a level (even if inside another transparent instance). Whenever a given level has more child instances inside it than the value of Filtered Instances, it is divided into multiple sheets. (Only children are counted, not grandchildren or below. Instance A is a *child* of instance B if it is inside no other instance that is inside B.) In fact, at each level except the sheet itself, an additional margin of allowable child instances is added to the Maximum filtered instances value, increasing its effective value. This means that you can see more child instances than Maximum filtered instances itself implies. The Maximum filtered instances value must be at least the Maximum instances value. See *Multiple-sheet Schematics*, on page 432.
Maximum Instance Ports	Defines the maximum number of instance pins to display on a schematic sheet.

Visual Properties Panel

Controls the display of the selected property in open HDL Analyst views. The properties are displayed as colored boxes on the relevant objects. To display these properties, the View->Visual Properties command must also be enabled. For more information about properties, see *Viewing Object Properties*, on page 549 in the *User Guide*.

The following options are in the Visual Properties panel.

Show	Toggles the property name and value is displayed in a color-coded box on the object.
Property	Sets the properties to display.
RTL	Enables or disables the display of visual properties in the RTL view.
Tech View	Enables or disables the display of visual properties of in the Technology view.
Value Only	Displays only the value of an item and not its property name.

Copyright © 2013 Synopsys, Inc.

Configure External Programs Command

This command is for Linux platforms only. It lets you specify the web browser and PDF reader for accessing Synopsys support (see *Web Menu*, on page 357 for details) and online documents.

Field/Option	Description
Web Browser	Specify your web browser as an absolute path. You can use the Browse button to locate the browser you need. The default is netscape. If your browser requires additional environment settings, you must do so outside the synthesis tool.
Acrobat Reader	Specify your PDF reader as an absolute path. You can use the Browse button to locate the reader you need. The default is acroread.

Tech-Support Menu

The Tech-Support menu contains information and the actions you can take when you encounter problems running your designs or working with the Synopsys FPGA Implementation products.

Command	Description
Submit Support Request	Opens the Technical Support wizard, which allows you to submit online support requests via SolvNet. The wizard includes provisions for attaching a testcase. See *Submit Support Request Command*, on page 353 for more information.
Web Support	Opens the Synopsys SolvNet Support page from where you can: • Log on to SolvNet to request Synopsys technical support. • Access the Synopsys Products, Downloads, Training, and Documentation pages that have links to product information. See *Web Support Command*, on page 356 for more information.

Submit Support Request Command

To open a request for Synopsys Technical Support, select Submit Support Request from the Tech-Support menu. This command brings up the web-based technical support wizard that helps you prepare the information required to provide technical support for your request through SolvNet.

Copyright © 2013 Synopsys, Inc.

Command	Description
Archive	Brings up the Synopsys Archive Utility to create an archive of your design. Note that designs are limited to 10 MBytes.
Testcase name	The name of the testcase or file to be transferred. To transfer multiple files, use the Synopsys Archive utility to create a single .sar file for the transfer.
Upload	Displays the FTP Archive File form to initiate the transfer of the testcase to the FTP file server. See *FTP Archive File Form*, on page 355.
SolvNet	Displays the Synopsys Sign In screen to access protected Synopsys applications. Signing in opens the SolvNet application which allows you to submit an online support request.

FTP Archive File Form

The FTP Archive File form is displayed when transferring a testcase or file to an FTP file server.

Command	Description
E-Mail Address	Your e-mail address. The name entered is used to create the FTP filename and is automatically entered into the Password field when using the Synopsys site.
Filename	A read-only field displaying the name of the testcase or file to be transferred. A .sar extension is added for single files (archive files are automatically assigned a .sar extension), and the filename is created with the e-mail address.
FTP Destination	Radio buttons that select either the Synopsys site or an alternate site. When Other is specified, you must supply FTP Site, Username, and Password entries.
Status	Reports the status of the transfer.
Transfer	Initiates the FTP file transfer. The results of the transfer are displayed in the Status field.

Copyright © 2013 Synopsys, Inc.

Web Support Command

Through the Synopsys SolvNet Support page, you can access Products, Downloads, Training, and Documentation pages on the Synopsys website as well as submit requests for technical support through SolvNet. First sign in to SolvNet.

Once you logon, the SolvNet Support page is displayed.

Web Menu

This menu contains commands that access up-to-date information from Synopsys Support.

Command	Description
Go to SolvNet	Opens the home page for the Synopsys SolvNet Search support. This website contains links to useful technical information. You can search for new or updated articles or documentation, such as application notes, white papers, release notes, and other user-oriented documentation.
Check Resource Center Messages	Opens a web page that contains updated messages, customized according to the options you set with Set Resource Center Options. You can also access this page by clicking the Message (envelope) icon in the status bar at the bottom of the application window. See *Check Resource Center Messages Command*, on page 358 for additional information.
Configure Resource Center	Lets you set options for Technical Resource Center (TRC) updates. See *Configure Resource Center Command*, on page 360 for details.
Go to Training Center	Opens the Synopsys training web page for Synopsys products. Synopsys offers both online web-based training courses, as well as classroom training courses taught by Synopsys personnel. Select the FPGA Implementation courses from the drop-down menu.
Synopsys Home	Opens the Synopsys home web page for Synopsys products.
FPGA Implementation Tools	Opens the Synopsys FPGA design solution web page for Synopsys FPGA products. You can find information about the full line of Synopsys FPGA Implementation products here.

Copyright © 2013 Synopsys, Inc.

Check Resource Center Messages Command

This command lets you set options for messages from the Resource Center. Se*Using the Resource Center*, on page 358 and *Resource Center Messages*, on page 359 for details.

Using the Resource Center

The following procedure explains how to set preferences and check the Technical Resource Center (TRC) for information.

1. To go to the TRC, select Web->Check Resource Center Messages.
2. To set preferences for accessing the Resource Center, select Web->Configure Resource Center and do the following:

– Set the frequency at which you wish to be notified.

– Select the type of information you want to receive.

– Select the Show new messages available notification window if you wish to receive immediate notification of any new messages.

  If enabled, the following window appears when a new message is added.

– To go to the resource center immediately, click Check Now.

If the TRC has never been configured, the envelope icon in the lower right of the application window appears yellow (to indicate available messages). Once you have configured TRC access, a yellow blinking envelope indicates new messages, and a gray envelope indicates there are no new messages.

3. To check updates, do one of the following:
   - Click the envelope icon in the status bar (lower right of the application window) when it's yellow and blinking to view new information.
   - Select Web->Check Resource Center Messages.

     If you have not set your TRC preferences, the software opens the Resource Center Options dialog box described in the previous step.
4. For other Synopsys FPGA product information, select the following:

Web->Go to SolvNet	http://solvnet.synopsys.com
Web->Go to Training Center	Synopsys training page
Web->Synopsys Home page	Synopsys home page
Web->FPGA Implementation Tools	Synopsys FPGA design solution information page
Tech-Support>Web Support	SolvNet technical support

Resource Center Messages

The envelope icon in the status bar (lower right of the application window) indicates when new messages are available:

Appearance	Signifies
Yellow or blinking yellow	New messages available.
Gray	No new messages, or you have elected not to check for new messages.

Click on the icon to go to the Messages page of the Technical Resource Center.

Copyright © 2013 Synopsys, Inc.

Configure Resource Center Command

Sets options for automatic notification from the Technical Resource Center. From the Resource Center Options dialog box, you determine if you want to receive notification on updates for your product, all products, and/or promotional offers. You also determine how often you want the tool to check for new messages and alerts.

Option	Description
Check for messages and updates	Determines the frequency at which the software checks for updates. Select a setting from the menu: • Each invocation – checks for new messages and alerts each time you start the tool. • Weekly • Daily • Manually – does not automatically check for updates and/or alerts. With this setting, you must remember to go to the Resource Center to check for pertinent information on your product.
Check Now	Opens the home page for the Technical Resource Center (same as the Web->Check Resource Center Messages command).
Check for these types of messages	Determines the type of notification to receive. Check the boxes, as desired. • Messages for this product – Updates and critical bulletins for the product. • Messages for all Synplicity products – Updates and critical bulletins for all Synopsys FPGA products. • Special promotions – Promotional packages and pricing, occasional surveys, and other related information.

Copyright © 2013 Synopsys, Inc.

Help Menu

There are four help systems accessible from the Help menu:

- Help on the Synopsys FPGA synthesis tool (Help->Help)
- Help on standard Tcl commands (Help->TCL)
- Help on error messages (Help->Error Messages)
- Help on using online help (Help->How to Use Help)

The following table describes the Help menu commands. Some commands are only available in certain views.

Command	Description
Help	Displays hyperlinked online help for the product.
Search Solvnet	*Synplify only* Lets you access the Solvnet database. See *Search SolvNet, on page 363* for details.
Demos & Examples	Lets you quickly access information on key features, examples, and an attribute wizard. See *Demos & Examples, on page 101* for a description.
Additional Products	Displays the Synopsys FPGA family of products and a brief description.
How to Use Help	Displays help on how to use Synopsys FPGA online help.
Online Documents	Displays an Open dialog box with hyperlinked PDF documentation on the product including release notes, user guide, reference manual, and licensing configuration and setup. You need Adobe Acrobat Reader® to view the PDF files.
TCL	Displays help for Tcl commands.
Mouse Stroke Tutor	Displays the Mouse Stroke Tutor dialog box which provides information on the available mouse strokes – see *Using Mouse Strokes, on page 108* for details.
License Agreement	Displays the Synopsys FPGA software license agreement.
License Request	Displays a dialog box where you can request a trial license or license upgrade.

Copyright © 2013 Synopsys, Inc.

Command	Description
Floating License Usage	Specifies the number of floating licenses currently being used and their users.
Preferred License Selection	Displays the floating licenses that are available for your selection. See *Preferred License Selection Command*, on page 364.
Tip of the Day	Displays a daily tip on how to use the Synopsys FPGA synthesis tools better. See *Tip of the Day Command*, on page 365.
About this program	Displays the About dialog box, showing the synthesis tool product name, license expiration date, customer identification number, version number, and copyright. Clicking the Versions button in the About dialog box displays the Version Information dialog box, listing the installation directory and the versions of all the synthesis tool compiler and mapper programs.

Search SolvNet

Lets you quickly access SolvNet from within the Project view. In the Synplify tool, you can access it with the Help->Search Solvnet command.

In the Synplify Pro and Synplify Premier tools, click the Search SolvNet button in the Project view to open the dialog box.

Copyright © 2013 Synopsys, Inc.

Option	Description
Search application notes and articles	Specifies a term to search for in the Solvnet database. The tool goes to Solvnet and lists Application Notes and Articles .
Browse all application notes	Takes you to a Solvnet article that has links to all the Application Notes for Synopsys FPGA synthesis products.
Browse all articles	Takes you to the Browse Articles by Product SolvNet page.
Go to tutorial	Takes you to the tutorial page for the Synopsys FPGA product you are using (same as Help->Tutorial).

Preferred License Selection Command

Select Help->Preferred License to display the Select Preferred License dialog box, listing the available licenses for you to choose from. Select a license from the License Type column and click Save. Close and restart the Synopsys FPGA synthesis tool. The new session uses the preferred license you selected.

Tip of the Day Command

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

Popup Menus

Popup menus, available by clicking the right mouse button, offer quick ways to access commonly used menu commands that are specific to the view where you click. Commands shown grayed out (dimmed) are currently inaccessible. Popup menu commands generally duplicate commands

Copyright © 2013 Synopsys, Inc.

available from the regular menus, but sometimes have commands that are only available from the popup menu. The following table lists the popup menus:

Popup Menu	Description
Project view	See *Project View Popup Menus*, on page 372 for details.
SCOPE window	Contains commonly used commands from the Edit menu.
Watch Window	*Synplify Pro and Synplify Premier*. See *Watch Window Popup Menu*, on page 366 for details.
Tcl window	*Synplify Pro and Synplify Premier*. Contains commands from the Edit menu. For details, see *Edit Menu Commands for the Text Editor*, on page 148.
Text Editor window	See *Text Editor Popup Menu*, on page 367 for more information.
RTL and Technology views	See *RTL and Technology Views Popup Menus*, on page 402.
FSM viewer	*Synplify Pro and Synplify Premier*. See *FSM Viewer Popup Menu*, on page 369.
Design Planner	*Synplify Premier*. See *Design Planner Popup Menus*, on page 558.
Physical Analyst	*Synplify Premier*. See *Physical Analyst View Popup Menus*, on page 580.

Watch Window Popup Menu

The Watch window is available only with the Synplify Pro and Synplify Premier products. The Watch window popup menu contains the following commands:

Command	Description
Configure Watch	Displays the Log Watch Configuration dialog box, where you choose the implementations to watch.
Refresh	Refreshes (updates) the window display.
Clear Parameters	Empties the Watch window.

Copyright © 2013 Synopsys, Inc.

For more information on the Watch window and the Configure Watch dialog box, see *Watch Window*, on page 81.

Tcl Window Popup Menu

The Tcl window is available only with the Synplify Pro and Synplify Premier products. The Tcl window popup menu contains the Copy, Paste, and Find commands from the Edit menu, as well as the Clear command, which empties the Tcl window. For information on the Edit menu commands available in the Tcl window, see *Edit Menu Commands for the Text Editor*, on page 148.

Text Editor Popup Menu

The popup menu in the Text Editor window contains the following commonly used text-editing commands from the Edit menu: Undo, Redo, Cut, Copy, Paste, and Toggle Bookmark. In addition, HDL Analyst specific commands appear when both an HDL Analyst view and it's corresponding HDL source file is open. For details of these commands, see *Edit Menu Commands for the Text Editor*, on page 148 and *HDL Analyst Menu*, on page 304.

The following table lists the commands that are unique to the popup menu:

Command	Description
Filter Analyst	Filters your design to show only the currently selected objects in the HDL text file. This is the same as HDL Analyst->Filter Schematic.
Select in Analyst	Crossprobes from the Text Editor and selects the objects in the HDL Analyst view. To use this command, the Enhanced Text Crossprobing (option must be engaged.

Copyright © 2013 Synopsys, Inc.

Log File Popup Menu

The popup menu in the log file contains commands that control operations in the log file. The popup menu differs when the log file is opened in the HTML mode or in the ASCII text mode.

Log File Filter Dialog Box

The Log File Filter dialog box is available by selecting Log File Message Filter from the log file popup menu when the log file is opened in the HDML mode. The dialog box allows messages in the current session to be promoted or demoted in severity or suppressed from the log files for subsequent sessions. For additional information on using this dialog box, see *Log File Message Controls, on page 528* of the *User Guide*.

The following table describes the dialog box functions.

Function	Description
Log File Messages window	Displays the message ID and text and the default message type of messages generated during the current session.
Suppress Message button	Suppresses the selected note, warning, or advisory message. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating suppress status. Note that error messages cannot be suppressed.
Make Error button	Promotes the status of the selected warning (or note) to an error. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating error status.
Make Warning button	Promotes the status of the selected note to a warning. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating warning status.
Make Note button	Demotes the status of the selected warning to a note. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating note status.
Remove Override button	Removes the override status on the selected message in the lower window and returns the message to the upper Log File Messages window.
lower window	Lists the status of all messages that have been promoted, demoted, or suppressed.
OK button	Updates the status of any changed messages in the .pfl file. Note that you must recompile/resynthesize the design before any message status changes become effective.

FSM Viewer Popup Menu

Available with the Synplify Pro and Synplify Premier tools. The popup menu in the FSM Viewer contains commands that determine what is shown in the FSM Viewer. The following table lists the popup commands in the FSM Viewer.

Copyright © 2013 Synopsys, Inc.

Command	Description
Properties	Displays the Object Properties dialog box and view properties of a selected state or transition. Information about a selected transition includes the conditions enabling the transition and the identities of its origin and destination states. Information about a selected state includes its name, RTL encoding, and mapped encoding.
Filter	See *View Menu: FSM Viewer Commands*, on page 162.
Unfilter	See *View Menu: FSM Viewer Commands*, on page 162.
FSM Properties	Displays the Object Properties dialog box indicating the FSM identity and location, encoding style, reset state, and the number of states and transitions.

Copyright © 2013 Synopsys, Inc.

FSM Properties

State properties
(state selected)

Transition properties
(transition selected)

Copyright © 2013 Synopsys, Inc.

Project View Popup Menus

The popup menu commands available in the Project view are context-sensitive, depending on what is currently selected and where in the view you click to open the popup menu. Most commands duplicate commands from the File, Project, Run, and Options menus.

Synplify Project Management View Popup Commands

The following table describes the Project Management view commands that are not duplicated on other menus in the Synplify tool:

Command	Description
Project Management View, No Selections	
Open Project	Displays the Open Project Dialog. See *Open Project Command*, on page 146.
New Project	Creates a new empty project in the Project Window.
Refresh	Refreshes the display.
Project View Options	Displays the Project View Options dialog. See *Project View Options Command*, on page 335.
Project Selected	
Open as Text	Opens the selected file in the Text Editor.
Add File	Displays the Add Files to Project dialog. See *Add Source File Command*, on page 170.
New Implementation	Displays the Implementation Options dialog box. See *Configure Revision Control Tools Command*, on page 184.
Synthesize	Compiles and maps the selected design.
Compile Only	Compiles the selected design.

Copyright © 2013 Synopsys, Inc.

Command	Description
Write Output Netlist Only	Writes the mapped output netlist to structural Verilog (vm) or VHDL (vhm) format. Same as enabling: • Write Mapped Verilog Netlist • Write Mapped VDHL Netlist on the Implementation Results tab of the Implementation Options dialog box.
Arrange VHDL Files	Reorders the VHDL source files.
Save Project	Displays the Save Project As dialog box.
Close Project	Closes your project.
Revision Control Management Menu	The commands on this menu run third-party SCM tools to manage your hierarchical design files. See *Revision Control Management Popup Menu Commands*, on page 391.
Project Folder or File Selected	
Add Folder	Creates a folder with the new name you specified and adds it to the Project view. See *Add Folder Command*, on page 378.
Rename Folder	Renames an existing folder with the new name you specified in the Project view. See *Rename Folder Command*, on page 378.
Delete Folder	Deletes the specified folder and all its contents as necessary. See *Delete Folder Command*, on page 379.
Remove from Folder	Removes the selected file from its corresponding folder.
Place in Folder	Places the selected file into the folder you specify.
Launch Tools->Run Vendor Tool	Launches the vendor application or Tcl procedure tool from the Project view for the selected file of folder. See *Vendor Tool Invocation Popup Menu Command*, on page 380.
Constraint File Selected	
File Options	Displays the File Options dialog box. See *File Options Popup Menu Command*, on page 381.
Open	Opens the SCOPE window.
Open as Text	Opens the selected file in the Text Editor.

Copyright © 2013 Synopsys, Inc.

Command	Description
Copy File	Displays the Copy File dialog box, where you copy the selected file and add it to the current project. You specify a new name for the file. See *Copy File Popup Menu Command*, on page 383.
Change File	Opens the Source File dialog box where you choose a new file to replace the selected file. See *Change File Command*, on page 171.
Remove File From Project	Removes the file from the project.
HDL File Selected	
File Options	Displays the File Options dialog box. See *File Options Popup Menu Command*, on page 381.
Open	Opens the file in the Text Editor.
Syntax Check	Runs a syntax check on your design code. Reports errors, warnings, or notes in the Tcl Window.
Synthesis Check	Runs a synthesis check on your design code. This includes a syntax check and a check to see if the synthesis tool could map the design to the hardware. No optimizations are performed. Reports errors, warnings, or notes in the Tcl Window.
Copy File	Displays the Copy File dialog box, where you copy the selected file and add it to the current project. You specify a new name for the file. See *Copy File Popup Menu Command*, on page 383.
Change File	Opens the Source File dialog box where you choose a new file to replace the selected file. See *Change File Command*, on page 171.
Remove File From Project	Removes the file from the project.
Implementation Selected	
Implementation Options	Displays the Implementation Options dialog box. See *Implementation Options Command*, on page 194.

Copyright © 2013 Synopsys, Inc.

Command	Description
Change Implementation Name	Displays the Implementation Name dialog box, where you rename the selected implementation. See *Change Implementation Popup Menu Commands*, on page 384.
Edit Design Plan	Displays the Synplify Premier Design Planner, where you can edit the design. For more information, see Chapter 12, *Floorplanning with Design Planner*.
Copy Implementation	Copies the selected implementation and adds it to the current project with the name you specify in the dialog box. See *Change Implementation Popup Menu Commands*, on page 384.
Remove Implementation	Removes the selected implementation from the project.
RTL View	Creates an RTL View based on the properties of the selected implementation.
Tech View	Creates a Technology View based on the properties of the selected implementation.
Add P&R Implementation	Displays the Add New Place & Route Task dialog box where you set options to run place & route after synthesis. See *Add P&R Implementation Popup Menu Command*, on page 386
Run	Starts a synthesis run on your design.
Place & Route Implementation Selected	
Add Place & Route Job	Displays the Add New Place & Route Task dialog box, so you can set options and run placement and routing. See *Add P&R Implementation Popup Menu Command*, on page 386.
Remove Place & Route Job	Deletes the place-and-route implementation from the project.
Run Place & Route Job	Runs the place-and-route job for the design.
Run Backannotation Only	Only runs backannotation with placement and timing data immediately following placement and routing.

Copyright © 2013 Synopsys, Inc.

Synplify Pro and Synplify Premier Project Management Commands

The following table lists the popup commands in the Synplify Pro and Synplify Premier Project Management views that are not available on the tool command menus. The Project Management view consists of two tabs, and the table lists the popup commands available in both tabs.

For the Design Hierarchy tab, The Synplify Pro and Synplify Premier tools support all the Synplify project management commands listed in *Synplify Project Management View Popup Commands*, on page 372, as well as the unique commands listed here.

Command	Description
Project Management View -> Project FilesTab Popup Commands	
All Synplify project management commands	Refer to the table in *Synplify Project Management View Popup Commands*, on page 372 for descriptions of these commands.
Project Options	With the project selected, displays project properties such as name and location. See *Project Options Popup Menu Command*, on page 385.
Show Compile Points	Displays the compile points of the selected implementation and lets you edit them. See *Show Compile Points Popup Menu Command*, on page 384.
P & R Options	With a place-and-route implementation selected, displays the Options for Place & Route on Implementation dialog box, so you can change options and rerun placement and routing. See *Options for Place & Route Jobs Popup Menu Command*, on page 390 for a description of the features.
Create Sub-project (Instance)	Makes an instance into a sub-project of the top-level project. See *Create Subproject (Instance) Command*, on page 393.
Set as Black Box	When enabled, specifies that the design block be implemented as a black box during synthesis. Only available when the sub-project is selected.
Design Block Source	Takes you to the design block or instance block definition in the HDL source file. Only available when the sub-project is selected.
Refresh Hierarchy	Refreshes the Design Hierarchy view after design blocks or instance blocks have changed.

Copyright © 2013 Synopsys, Inc.

Command	Description
Properties	Displays the design block or instance block properties. For details, see *Design Block/Instance Properties Popup Menu Command*, on page 395.
Hierarchical Project Options	Configures synthesis run for sub-projects or top-level projects. For details, see *Hierarchical Project Options Command*, on page 182. Same as the Project->Hierarchical Project Options command.
Add SubProject Implementations	Adds new implementations to the blocks in the top-level project. This command is only available when the top-level implementation is selected. Same as the Project-> SubProject Implementation command. See *Working with Multiple Implementations in Hierarchical Projects*, on page 270 in the *User Guide* for information about using this command.
Insert SubProject	Allows you to nest sub-projects within a hierarchy. For details, see *Insert Sub-project Command*, on page 183. Only available as a popup menu command.
Sub-project Parameter Sync	Synchronizes parameters for all the subprojects from the top-level project. See *Sub-project Parameter Sync*, on page 397.
Project Management View -> Design Hierarchy Tab Commands	
Create Sub-project (Design Block)	Defines a design or instance block as a sub-project of the top-level project. See *Create Subproject Popup Menu Commands*, on page 391 for a description.
Insert and Link Subproject to Module	Specifies a design as a subproject and links it to the top-level module. See *Insert & Link Subproject to Module Command*, on page 398 or a description.
Set as Black Box	Specifies that the design block be implemented as a black box during synthesis.
Design Block source	Takes you to the design block or instance block definition in the HDL source file.

Copyright © 2013 Synopsys, Inc.

Command	Description
Refresh Hierarchy	Refreshes the Design Hierarchy view after design blocks or instance blocks have changed.
Properties	Displays the design block or instance block properties. For details, see *Design Block/Instance Properties Popup Menu Command*, on page 395.
Allocate Timing and Resource Budgets	Generates the timing and resource constraints for the instance-based subproject(s) of a hierarchical design. For a description of the dialog box, see *Allocate Timing and Resource Budgets*, on page 400. The Tcl equivalent is generate_instance_constraints. For the syntax description, see *generate_instance_constraints*, on page 1652.

Project Management View Popup Folder Commands

The Project view popup menu includes commands for manipulating folders.

Add Folder Command

Use this option to add a folder to the Project view.

Rename Folder Command

Use this option to rename an existing folder in the Project view.

Delete Folder Command

Use this option to delete a folder from the Project view.

This dialog box includes the following options:

Feature	Description
Yes	Select Yes to delete the folder and all files contained in the folder from the Project view.
No	Select No to delete just the folder from the Project view.
Cancel	Select Cancel, to discontinue the operation.

Copyright © 2013 Synopsys, Inc.

Vendor Tool Invocation Popup Menu Command

Use the Vendor Tool Invocation command to invoke third-party tools, such as the System Designer or Embedded Development Kit (EDK) from within the Synopsys FPGA products. This allows you to modify source files or libraries added to your synthesis projects from within the third-party tool directly. Use the following dialog box to run the vendor tools.

For more information, see *Invoking Third-Party Vendor Tools*, on page 836 in the *User Guide*.

The Vendor Tool Invocation dialog box includes the following options:

Feature	Description
Application Tag Name	Specifies an application or Tcl procedure name. Type in the name or select a preconfigured application from the list.
Additional Options	Defines any additional arguments for the Tcl procedure or third-party application. You can select arguments from the drop-down list or type them. Note: For internal Synopsys tools, such as the System Designer, you must include the $Syncode parameter.

Feature	Description
Command Preview	Sets up the direct invocation of a third-party tool from within the FPGA synthesis tool, using the path defined for the executable in Application Name with Path or to execute the Tcl procedure from within the FPGA synthesis tool, using the path defined for the procedure in Tcl Procedure Name.
Run	The synthesis tool launches the third-party tool or runs the Tcl procedure with the arguments you specified.

File Options Popup Menu Command

To display the File Options dialog box, right-click on a project file and select File Options from the popup menu. Specify the path as relative or absolute when listing the file in the project (`prj`) file and if the file is to be passed to the place-and-route tool or used only for simulation.

Field/Option	Description
File Path	Path to the selected file.
File Type	The folder type for the selected file. You can select the file folder type from a large list of file types. Changing the folder file type does *not* change the file contents or its extension; it simply places the file in the specified Project view folder. For example, if you change the file type of a VHDL file to Verilog, the file retains its Verilog extension, but is moved from the VHDL folder to the Verilog folder.
Library Names	Name of the library which must be compatible with the HDL simulator. For VHDL files, the dialog box is the same as that accessed by Project->Set VHDL Library – see *Set VHDL Library Command*, on page 172.
Last modified	Date the file was last modified.
Save file	The format for the path type: choose either Relative to Project (the default) or with an Absolute Path.

Copyright © 2013 Synopsys, Inc.

Field/Option	Description
Verilog Standard (Verilog only)	Select the Verilog file type from the menu: Use Project Default, Verilog 95, Verilog 2001, or SystemVerilog. Use Project Default sets the type of the selected file to the default for the project (new projects default to SystemVerilog).
Use for Place and Route	Determines if files are automatically passed to the backend place-and-route tool. The files are copied to the place-and-route implementation directory and then invoked when the place-and-route tool is run. For example, IP core wrappers generated by the Altera MegaWizard, as well as their associated instantiated components (including encrypted files) must be passed on to the place-and-route tool since they are not written out to the final netlist.
Use for Simulation Only	Determines if files are only to be used for simulation. For example, files such as test benches containing HDL constructs used only for simulation can be specified using this option.

The following is the Verilog dialog box:

The following is the VHDL dialog box:

Copy File Popup Menu Command

With a file selected, select the Copy File popup menu command to copy the selected file and add it to the current project. This displays the Copy File dialog box where you specify the name of the new file.

Copyright © 2013 Synopsys, Inc.

Change Implementation Popup Menu Commands

With an implementation selected, right-click and select the Change Implementation Name or Copy Implementation popup menu commands to display a dialog box where you specify the new name.

Command	Description
Change Implementation Name	The implementation name you specify is the new name for the implementation.
Copy Implementation	The currently selected implementation is copied and saved to the project with the new implementation name you specify.

Show Compile Points Popup Menu Command

The compile point feature is available only with the Synplify Pro and Synplify Premier products. With an implementation selected, select the Show Compile Points popup menu command to display the Compile Points dialog box and view or edit the compile points of the selected implementation.

Compile points are only available for certain technologies. For more information on compile points and the compile-point synthesis flow, see *Compile Point Types*, on page 1092 and *Synthesizing Compile Points*, on page 1109 of the *User Guide*.

The columns Enb, Module, Type, and Comment in the dialog box correspond to the columns Enabled, Module, Type, and Comment in the SCOPE spreadsheet for the compile point. The File column lists the top-level constraint file where the compile point is defined.

To open and edit the SCOPE spreadsheet for a compile point, either double-click the row of the compile point or select it and click the Edit Compile Point button.

Project Options Popup Menu Command

With a project selected, select the Project Options popup menu command to display the Project Properties dialog box and change the implementation of a project.

In the dialog box, select an implementation in the Implementations list, then click OK or Apply to make it the active implementation of the project.

Copyright © 2013 Synopsys, Inc.

Add P&R Implementation Popup Menu Command

Displays the Add New Place & Route Task dialog box. For information about using this command for place-and-route encapsulation, see *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide*. The place and route feature is not available in the Synplify tool.

Command	Description
Place & Route Implementation Name	Enter a name for the place & route implementation. Do not use spaces for the implementation name.

Copyright © 2013 Synopsys, Inc.

Command	Description
Place & Route Source (Physical Plus)	Enter the type of place & route to run for the implementation. You can choose either: • synthesis—runs place and route using the synthesis flow `edf` as input. • Physical Plus—runs place and route using the Physical Plus flow `edf` as input.
Flow Settings	
Run Place & Route following synthesis	Enable/disable the running of the place & route tool from the synthesis tool immediately following synthesis.
Smart Guide	Enable/disable running the SmartGuide flow.
Run Timing & Placement Backannotation + Generate Congestion Analysis Report	This feature is only available with the Synplify Premier tool for certain technologies. Enable/disable backannotation of placement and timing data immediately following place & route and generate a congestion analysis report.
Use placement constraints from physical synthesis	This option is only available in the Synplify Premier tools for Altera Stratix II, Stratix II GX, Stratix III, Stratix IV, Stratix IV GT, and Stratix V devices. • By default, the Quartus II place-and-route tool uses the placement constraints forward annotated from the physical synthesis tool. • When this option is disabled (not checked), the Quartus II place-and-route tool determines how it will handle physical constraints.
Place & Route Options File	This option lets you specify a place & route options file. You can select either the: • Smart Guide Options File – use this option to run the Xilinx SmartGuide incremental place-and-route flow. • Standard Options File – use this option to run the standard synthesis place-and-route flows. For more information about the SmartGuide flow, see the *Xilinx SmartGuide Flow*, on page 870.
Add Place & Route Options File Existing Options File	This option opens the Select Place & Route option file dialog box where you browse for an existing place & route options file. See *Running P&R Automatically after Synthesis*, on page 842 for information about using this feature.

Command	Description
Add Place & Route Options File Create New Options File	This option opens the Create Place & Route Options File dialog box where you specify a new place & route options file. See *Running P&R Automatically after Synthesis, on page 842* for information about creating a new options file.

Once the par implementation is created, then you can right-click and perform any of the following options:

- P&R Options—See Options for Place & Route Jobs Popup Menu Command, on page 390.
- Add Place & Route Job—See Add P&R Implementation Popup Menu Command, on page 386.

Copyright © 2013 Synopsys, Inc.

- Run Place & Route Job—Runs the place-and-route job.

Options for Place & Route Jobs Popup Menu Command

You can select a place-and-route job for a particular implementation, easily change options and then rerun the job. These options are the same found on the Options for Place & Route on Implementation dialog box. For a description of these options, see *Add P&R Implementation Popup Menu Command*, on page 386.

Revision Control Management Popup Menu Commands

Synplify Premier

The commands on this menu let you use a third-party SCM tool to manage your design files. The following standard source control management (SCM) functions are provided; note that these functions are available depending on the revision control tool you use:

Add Files to Repository	Adds files to the revision control tool.
Check In	Checks in files to commit changes to the revision control tool.
Check Out	Checks out files from the revision control tool.
Get Latest Revision	Get latest revision of files from the revision control tool.
Refresh	Refresh file information to provide SCM file status.
Revert Changes	Discards changes and reverts back to the last checked-in files.

To run these commands, you must first configure and enable the third-party tool with the Project->Configure Revision Control (Beta) command (*Configure Revision Control Tools Command*, on page 184). For information on using revision control tools for hierarchical designs, see *Using Revision Control Tools*, on page 1149 in the *User Guide*.

Create Subproject Popup Menu Commands

Synplify Pro and Synplify Premier

There are two subproject creation commands:

- Create Subproject (Design Block), on page 392
- Create Subproject (Instance) Command, on page 393

Copyright © 2013 Synopsys, Inc.

Create Subproject (Design Block)

The Create Subproject (Design Block) command displays the Create "*blockName*" as Sub-project dialog box, which lets you specify a name, location, and source files for the sub-project you are creating. You can create a sub-project for an instance block or a design block.

For more information about using this command, refer to *Creating Hierarchical Subprojects by Exporting Blocks*, on page 254 in the *User Guide*.

The following table describes the options:

Option	Description
Project Name	Specifies the name of the block sub-project to be created for the block.
Project Location	Specifies the location for the block sub-project.
Full Path	Specifies the full path name of the sub-project file.

Option	Description
Source files from parent project	Lists the source file(s) from the parent project. The selected design block is enabled by default in the source file display window.
Check All	Selects all source files listed in the source file display window. The selected source files are added to the block project when you click OK.
Check Selected	Re-selects the recently unchecked source files in the source file display window. The selected source files are added to the block project when you click OK.
Uncheck all	Disables all the source files selected in the source file display window. Deselected files are not added to the block project when you click OK.
Uncheck Selected	Disables the selected source files in the source file display window. Deselected files are not added to the block project when you click OK.
Add File	Opens the Add Files to Project dialog box, so you can select source files to add to the displayed list. The compiler might not always identify all the source files that belong to the parent project, but you can use Add File to add the missing source files.

Create Subproject (Instance) Command

The Create Subproject (Instance) command displays the Create "*instanceName*" as Sub-project dialog box, which lets you specify a name, location, and source files for the sub-project you are creating. You can create a sub-project for an instance block.

For more information about using this command, refer to *Creating Hierarchical Subprojects by Exporting Instances*, on page 257 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

The following table describes the options:

Option	Description
Project Name	Specifies the name of the sub-project to be created for the instance.
Project Location	Specifies the location for the instance sub-project.
Full Path	Specifies the full path name of the sub-project file.
Source files from parent project	Lists the source file(s) from the parent project. The selected instance is enabled by default in the source file display window. The synthesis tool 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.

Option	Description
Check All	Selects all source files listed in the source file display window. The selected source files are added to the instance project when you click OK.
Check Selected	Re-selects the recently unchecked source files in the source file display window. The selected source files are added to the instance project when you click OK.
Uncheck all	Disables all the source files selected in the source file display window. Deselected files are not added to the instance project when you click OK.
Uncheck Selected	Disables the selected source files in the source file display window. Deselected files are not added to the instance project when you click OK.
Add File	Opens the Add Files to Project dialog box, so you can select source files to add to the displayed list. The compiler might not always identify all the source files that belong to the parent project, but you can use Add File to add the missing source files.

Design Block/Instance Properties Popup Menu Command

Synplify Pro and Synplify Premier

The Properties dialog box in the Design Hierarchy view displays properties for a hierarchical design block or instance block. The properties displayed vary, according to the status of the design block or instance block. The Properties tab is always displayed and lists properties such as the following:

- Current block name
- Original block name
- Language used for the block, such as Verilog
- The kind of block. For example, it could be any of the following:
  - A black box
  - Parameterized (the block can have multiple combinations of parameters)
  - The top-level block

Copyright © 2013 Synopsys, Inc.

  - Implemented as a sub-project
- Block source file

The Parameters or Attributes tabs are displayed for some design or instance blocks. Parameters are user-defined variables in the HDL source code, such as register width. Attributes can be specified in the HDL source or created by the compiler, such as syn_black_box or syn_sub_project. See the following figure.

Insert Sub-project Command

Synplify Pro and Synplify Premier

This command in the Project Files view lets you nest subprojects within a sub-project hierarchy. To do this, right-click on a sub-project and select Insert Subproject from the popup menu. You can add an existing project or a new project from the Insert Project dialog box. Then, begin adding design files to your sub-project after you have created the new project.

Sub-project Parameter Sync

The Sub-project Parameter Sync command synchronizes parameter properties for all the subprojects from the top-level project. To do this, simply click the Synchronize Sub-projects button shown on the dialog box.

Copyright © 2013 Synopsys, Inc.

Insert & Link Subproject to Module Command

Synplify Pro and Synplify Premier

This popup command in the Design Hierarchy view adds the specified design module/instance as a subproject and links it to the top-level module.

Insert & Link Sub-project to Design Block

Highlight the desired module, right-click and Insert & Link Sub-project to Design Block. The following dialog box appears.

Option	Description
Existing Project	Adds the specified module to an existing subproject and links it to the top-level module.
New Project	Adds the specified module to a new subproject and links it to the top-level module.
Project Run-Type	Specifies the run type for how you want the modules synthesized for the subproject: top-down or bottom-up.

Insert & Link Sub-project to Instance

Highlight the desired module, right-click and Insert & Link Sub-project to Instance. The following dialog box appears.

Copyright © 2013 Synopsys, Inc.

Option	Description
Existing Project	Adds the specified instance to an existing subproject and links it to the top-level module.
New Project	Adds the specified instance to a new subproject and links it to the top-level module.
Project Run-Type	Specifies the run type for how you want the instances synthesized for the subproject: top-down or bottom-up.

Allocate Timing and Resource Budgets

The Allocate Timing and Resource Budgets command generates the timing and resource constraints for the instance-based subproject(s) of a hierarchical design. You can access this command by right-clicking on an instance in the Design Hierarchy view or by right-clicking anywhere in the Design Hierarchy view.

Copyright © 2013 Synopsys, Inc.

Option	Description
Instance/Project	Specifies the name of the instance-based subproject.
Timing Budgets	When enabled, sets initial timing constraints for the instance-based subproject, based on the top-level constraints. See *Setting Initial Timing Budgets for Subprojects*, on page 277 in the User Guide for information on using this option.
Resource	When enabled, allocates RAM and DSP resources to the instance-based subproject, based on top-level resources. See *Allocating Resources for Subprojects*, on page 275 in the User Guide for information on using this option.

Copyright © 2013 Synopsys, Inc.

RTL and Technology Views Popup Menus

Some commands are only available from the popup menus in the RTL and Technology views, but most of the commands are duplicates of commands from the HDL Analyst, Edit, and View menus. The popup menus in the RTL and Technology views are nearly identical. See the following:

- Hierarchy Browser Popup Menu Commands, on page 402
- RTL View and Technology View Popup Menu Commands, on page 402
- Simulation Popup Menu Commands in the RTL/Technology View, on page 407

Hierarchy Browser Popup Menu Commands

The following commands become available when you right-click in the Hierarchy Browser of an RTL or Technology view. The Filter, Hide Instances, and Unhide Instances commands are the same as the corresponding commands in the HDL Analyst menu. The following commands are unique to this popup menu.

Command	Description
Collapse All	Collapses all trees in the Hierarchy Browser.
Filter	Highlights and filters objects such as ports, instances, and primitives in the HDL analyst window.
Reload	Refreshes the Hierarchy Browser. Use this if the Hierarchy Browser and schematic view do not match.
Hide/Unhide Instances	Hides or unhides selected instances in the HDL analyst window. For more information on hidden instances, see *Hidden Hierarchical Instances*, on page 423.

RTL View and Technology View Popup Menu Commands

The commands on the popup menu are context-sensitive, and vary depending on the object selected, the kind of view, and where you click. In general, if you have a selected object and you right-click in the background, the menu includes global commands as well as selection-specific commands for the objects. In the Synplify Premier tool, physical regions are treated as

Copyright © 2013 Synopsys, Inc.

hierarchical instances by the HDL Analyst tool. The popup commands available for a selected region are the same as those available for a selected instance.

Most of the commands duplicate commands available on the HDL Analyst menu (see *HDL Analyst Menu*, on page 304). The following table lists the unique commands.

Common Commands

Command	See...
Show Critical Path	HDL Analyst Menu: Timing Commands, on page 311.
Timing Analyst	HDL Analyst Menu: Timing Commands, on page 311.
Find	Find Command (HDL Analyst), on page 152.
Filter Schematic	HDL Analyst Menu: Filtering and Flattening Commands, on page 308.
Push/Pop Hierarchy	*HDL Analyst Menu->RTL and Technology View Submenus*, on page 305.
Select All Schematic	HDL Analyst Menu: Selection Commands, on page 315.
Select All Sheet	HDL Analyst Menu: Selection Commands, on page 315.
Unselect All	HDL Analyst Menu: Selection Commands, on page 315.
Flatten Schematic	HDL Analyst Menu: Filtering and Flattening Commands, on page 308.
Unflatten Current Schematic	HDL Analyst Menu: Filtering and Flattening Commands, on page 308.
HDL Analyst Options	HDL Analyst Options Command, on page 345.
SCOPE->Edit Attributes (object *name*)	Opens a SCOPE window where you can enter attributes for the selected object. It displays the Select Constraint File dialog box (Edit Attributes Popup Menu Command, on page 407), where you select the constraint file to edit. If no constraint file exists, you are prompted to create one.

Copyright © 2013 Synopsys, Inc.

SCOPE->Edit Compile Point Constraints (module *moduleName*)	*Synplify Pro* and *Synplify Premier* For technologies that support compile points, it opens a SCOPE window where you can enter constraints for the selected compile point. It displays the Select Compile Point Definition File dialog box and lets you create or edit a compile-point constraint file for the selected region or instance. See Edit Attributes Popup Menu Command, on page 407.
SCOPE->Edit Module Constraints (module *moduleName*)	*Synplify Pro* and *Synplify Premier* Opens a SCOPE window so you can define module constraints for the selected module). If you do not have a constraint file, it prompts you to create one. The file created is a separate, module-level constraint file.
Simulation	*Synplify Premier* Allows you to watch nets in the Simulation Panel and annotate the following values: • None • Previous, Current, Next • Current For more information, see *Simulation Popup Menu Commands in the RTL/Technology View*, on page 407.

Instance Selected

Command	**See...**
Isolate Paths	Isolate Paths, on page 312.
Expand Paths	Hierarchical->Expand Paths, on page 306.
Current Level Expand Paths	Current Level->Expand Paths, on page 307.
Show Context	Show Context, on page 312.
Hide Instance	Hide Instances, on page 312.
Unhide Instance	Unhide Instances, on page 312.
Show All Hier Pins	Show All Hier Pins, on page 313.
Dissolve Instance	Dissolve Instances, on page 313.

Copyright © 2013 Synopsys, Inc.

Dissolve to Gates	Dissolve to Gates, on page 313.
Generate Dependent File List	Generates a comprehensive list of all of the dependent files for a sub-project. For details, see *Generating Dependent File Lists*, on page 269 in the *User Guide*.

Port Selected

Command	**See...**
Expand to Register/Port	Hierarchical->Expand to Register/Port, on page 306.
Expand Inwards	Hierarchical->Expand Inwards, on page 306.
Current Level->Expand	Current Level->Expand, on page 307.
Current Level->Expand to Register/Port	Current Level->Expand to Register/Port, on page 307.
Current Level->Expand Paths	Current Level->Expand Paths, on page 307.
Properties	Properties Popup Menu Command, on page 406.

Net Selected

Command	**See...**
Goto Net Driver	Hierarchical->Goto Net Driver, on page 307.
Select Net Driver	Hierarchical->Select Net Driver, on page 307.
Select Net Instances	Hierarchical->Select Net Instances, on page 307.
Current Level->Goto Net Driver	Current Level->Goto Net Driver, on page 307.
Current Level->Select Net Driver	Current Level->Select Net Driver, on page 307.
Current Level->Select Net Instances	Current Level->Select Net Instances, on page 307.
Set Net Color	Sets the color of the selected net from a color pallet. For details, see Set Net Color Popup Menu Command, on page 406.

Copyright © 2013 Synopsys, Inc.

Set Net Color Popup Menu Command

The set net color command sets the color of the selected net in the HDL Analyst for the current session. To use the command, select the desired net or nets in the RTL view and select set net color from the popup menu to display the dialog box.

Double click on the corresponding color in the Color column to display the color pallet and then double click the desired color and click OK. Nets can be grouped and assigned to the same color by selecting the same group number in the Group Number column.

Properties Popup Menu Command

The software displays property information about the selected object when you right-click on a net, instance, pin, or port in a HDL Analyst view. See *Visual Properties Panel*, on page 351 or Viewing Object Properties, on page 549 in the *User Guide* for more information about viewing object properties.

Lists pins, if the selected object is an instance or net.
Lists bits, if the selected object is a port.

Edit Attributes Popup Menu Command

Synplify Pro and Synplify Premier. You use the Select a Constraint File dialog box to choose or create a constraint file. You can open the constraint file and edit it. For technologies that support the compile points, it lets you create or edit a compile-point constraint file for the selected region or instance.

For more information about creating constraint files, see *Specifying Xilinx Constraints (Legacy)*, on page 195 of the *User Guide*.

Simulation Popup Menu Commands in the RTL/Technology View

When you use the VCD-Analyst Integration tool, the following simulation popup commands are available if you right-click in the views.

Copyright © 2013 Synopsys, Inc.

For more information see *HDL Analyst Menu: VCD Commands*, on page 316 and *Using the VCD/Identify-Analyst Integration Tool*, on page 904 in the *User Guide*.

Command	Description
Popup Menu Commands in the Simulation Panel View	
Launch DVE	Launches DVE.
Configure DVE	Allows you to configure DVE.
Export Signals to DVE	Exports watched signals from VCS-Analyst tool to DVE.
Import Signals from DVE	Imports watched signals from DVE to the VCS-Analyst tool.
Simulation Properties	Displays properties for the waveform viewer global data and parameters.
VCD->Test Bench Extraction	Generates the test bench files and launches the VCS simulator for the xDUT module. See the *Test Bench Extraction Command*, on page 410.
Popup Menu Commands Within the Name View Fields	
Show Full Name	Displays full name for the signal. Otherwise, use Show Leaf Name.
Popup Menu Commands Within the Signal Value Fields	
Show Current Value Only	Displays the current value for the watched signal. Otherwise, select Show Previous, Current and Next Values.
Popup Menu Commands Within the Name and Signal Value Fields	
Collapse / Collapse all	Collapses selected signals or all signals.
Re-Expand	Expand signals that have been collapsed.
Remove all	Removes all signals from the Simulation Panel watch list.

Copyright © 2013 Synopsys, Inc.

Command	Description
Format	Allows you to change the signal format to: • Enum • Binary • Octal • Unsigned decimal • Decimal • Hex • ASCII
Color	Allows you to change the default color for the signal name or values.
Remove	Removes the selected signal from the Simulation Panel watch list.
Properties	Displays the register properties associated with the selected signal.

Copyright © 2013 Synopsys, Inc.

Test Bench Extraction Command

Use the VCD->Test Bench Extraction command to generate test bench files and launch the VCS simulator for the Design Under Test module (xDUT). For information about the Test Bench Extraction flow, see *VCS Test Bench Extraction*, on page 927.

The following table describes the options:

Option	Description
VCD File and Module Scope	Specifies the path to the simulation VCD file of the RTL DUT and the scope path within the file to the top-level module of the DUT.
Module Under Test (Path/Module)	Shows the xDUT instance path and module name that are read-only fields. To choose another instance path, close the dialog box and select another xDUT.
Test Bench Output	Displays a file browser to select the main xDUT test bench file. The directory for this file contains the stimulus and golden test bench files generated by `vcat`, as well as, log files and other working files. Note: The names of the test bench files are generated by `vcat`.
Parameters	
• Clock Name	Specifies the name of the clock signal for the xDUT. The default is clk.
• Offset	Compares the xDUT outputs with the Golden outputs at the positive transition of the clock. To delay the comparison (for example, to account for propagation delay through the xDUT), you can specify a positive offset in VCD time units. This is typically a small percentage of the clock cycle.
• Reset Name	Specifies the name of the reset signal for the xDUT. The default is rst
• Value	Specifies the value of the reset signal that enables comparison of the xDUT and golden values. The default is 1.
• Time Scale	Displays the VCD time scale in read-only mode.
• Time Range	Enter a time range for the simulation in VCD time units. If left blank, uses the entire duration of the original simulation.

Copyright © 2013 Synopsys, Inc.

Option	Description
• Stimulus Offset	Enter the time in VCD units for delaying application of the stimulus input signals. You might need to correct race conditions by adjusting signal arrival times. Beginning at time 0, `vcat` applies the signals with blocking assignments in both the stimulus and module generation test bench files. Specify an offset, to offset this time. You can also manually edit the files for more complex modifications.
• Golden Offset	Enter the time in VCD units for delaying the application of golden output signals. You might need to correct race conditions by adjusting signal arrival times. Beginning at time 0, `vcat` applies the signals with blocking assignments in both the golden and module generation test bench files. Specify an offset, to offset this time. You can also manually edit the files for more complex modifications.
• Error Limit	Enter a positive number to limit the number of times to report mismatches.
Test Bench Extraction actions that can be performed...	
Gen Test Bench	Generates the test bench files. The following files are generated/updated: • Main xDUT test bench file • Stimulus file for xDUT • Golden file
Edit Test Bench	Allows you to edit the main xDUT test bench file. You are prompted to over-write the existing file.
Edit Stimulus	Allows you to edit the stimulus file for the xDUT. You are prompted to over-write the existing file.
Edit Golden	Allows you to edit the golden file. You are prompted to over-write the existing file.

Copyright © 2013 Synopsys, Inc.

Option	Description
Simulate	Launches VCS simulation for the xDUT. Uses the Synplify Premier/VCS Integration utility to: • Add the xDUT test bench file, stimulus file, and golden file to any test bench files you may have configured manually. • Set the top-level module to the main test bench module. • Specify the -sverilog switch. • Run the simulation script.
VCS Setup	Displays the VCS setup dialog box. Allows you to configure VCS setup.
View Sim Log	Lets you view the simulation log file. This view is continuously updated as simulation progresses.
Implementation	Selects the xDUT implementation to use. You must have a xDUT implementation in order to perform simulation.
• Pre-synthesis	Pre-synthesis simulation is performed.
• Post-synthesis	Post-synthesis simulation is performed. This is the default.
• Post P&R	Post P&R synthesis is performed.

Select Top Level Module Path

Click on the Locate Top Level Scope button to display a dialog box shwon below to designate the top-level scope module. As you click on a module, the scopes are dynamically matched to the netlist.

Copyright © 2013 Synopsys, Inc.

the online help sidebar has three tabs (Contents, Index, and Bookmarks) with the Contents tab active.

CHAPTER 4

HDL Analyst Tool

The HDL Analyst tool helps you examine your design and synthesis results, and analyze how you can improve design performance and area. The HDL Analyst tool is built into the Synplify Pro and Synplify Premier tools, but it is a separate option and requires an additional license in the Synplify software. Enter the HDL Analyst license in your Synplify tool license file. If you need assistance, ask your system administrator or contact Synopsys support.

The following describe the HDL Analyst tool and the operations you can perform with it.

- HDL Analyst Views and Commands, on page 416
- Schematic Objects and Their Display, on page 418
- Basic Operations on Schematic Objects, on page 427
- Multiple-sheet Schematics, on page 432
- Exploring Design Hierarchy, on page 435
- Filtering and Flattening Schematics, on page 442
- Timing Information and Critical Paths, on page 448

For additional information, see the following:

- Descriptions of the HDL Analyst commands in Chapter 3, *User Interface Commands*:
- Chapter 22, *Optimizing Processes for Productivity* in the *User Guide*

Copyright © 2013 Synopsys, Inc.

HDL Analyst Views and Commands

The HDL Analyst tool graphically displays information in two schematic views: the RTL and Technology views (see *RTL View*, on page 90 and *Technology View*, on page 91 for information). The graphic representation is useful for analyzing and debugging your design, because you can visualize where coding changes or timing constraints might reduce area or increase performance.

This section gives you information about the following:

- Filtered and Unfiltered Schematic Views, on page 416
- Accessing HDL Analyst Commands, on page 417

Filtered and Unfiltered Schematic Views

HDL Analyst views (*RTL View*, on page 90 and *Technology View*, on page 91) consist of schematics that let you analyze your design graphically. The schematics can be filtered or unfiltered. The distinction is important because the kind of view determines how objects are displayed for certain commands.

- Unfiltered schematics display all the objects in your design, at appropriate hierarchical levels.
- Filtered schematics show only a subset of the objects in your design, because the other objects have been filtered out by some operation. The Hierarchy Browser in the filtered view always list all the objects in the design, not just the filtered objects. Some commands, such as HDL Analyst -> Show Context, are only available in filtered schematics. Views with a filtered schematic have the word Filtered in the title bar.

Filtering commands affect only the displayed schematic, not the underlying design. For a detailed description of filtering, see *Filtering and*

Flattening Schematics, on page 442. For procedures on using filtering, see *Filtering Schematics*, on page 593 in the *User Guide*.

Accessing HDL Analyst Commands

You can access HDL Analyst commands in many ways, depending on the active view, the currently selected objects, and other design context factors. The software offers these alternatives to access the commands:

- HDL Analyst and View menus
- HDL Analyst popup menus appear when you right-click in an HDL Analyst view. The popup menu is context-sensitive, and includes commonly used commands from the HDL Analyst and View menus, as well as some additional commands.
- HDL Analyst toolbar icons provide shortcuts to commonly used commands

For brevity, this document primarily refers to the menu method of accessing the commands and does not list alternative access methods.

See also:

- HDL Analyst Menu, on page 304
- View Menu, on page 160
- RTL and Technology Views Popup Menus, on page 402
- Analyst Toolbar, on page 116

Copyright © 2013 Synopsys, Inc.

Schematic Objects and Their Display

Schematic objects are the objects that you manipulate in an HDL Analyst schematic: instances, ports, and nets. Instances can be categorized in different ways, depending on the operation: hidden/unhidden, transparent/opaque, or primitive/hierarchical. The following topics describe schematic objects and the display of associated information in more detail:

- Object Information, on page 418
- Sheet Connectors, on page 419
- Primitive and Hierarchical Instances, on page 420
- Hidden Hierarchical Instances, on page 423
- Transparent and Opaque Display of Hierarchical Instances, on page 421
- Schematic Display, on page 423

For most objects, you select them to perform an operation. For some objects like sheet connectors, you do not select them but right-click on them and select from the popup menu commands.

Object Information

To obtain information about specific objects, you can view object properties with the Properties command from the right-click popup menu, or place the pointer over the object and view the object information displayed. With the latter method, information about the object displays in these two places until you move the pointer away:

- The status bar at the bottom of the synthesis window displays the name of the instance, net, port, or sheet connector and other relevant information. If HDL Analyst->Show Timing Information is enabled, the status bar also displays timing information for the object. Here is an example of the status bar information for a net:

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

  You can enable and disable the display of status bar information by toggling the command View -> Status Bar.

- In a tooltip at the mouse pointer
  Displays the name of the object and any attached attributes. The following figure shows tooltip information for a state machine:

To disable tooltip display, select View -> Toolbars and disable the Show Tooltips option. Do this if you want to reduce clutter.

See also

- Pin and Pin Name Display for Opaque Objects, on page 424
- HDL Analyst Options Command, on page 345

Sheet Connectors

When the HDL Analyst tool divides a schematic into multiple sheets, sheet connector symbols indicate how sheets are related. A sheet connector symbol is like a port symbol, but it has an empty diamond with sheet numbers at one end. Use the Options->HDL Analyst Options command (see *Sheet Size Panel*, on page 349) to control how the schematic is divided into multiple sheets.

If you enable the Show Sheet Connector Index option in the (Options->HDL Analyst Options), the empty diamond becomes a hexagon with a list of the connected sheets. You go to a connecting sheet by right-clicking a sheet connector and choosing the sheet number from the popup menu. The menu has as many sheet numbers as there are sheets connected to the net at that point.

Copyright © 2013 Synopsys, Inc.

Show Sheet Connector Index *disabled* Show Sheet Connector Index *enabled*

See also

- Multiple-sheet Schematics, on page 432
- HDL Analyst Options Command, on page 345
- RTL and Technology Views Popup Menus, on page 402

Primitive and Hierarchical Instances

HDL Analyst instances are either primitive or hierarchical, and sorted into these categories in the Hierarchy Browser. Under Instances, the browser first lists hierarchical instances, and then lists primitive instances under Instances->Primitives.

Primitive Instances

Although some primitive objects have hierarchy, the term is used here to distinguish these objects from *user-defined* hierarchies. Primitive instances include the following:

RTL View	Technology View
High-level logic primitives, like XOR gates or priority-encoded multiplexers	Black boxes
Inferred ROMs, RAMs, and state machines	Technology-specific primitives, like LUTs or FPGA block RAMs
Black boxes	
Technology-specific primitives, like LUTs or FPGA block RAMs	

Copyright © 2013 Synopsys, Inc.

In a schematic, logic gate primitives are represented with standard schematic symbols, and technology-specific primitives with various symbols (see *Hierarchy Browser Symbols*, on page 94). You can push into primitives like technology-specific primitives, inferred ROMs, and inferred state machines to view internal details. You cannot push into logic primitives.

Hierarchical Instances

Hierarchical instances are user-defined hierarchies; all other instances are considered to be primitives. Hierarchical instances correspond to Verilog modules and VHDL entities. The Synplify Premier tool also treats physical regions as hierarchical instances.

The Hierarchy Browser lists hierarchical instances under Instances, and uses this symbol: . In a schematic, the display of hierarchical instances depends on the combination of the following:

- Whether the instance is transparent or opaque. Transparent instances show their internal details nested inside them; opaque instances do not. You cannot directly control whether an object is transparent or opaque; the views are automatically generated by certain commands. See *Transparent and Opaque Display of Hierarchical Instances*, on page 421 for details.
- Whether the instance is hidden or not. This is user-controlled, and you can hide instances so that they are ignored by certain commands. See *Hidden Hierarchical Instances*, on page 423 for more information.

Transparent and Opaque Display of Hierarchical Instances

A hierarchical instance can be displayed transparently or opaquely. You cannot directly control the display; certain commands cause instances to be transparent. The distinction between transparent and opaque is important because some commands operate differently on transparent and opaque instances. For example, in a filtered schematic Flatten Current Schematic flattens only transparent hierarchical instances.

- Opaque instances are pale yellow boxes, and do not display their internal hierarchy. This is the default display.

Copyright © 2013 Synopsys, Inc.

- Transparent instances display some or all their lower-level hierarchy nested inside a hollow box with a pale yellow border. Transparent instances are only displayed in filtered schematics, and are a result of certain commands. See *Looking Inside Hierarchical Instances*, on page 440 for information about commands that generate transparent instances.

  A transparent instance can contain other opaque or transparent instances nested inside. The details inside a transparent instance are independent schematic objects and you can operate on them independently: select, push into, hide, and so on. Performing an operation on a transparent object does not automatically perform it on any of the objects nested inside it, and conversely.

See also

- Looking Inside Hierarchical Instances, on page 440
- Multiple Sheets for Transparent Instance Details, on page 434
- Filtered and Unfiltered Schematic Views, on page 416

Hidden Hierarchical Instances

Certain commands do not operate on the lower-level hierarchy of hidden instances, so you can hide instances to focus the operation of a command and improve performance. You hide opaque or transparent hierarchical instances with the Hide Instances command (described in *RTL and Technology Views Popup Menus*, on page 402). Hiding and unhiding only affects the current HDL Analyst view, and does not affect the Hierarchy Browser. You can hide and unhide instances as needed. The hierarchical logic of a hidden instance is not removed from the design; it is only excluded from certain operations.

The schematics indicate hidden hierarchical instances with a small H in the lower left corner. When the mouse pointer is over a hidden instance, the status bar and the tooltip indicate that the instance is hidden.

Schematic Display

The HDL Analyst Options dialog box controls general properties for all HDL Analyst views, and can determine the display of schematic object information. Setting a display option affects all objects of the given type in all views. Some schematic options only take effect in schematic windows opened after the setting change; others affect existing schematic windows as well.

The following are some commonly used settings that affect the display of schematic objects. See *HDL Analyst Options Command*, on page 345 for a complete list of display options.

Copyright © 2013 Synopsys, Inc.

Option	Controls the display of...
Show Cell Interior	Internal logic of technology-specific primitives
Compress Buses	Buses as bundles
Dissolve Levels	Hierarchical levels in a view flattened with HDL Analyst -> Dissolve Instances or Dissolve to Gates, by setting the number of levels to dissolve.
Instances Filtered Instances Instances added for expansion	Instances on a schematic by setting limits to the number of instances displayed
Instance Name Show Conn Name Show Symbol Name Show Port Name	Object labels
Show Pin Name HDL Analyst->Show All Hier Pins	Pin names. See *Pin and Pin Name Display for Opaque Objects*, on page 424 and *Pin and Pin Name Display for Transparent Objects*, on page 425 for details.

Pin and Pin Name Display for Opaque Objects

Although it always displays the pins, the software does not automatically display pin names for opaque hierarchical instances, technology-specific primitives, RAMS, ROMs, and state machines. To display pin names for these objects, enable Options-> HDL Analyst Options->Text->Show Pin Name. The following figures illustrate this display. The first figure shows pins and pin names of an opaque hierarchical instance, and the second figure shows the pins of a technology-specific primitive with its cell contents not displayed.

Copyright © 2013 Synopsys, Inc.

Pin and Pin Name Display for Transparent Objects

This section discusses pin name display for transparent hierarchical instances in filtered views and technology-specific primitives.

Transparent Hierarchical Instances

In a filtered schematic, some of the pins on a transparent hierarchical instance might not be displayed because of filtering. To display all the pins, select the instance and select HDL Analyst -> Show All Hier Pins.

To display pin names for the instance, enable Options->HDL Analyst Options->Text ->Show Pin Name. The software temporarily displays the pin name when you move the cursor over a pin. To keep the pin name displayed even after you move the cursor away, select the pin. The name remains until you select something else.

Copyright © 2013 Synopsys, Inc.

Primitives

To display pin names for technology primitives in the Technology view, enable Options-> HDL Analyst Options->Text->Show Pin Name. The software displays the pin names until the option is disabled. If Show Pin Name is enabled when Options-> HDL Analyst Options->General->Show Cell Interior is also enabled, the primitive is treated like a transparent hierarchical instance, and primitive pin names are only displayed when the cursor moves over the pins. To keep a pin name displayed even after you move the cursor away, select the pin. The name remains until you select something else.

See also:

- HDL Analyst Options Command, on page 345
- Controlling the Amount of Logic on a Sheet, on page 432
- Analyzing Timing in Schematic Views, on page 722 in the *User Guide*

Basic Operations on Schematic Objects

Basic operations on schematic objects include the following:

- Finding Schematic Objects, on page 427
- Selecting and Unselecting Schematic Objects, on page 428
- Crossprobing Objects, on page 429
- Dragging and Dropping Objects, on page 431

For information about other operations on schematics and schematic objects, see the following:

- Filtering and Flattening Schematics, on page 442
- Timing Information and Critical Paths, on page 448
- Multiple-sheet Schematics, on page 432
- Exploring Design Hierarchy, on page 435

Finding Schematic Objects

You can use the following techniques to find objects in the schematic. For step-by-step procedures using these techniques, see *Finding Objects*, on page 568 in the *User Guide*.

- Zooming and panning
- HDL Analyst Hierarchy Browser

  You can use the Hierarchy Browser to browse and find schematic objects. This can be a quick way to locate an object by name if you are familiar with the design hierarchy. See *Browsing With the Hierarchy Browser*, on page 568 in the *User Guide* for details.

- Edit -> Find command

  The Edit -> Find command is described in *Find Command (HDL Analyst)*, on page 152. It displays the Object Query dialog box, which lists schematic objects by type (Instances, Symbols, Nets, or Ports) and lets you use wildcards to find objects by name. You can also fine-tune your search by setting a range for the search.

Copyright © 2013 Synopsys, Inc.

This command selects all found objects, whether or not they are displayed in the current schematic. Although you can search for hidden instances, you cannot find objects that are inside hidden instances at a lower level. Temporarily hiding an instance thus further refines the search range by excluding the internals of a a given instance. This can be very useful when working with transparent instances, because the lower-level details appear at the current level, and cannot be excluded by choosing Current Level Only. See *Using Find for Hierarchical and Restricted Searches*, on page 570 in the *User Guide*.

- Edit -> Find command combined with filtering

  Edit->Find enhances filtering. Use Find to select by name and hierarchical level, and then filter the design to limit the display to the current selection. Unselected objects are removed. Because Find only adds to the current selection (it never deselects anything already selected), you can use successive searches to build up exactly the selection you need, before filtering.

- Filtering before searching with Edit->Find

  Filtering helps you to fine-tune the range of a search. You can search for objects just within a filtered schematic by limiting the search range to the Current Level Only.

  Filtering adds to the expressive power of displaying search results. You can find objects on different sheets and filter them to see them all together at once. Filtering collapses the hierarchy visually, showing lower-level details nested inside transparent higher-level instances. The resulting display combines the advantage of a high-level, abstract view with detail-rich information from lower levels.

  See *Filtering and Flattening Schematics*, on page 442 for further information.

Selecting and Unselecting Schematic Objects

Whenever an object is selected in one place it is selected and highlighted everywhere else in the synthesis tool, including all Hierarchy Browsers, all schematics, and the Text Editor. Many commands operate on the currently selected objects, whether or not those objects are visible.

Copyright © 2013 Synopsys, Inc.

The following briefly list selection methods; for a concise table of selection procedures, see *Selecting Objects in the RTL/Technology Views*, on page 553 in the *User Guide*.

Using the Mouse to Select a Range of Schematic Objects

In a Hierarchy Browser, you can select a *range* of schematic objects by clicking the name of an object at one end of the range, then holding the Shift key while clicking the name of an object at the other end of the range.To use the mouse for selecting and unselecting objects in a schematic, the cross-hairs symbol (+) must appear as the mouse pointer. If this is not currently the case, right-click the schematic background.

Using Commands to Select Schematic Objects

You can select and deselect schematic objects using the commands in the HDL Analyst menu, or use Edit->Find to find and select objects by name.

The HDL Analyst menu commands that affect selection include the following:

- Expansion commands like Expand, Expand to Register/Port, Expand Paths, and Expand Inwards select the objects that result from the expansion. This means that (except for Expand to Register/Port) you can perform successive expansions and expand the set of objects selected.
- The Select All Schematic and Select All Sheet commands select all instances or ports on the current schematic or sheet, respectively.
- The Select Net Driver and Select Net Instances commands select the appropriate objects according to the hierarchical level you have chosen.
- Deselect All deselects all objects in *all* HDL Analyst views.

See also

- Finding Schematic Objects, on page 427
- HDL Analyst Menu, on page 304

Crossprobing Objects

Crossprobing helps you diagnose where coding changes or timing constraints might reduce area or increase performance. When you crossprobe, you select an object in one place and it or its equivalent is automatically selected and

Copyright © 2013 Synopsys, Inc.

highlighted in other places. For example, selecting text in the Text Editor automatically selects the corresponding logic in all HDL Analyst views. Whenever a net is selected, it is highlighted through all the hierarchical instances it traverses, at all schematic levels.

Crossprobing Between Different Views

You can crossprobe objects (including logic inside hidden instances) between RTL views, Technology views, the FSM Viewer, HDL source code files, and other text files. Some RTL and source code objects are optimized away during synthesis, so they cannot be crossprobed to certain views.

The following table summarizes crossprobing to and from HDL Analyst (RTL and Technology) views. For information about crossprobing procedures, see *Crossprobing*, on page 581 in the *User Guide*.

From . . .	To . . .	Do this . . .
Text Editor: log file	Text Editor: HDL source file	Double-click a log file note, error, or warning. The corresponding HDL source code appears in the Text Editor.
Text Editor: HDL code	Analyst view FSM Viewer (Synplify Pro and Synplify Premier only)	The RTL view or Technology view must be open. Select the code in the Text Editor that corresponds to the object(s) you want to crossprobe. The object corresponding to the selected code is automatically selected in the target view, if an HDL source file is in the Text Editor. Otherwise, right-click and choose the Select in Analyst command. To cross-probe from text other than source code, first select Options->HDL Analyst Options and then enable Enhanced Text Crossprobing.
FSM Viewer	Analyst view	The target view must be open. The state machine must be encoded with the onehot style to crossprobe from the transition table. Select a state anywhere in the FSM Viewer (bubble diagram or transition table). The corresponding object is automatically selected in the HDL Analyst view.

Copyright © 2013 Synopsys, Inc.

From . . .	To . . .	Do this . . .
Analyst view FSM Viewer	Text Editor	Double-click an object. The source code corresponding to the object is automatically selected in the Text Editor, which is opened to show the selection. If you just select an object, without double-clicking it, the corresponding source code is still selected and displayed in the editor (provided it is open), but the editor window is not raised to the front.
Analyst view	Another open view	Select an object in an HDL Analyst view. The object is automatically selected in all open views. If the target view is the FSM Viewer (Synplify Pro and Synplify Premier only), then the state machine must be encoded as onehot.
Tcl window	Text Editor	Double-click an error or warning message (available in the Tcl window errors or warnings panel, respectively). The corresponding source code is automatically selected in the Text Editor, which is opened to show the selection.
Text Editor: any text containing instance names, like a timing report	Corresponding instance	Highlight the text, then right-click & choose Select or Filter. Use this to filter critical paths reported in a text file by the FPGA timing analysis tool.

Dragging and Dropping Objects

You can drag and drop objects like instances, nets and pins from the HDL Analyst schematic views to other windows to help you analyze your design or set constraints. You can drag and drop objects from an RTL or Technology views to the following other windows:

- SCOPE editor
- Text editor window
- Tcl window

Copyright © 2013 Synopsys, Inc.

Multiple-sheet Schematics

When there is too much logic to display on a single sheet, the HDL Analyst tool uses additional schematic sheets. Large designs can take several sheets. In a hierarchical schematic, each module consists of one or more sheets. Sheet connector symbols (*Sheet Connectors*, on page 419) mark logic connections from one sheet to the next.

For more information, see

- Controlling the Amount of Logic on a Sheet, on page 432
- Navigating Among Schematic Sheets, on page 432
- Multiple Sheets for Transparent Instance Details, on page 434

Controlling the Amount of Logic on a Sheet

You can control the amount of logic on a schematic sheet using the options in Options->HDL Analyst Options->Sheet Size. The Maximum Instances option sets the maximum number of instances on an unfiltered schematic sheet. The Maximum Filtered Instances option sets the maximum number of instances displayed at any given hierarchical level on a filtered schematic sheet.

See also:

- HDL Analyst Options Command, on page 345
- Setting Schematic View Preferences, on page 557 of the *User Guide.*

Navigating Among Schematic Sheets

This section describes how to navigate among the sheets in a given schematic. The window title bar lets you know where you are at any time.

Multisheet Orientation in the Title Bar

The window title bar of an RTL view or Technology view indicates the current context. For example, uc_alu (of module alu) in the title indicates that the current schematic level displays the instance uc_alu (which is of module alu). The objects shown are those comprising that instance.

Copyright © 2013 Synopsys, Inc.

The title bar also indicates, for the current schematic, the number of the displayed sheet, and the total number of sheets — for example, sheet 2 of 4. A schematic is initially opened to its first sheet.

Navigating Among Sheets

You can navigate among different sheets of a schematic in these ways:

- Follow a sheet connector, by right-clicking it and choosing a connecting sheet from the popup menu
- Use the sheet navigation commands of the View menu: Next Sheet, Previous Sheet, and View Sheets, or their keyboard shortcut or icon equivalents
- Use the history navigation commands of the View menu (Back and Forward), or their keyboard shortcuts or icon equivalents to navigate to sheets stored in the display history

For details, see *Working with Multisheet Schematics, on page 555* in the *User Guide*.

You can navigate among different design levels by pushing and popping the design hierarchy. Doing so adds to the display history of the View menu, so you can retrace your push/pop steps using View -> Back and View->Forward. After pushing down, you can either pop back up or use View->Back.

Copyright © 2013 Synopsys, Inc.

See also:

- Filtering and Flattening Schematics, on page 442
- View Menu: RTL and Technology Views Commands, on page 162
- Pushing and Popping Hierarchical Levels, on page 435

Multiple Sheets for Transparent Instance Details

The details of a transparent instance in a filtered view are drawn in two ways:

- Generally, these interior details are spread out over multiple sheets at the same schematic level (module) as the instance that contains them. You navigate these sheets as usual, using the methods described in *Navigating Among Schematic Sheets*, on page 432.
- If the number of nested contents exceeds the limit set with the Filtered Instances option (Options->HDL Analyst Options), the nested contents are drawn on separate sheets. The parent hierarchical instance is empty, with a notation (for example, Go to sheets 4-16) inside it, indicating which sheets contain its lower-level details. You access the sheets containing the lower-level details using the sheet navigation commands of the View menu, such as Next Sheet.

See also:

- Controlling the Amount of Logic on a Sheet, on page 432
- View Menu: RTL and Technology Views Commands, on page 162

Copyright © 2013 Synopsys, Inc.

Exploring Design Hierarchy

The hierarchy in your design can be explored in different ways. The following sections explain how to move between hierarchical levels:

- Pushing and Popping Hierarchical Levels, on page 435
- Navigating With a Hierarchy Browser, on page 438
- Looking Inside Hierarchical Instances, on page 440

Pushing and Popping Hierarchical Levels

You can navigate your design hierarchy by pushing down into a high-level schematic object or popping back up. Pushing down into an object takes you to a lower-level schematic that shows the internal logic of the object. Popping up from a lower level brings you back to the parent higher-level object.

Pushing and popping is best suited for traversing the hierarchy of a specific object. If you want a more general view of your design hierarchy, use the Hierarchy Browser instead. See *Navigating With a Hierarchy Browser*, on page 438 and *Looking Inside Hierarchical Instances*, on page 440 for other ways of viewing design hierarchy.

Pushable Schematic Objects

To push into an instance, it must have hierarchy. You can push into the object regardless of its position in the design hierarchy; for example, you can push into the object if it is shown nested inside a transparent instance. You can push down into the following kinds of schematic objects:

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

Copyright © 2013 Synopsys, Inc.

When you push/pop, the HDL Analyst window displays the appropriate level of design hierarchy, except in the following cases:

- When you push into an inferred state machine in an RTL view, in the Synplify tool, the `statemachine.info` text file opens. In the Synplify Pro and Synplify Premier tools, the FSM Viewer opens, with graphical information about the FSM. See the *FSM Viewer Window*, on page 95, for more information.
- When you push into an inferred ROM in an RTL view, the Text Editor window opens and displays the ROM data table (`rom.info` file).

You can use the following indicators to determine whether you can push into an object:

- The mouse pointer shape when Push/Pop mode is enabled. See *How to Push and Pop Hierarchical Levels*, on page 436 for details.
- A small H symbol (H) in the lower left corner indicates a hidden instance, and you cannot push into it.
- The Hierarchy Browser symbols indicates the type of instance and you can use that to determine whether you can push into an object. For example, hierarchical instance ( ), technology-specific primitive ( ), logic primitive such as XOR ( ), or other primitive instance ( ). The browser symbol does not indicate whether or not an instance is hidden.
- The *status bar* at the bottom of the main synthesis tool window reports information about the object under the pointer, including whether or not it is a hidden instance or a primitive.

How to Push and Pop Hierarchical Levels

You push/pop design levels with the HDL Analyst Push/Pop mode. To enable or disable this mode, toggle View->Push/Pop Hierarchy, use the icon, or use the appropriate mouse strokes.

Copyright © 2013 Synopsys, Inc.

Once Push/Pop mode is enabled, you push or pop as follows:

- To *pop*, place the pointer in an empty area of the schematic background, then click or use the appropriate mouse stroke. The background area inside a transparent instance acts just like the background area outside the instance.
- To *push* into an object, place the mouse pointer over the object and click or use the appropriate mouse stroke. To push into a transparent instance, place the pointer over its pale yellow border, not its hollow (white) interior. Pushing into an object nested inside a transparent hierarchical instance descends to a lower level than pushing into the enclosing transparent instance. In the following figure, pushing into transparent instance inst2 descends one level; pushing into nested instance inst2.II_3 descends two levels.

The following arrow mouse pointers indicate status in Push/Pop mode. For other indicators, see *Pushable Schematic Objects*, on page 435.

Copyright © 2013 Synopsys, Inc.

A down arrow	Indicates that you can push (descend) into the object under the pointer and view its details at the next lower level.
An up arrow	Indicates that there is a hierarchical level above the current sheet.
A crossed-out double arrow	Indicates that there is no accessible hierarchy above or below the current pointer position. If the pointer is over the schematic background it indicates that the current level is the top and you cannot pop higher. If the pointer is over an object, the object is an object you cannot push into: a non-hierarchical instance, a hidden hierarchical instance, or a black box.

See also:

- Hidden Hierarchical Instances, on page 423
- Transparent and Opaque Display of Hierarchical Instances, on page 421
- Using Mouse Strokes, on page 108
- Navigating With a Hierarchy Browser, on page 438

Navigating With a Hierarchy Browser

Hierarchy Browsers are designed for locating objects by browsing your design. To move between design levels of a particular object, use Push/Pop mode (see *Pushing and Popping Hierarchical Levels*, on page 435 and *Looking Inside Hierarchical Instances*, on page 440 for other ways of viewing design hierarchy).

The browser in the RTL view displays the hierarchy specified in the RTL design description. The browser in the Technology view displays the hierarchy of your design after technology mapping.

Selecting an object in the browser displays it in the schematic, because the two are linked. Use the Hierarchy Browser to traverse your hierarchy and select ports, nets, components, and submodules. The browser categorizes the objects, and accompanies each with a symbol that indicates the object type. The following figure shows crossprobing between a schematic and the hierarchy browser.

Copyright © 2013 Synopsys, Inc.

Selecting an object in the browser causes the schematic to display it also.

Explore the browser hierarchy by expanding or collapsing the categories in the browser. You can also use the arrow keys (left, right, up, down) to move up and down the hierarchy and select objects. To select more than one object, press Ctrl and select the objects in the browser. To select a range of schematic objects, click an object at one end of the range, then hold the Shift key while clicking the name of an object at the other end of the range.

See also:

- Crossprobing Objects, on page 429
- Pushing and Popping Hierarchical Levels, on page 435
- Hierarchy Browser Popup Menu Commands, on page 402

Copyright © 2013 Synopsys, Inc.

Looking Inside Hierarchical Instances

An alternative method of viewing design hierarchy is to examine transparent hierarchical instances (see *Navigating With a Hierarchy Browser*, on page 438 and *Navigating With a Hierarchy Browser*, on page 438 for other ways of viewing design hierarchy). A transparent instance appears as a hollow box with a pale yellow border. Inside this border are transparent and opaque objects from lower design levels.

Transparent instances provide design context. They show the lower-level logic nested within the transparent instance at the current design level, while pushing shows the same logic a level down. The following figure compares the same lower-level logic viewed in a transparent instance and a push operation:

You cannot control the display of transparent instances directly. However, you can perform the following operations, which result in the display of transparent instances:

- Hierarchically expand an object (using the expansion commands in the HDL Analyst menu).
- Dissolve selected hierarchical instances in a *filtered* schematic (HDL Analyst -> Dissolve Instances).
- Filter a schematic, after selecting multiple objects at more than one level. See *Commands That Result in Filtered Schematics*, on page 442 for additional information.

These operations only make *non-hidden hierarchical* instances transparent. You cannot dissolve hidden or primitive instances (including technology-specific primitives). However, you can do the following:

- Unhide hidden instances, then dissolve them.
- Push down into technology-specific primitives to see their lower-level details, and you can show the interiors of all technology-specific primitives.

See also:

- Pushing and Popping Hierarchical Levels, on page 435
- Navigating With a Hierarchy Browser, on page 438
- HDL Analyst Command, on page 305
- Transparent and Opaque Display of Hierarchical Instances, on page 421
- Hidden Hierarchical Instances, on page 423

Copyright © 2013 Synopsys, Inc.

Filtering and Flattening Schematics

This section describes the HDL Analyst commands that result in filtered and flattened schematics. It describes

- Commands That Result in Filtered Schematics, on page 442
- Combined Filtering Operations, on page 443
- Returning to The Unfiltered Schematic, on page 444
- Commands That Flatten Schematics, on page 444
- Selective Flattening, on page 446
- Filtering Compared to Flattening, on page 447

Commands That Result in Filtered Schematics

A filtered schematic shows a subset of your design. Any command that *results in a filtered schematic* is a filtering command. Some commands, like the Expand commands, increase the amount of logic displayed, but they are still considered filtering commands because they result in a filtered view of the design. Other commands like Filter Schematic and Isolate Paths remove objects from the current display.

Filtering commands include the following:

- Filter Schematic, Isolate Paths – reduce the displayed logic.
- Dissolve Instances (in a filtered schematic) – makes selected instances transparent.
- Expand, Expand to Register/Port, Expand Paths, Expand Inwards, Select Net Driver, Select Net Instances – display logic connected to the current selection.
- Show Critical Path, Flattened Critical Path, Hierarchical Critical Path – show critical paths.

All the filtering commands, except those that display critical paths, operate on the currently selected schematic object(s). The critical path commands operate on your entire design, regardless of what is currently selected.

All the filtering commands except Isolate Paths are accessible from the HDL Analyst menu; Isolate Paths is in the RTL view and Technology view popup menus (along with most of the other commands above).

Copyright © 2013 Synopsys, Inc.

For information about filtering procedures, see *Filtering Schematics*, on page 593 in the *User Guide*.

See also:

- Filtered and Unfiltered Schematic Views, on page 416
- HDL Analyst Menu, on page 304 and RTL and Technology Views Popup Menus, on page 402

Combined Filtering Operations

Filtering operations are designed to be used in combination, successively. You can perform a sequence of operations like the following:

1. Use Filter Schematic to filter your design to examine a particular instance. See *HDL Analyst Menu: Filtering and Flattening Commands*, on page 308 for a description of the command.
2. Select Expand to expand from one of the output pins of the instance to add its immediate successor cells to the display. See *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306 for a description of the command.
3. Use Select Net Driver to add the net driver of a net connected to one of the successors. See *HDL Analyst Menu: Hierarchical and Current Level Submenus*, on page 306 for a description of the command.
4. Use Isolate Paths to isolate the net driver instance, along with any of its connecting paths that were already displayed. See *HDL Analyst Menu: Analysis Commands*, on page 311 for a description of the command.

Filtering operations add their resulting filtered schematics to the history of schematic displays, so you can use the View menu Forward and Back commands to switch between the filtered views. You can also combine filtering with the search operation. See *Finding Schematic Objects*, on page 427 for more information.

Copyright © 2013 Synopsys, Inc.

Returning to The Unfiltered Schematic

A filtered schematic often loses the design context, as it is removed from the display by filtering. After a series of multiple or complex filtering operations, you might want to view the context of a selected object. You can do this by

- Selecting a higher level object in the Hierarchy Browser; doing so always crossprobes to the corresponding object in the original schematic.
- Using Show Context to take you directly from a selected instance to the corresponding context in the original, unfiltered schematic.
- Using Goto Net Driver to go from a selected net to the corresponding context in the original, unfiltered schematic.

There is no Unfilter command. Use Show Context to see the unfiltered schematic containing a given instance. Use View->Back to return to the previous, unfiltered display after filtering an unfiltered schematic. You can go back and forth between the original, unfiltered design and the filtered schematics, using the commands View->Back and Forward.

See also:

- RTL and Technology Views Popup Menus, on page 402
- RTL and Technology Views Popup Menus, on page 402
- View Menu: RTL and Technology Views Commands, on page 162

Commands That Flatten Schematics

A flattened schematic contains no hierarchical objects. Any command that results in a flattened schematic is a flattening command. This includes the following.

Command	Unfiltered Schematic	Filtered Schematic
Dissolve Instances	Flattens selected instances	--
Flatten Current Schematic (Flatten Schematic)	Flattens at the current level and all lower levels. RTL view: flattens to generic logic level Technology view: flattens to technology-cell level	Flattens only non-hidden transparent hierarchical instances; opaque and hidden hierarchical instances are not flattened.
RTL->Flattened View	Creates a new, unfiltered RTL schematic of the entire design, flattened to the level of generic logic cells.	
Technology-> Flattened View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of technology cells.	
Technology-> Flattened to Gates View	Creates a new, unfiltered Technology schematic of the entire design, flattened to the level of Boolean logic gates.	
Technology-> Flattened Critical Path	Creates a filtered, flattened Technology view schematic that shows only the instances with the worst slack times and their path.	
Unflatten Schematic	Undoes any flattening done by Dissolve Instances and Flatten Current Schematic at the current schematic level. Returns to the original schematic, as it was before flattening (and any filtering).	

All the commands are on the HDL Analyst menu except Unflatten Schematic, which is available in a schematic popup menu.

The most versatile commands, are Dissolve Instances and Flatten Current Schematic, which you can also use for selective flattening (*Selective Flattening*, on page 446).

See also:

- Filtering Compared to Flattening, on page 447
- Selective Flattening, on page 446

Copyright © 2013 Synopsys, Inc.

Selective Flattening

By default, flattening operations are not very selective. However, you can selectively flatten particular instances with these command (see *RTL and Technology Views Popup Menus*, on page 402 for descriptions):

- Use Hide Instances to hide instances that you do *not* want to flatten, then flatten the others (flattening operations do not recognize hidden instances). After flattening, you can Unhide Instances that are hidden.
- Flatten selected hierarchical instances using one of these commands:
  - If the current schematic is unfiltered, use Dissolve Instances.
  - If the schematic is filtered, use Dissolve Instances, followed by Flatten Current Schematic. In a filtered schematic, Dissolve Instances makes the selected instances transparent and Flatten Current Schematic flattens only transparent instances.

The Dissolve Instances and Flatten Current Schematic (or Flatten Schematic) commands behave differently in filtered and unfiltered schematics as outlined in the following table:

Command	Unfiltered Schematic	Filtered Schematic
Dissolve Instances	Flattens selected instances	Provides virtual flattening: makes selected instances transparent, displaying their lower-level details.
Flatten Current Schematic Flatten Schematic	Flattens *everything* at the current level and below	Flattens only the non-hidden, *transparent* hierarchical instances: does not flatten opaque or hidden instances. See below for details of the process.

In a filtered schematic, flattening with Flatten Current Schematic is actually a two-step process:

1. The transparent instances of the schematic are flattened in the context of the entire design. The result of this step is the entire hierarchical design, with the transparent instances of the filtered schematic replaced by their internal logic.
2. The original filtering is then restored: the design is refiltered to show only the logic that was displayed before flattening.

Copyright © 2013 Synopsys, Inc.

Although the result displayed is that of Step 2, you can view the intermediate result of Step 1 with View->Back. This is because the display history is erased before flattening (Step 1), and the result of Step 1 is added to the history as if you had viewed it.

Filtering Compared to Flattening

As a general rule, use filtering to examine your design, and flatten it only if you really need it. Here are some reasons to use filtering instead of flattening:

- Filtering before flattening is a more efficient use of computer time and memory. Creating a new view where everything is flattened can take considerable time and memory for a large design. You then filter anyway to remove the flattened logic you do not need.
- Filtering is selective. On the other hand, the default flattening operations are global: the entire design is flattened from the current level down. Similarly, the inverse operation (UnFlatten Schematic) unflattens everything on the current schematic level.
- Flattening operations eliminate the *history* for the current view: You can not use View->Back after flattening. (You can, however, use UnFlatten Schematic to regenerate the unflattened schematic.).

See also:

- RTL and Technology Views Popup Menus, on page 402
- Selective Flattening, on page 446

Copyright © 2013 Synopsys, Inc.

Timing Information and Critical Paths

The HDL Analyst tool provides several ways of examining critical paths and timing information, to help you analyze problem areas. The different ways are described in the following sections.

- Timing Reports, on page 448
- Critical Paths and the Slack Margin Parameter, on page 449
- Examining Critical Path Schematics, on page 450

See the following for more information about timing and result analysis:

- Watch Window, on page 81
- Log File, on page 608
- Chapter 22, *Optimizing Processes for Productivity* in the *User Guide*

Timing Reports

When you synthesize a design, a default timing report is automatically written to the log file, which you can view using View->View Log File. This report provides a clock summary, I/O timing summary, and detailed timing information for your design.

For certain device technologies, you can use the Analysis->Timing Analyst command to generate a custom timing report. Use this command to specify start and end points of paths whose timing interests you, and set a limit for the number of paths to analyze between these points. By default, the sequential instances, input ports, and output ports that are currently selected in the Technology views of the design are the candidates for choosing start and end points. In addition, the start and end points of the previous Timing Analyst run become the default start and end points for the next run. When analyzing timing, any latches in the path are treated as level-sensitive registers.

The custom timing report is stored in a text file named *resultsfile*.ta, where *resultsfile* is the name of the results file (see *Implementation Results Panel*, on page 203). In addition, a corresponding output netlist file is generated, named *resultsfile*_ta.srm. Both files are in the implementation results directory.

The Timing Analyst dialog box provides check boxes for viewing the text report (Open Report) in the Text Editor and the corresponding netlist (Open Schematic) in a Technology view. This Technology view of the timing path, labeled Timing View in the title bar, is special in two ways:

- The Timing View shows only the paths you specify in the Timing Analyst dialog box. It corresponds to a special design netlist that contains critical timing data.
- The Timing Analyst and Show Critical Path commands (and equivalent icons and shortcuts) are unavailable whenever the Timing View is active.

See also:

- Analysis Menu, on page 284
- Timing Reports, on page 615
- Log File, on page 608

Critical Paths and the Slack Margin Parameter

The HDL Analyst tool can isolate critical paths in your design, so that you can analyze problem areas, add timing constraints where appropriate, and resynthesize for better results.

After you successfully run synthesis, you can display just the critical paths of your design using any of the following commands from the HDL Analyst menu:

- Hierarchical Critical Path
- Flattened Critical Path
- Show Critical Path

The first two commands create a new Technology view, hierarchical or flattened, respectively. The Show Critical Path command reuses the current Technology view. Neither the current selection nor the current sheet display have any effect on the result. The result is flat if the entire design was already flat; otherwise it is hierarchical. Use Show Critical Path if you want to maintain the existing display history.

All these commands filter your design to show only the instances (and their paths) with the worst slack times. They also enable HDL Analyst -> Show Timing Information, displaying timing information.

Copyright © 2013 Synopsys, Inc.

Negative slack times indicate that your design has not met its timing requirements. The worst (most negative) slack time indicates the amount by which delays in the critical path cause the timing of the design to fail. You can also obtain a *range* of worst slack times by setting the *slack margin* parameter to control the sensitivity of the critical-path display. Instances are displayed only if their slack times are within the slack margin of the (absolutely) worst slack time of the design.

The slack margin is the criterion for distinguishing worst slack times. The larger the margin, the more relaxed the measure of worst, so the greater the number of critical-path instances displayed. If the slack margin is zero (the default value), then only instances with the worst slack time of the design are shown. You use HDL Analyst->Set Slack Margin to change the slack margin.

The critical-path commands do not calculate a single critical path. They filter out instances whose slack times are not too bad (as determined by the slack margin), then display the remaining, worst-slack instances, together with their connecting paths.

For example, if the worst slack time of your design is -10 ns and you set a slack margin of 4 ns, then the critical path commands display all instances with slack times between -6 ns and -10 ns.

See also:

- HDL Analyst Menu, on page 304
- HDL Analyst Command, on page 305
- Handling Negative Slack, on page 728 of the *User Guide*
- Analyzing Timing in Schematic Views, on page 722 of the *User Guide*

Examining Critical Path Schematics

Use successive filtering operations to examine different aspects of the critical path. After filtering, use View -> Back to return to the previous point, then filter differently. For example, you could use the command Isolate Paths to examine the cone of logic from a particular pin, then use the Back command to return to the previous display, then use Isolate Paths on a different pin to examine a different logic cone, and so on. in the *User Guide*

Copyright © 2013 Synopsys, Inc.

Also, the Show Context and Goto Net Driver commands are particularly useful after you have done some filtering. They let you get back to the original, unfiltered design, putting selected objects in context.

See also:

- Returning to The Unfiltered Schematic, on page 444
- Filtering and Flattening Schematics, on page 442

Copyright © 2013 Synopsys, Inc.

CHAPTER 5

Constraints

Constraints are used in the FPGA synthesis environment to achieve optimal design results. Timing constraints set performance goals, non-timing constraints (design constraints) guide the tool through optimizations that further enhance performance and physical constraints define regions and locations for placement-aware synthesis.

This chapter provides an overview of how constraints are handled in the FPGA synthesis environment.

- Constraint Types, on page 454
- Constraint Files, on page 455
- Timing Constraints, on page 457
- FDC Constraints, on page 460
- Methods for Creating Constraints, on page 461
- Constraint Translation, on page 464
- Constraint Checking, on page 464
- Database Object Search, on page 466
- Forward Annotation, on page 467
- Auto Constraints, on page 467

Copyright © 2013 Synopsys, Inc.

Constraint Types

One way to ensure the FPGA synthesis tools achieve the best quality of results for your design is to define proper constraints. In the FPGA environment, constraints can be categorized by the following types:

Type	Description
Timing	Performance constraints that guide the synthesis tools to achieve optimal results. Examples: clocks (create_clock), clock groups (set_clock_groups), and timing exceptions like multicycle and false paths (set_multicycle_path...) See *Timing Constraints*, on page 457 for information on defining these constraints.
Design	Additional design goals that enhance or guide tool optimizations. Examples: Attributes and directives (define_attribute, define_global_attribute), I/O standards (define_io_standard), and compile points (define_compile_point).
Physical	Constraints that drive physical synthesis. Examples: Region constraints for floorplanning (create_region).

The easiest way to specify constraints is through the SCOPE interface. The tool saves timing and design constraints to an FDC file that you add to your project.

See Also

Constraint Files, on page 455	Overview of constraint files
Timing Constraints, on page 457	Overview of timing constraint definitions and FDC file generation.
SCOPE Constraints Editor, on page 469	Information about automatic generation of timing and design constraints.
Chapter 14, *Timing Constraint Syntax*	Timing constraint syntax
Chapter 15, *FPGA Design Constraint Syntax*	Design constraint syntax

Copyright © 2013 Synopsys, Inc.

Constraint Files

The figure below shows the files used for specifying various types of constraints. The FDC file is the most important one and is the primary file for both timing and non-timing design constraints. The other constraint files are used for specific features or as input files to generate the FDC file, as described in *Timing Constraints*, on page 457. The figure also indicates the specific processes controlled by attributes and directives.

Copyright © 2013 Synopsys, Inc.

The table is a summary of the various kinds of constraint files.

File	Type	Common Commands	Comments
FDC	Timing constraints Design constraints	create_clock, set_multicycle_delay … define_attribute, define_io_standard …	Used for synthesis. Includes timing constraints that follow the Synopsys standard format as well as design constraints.
ADC	Timing constraints for timing analysis	create_clock, set_multicycle_delay …	Used with the stand-alone timing analyzer.
CDC	Design constraints	define_attribute define_global_attribute	Directives read in during the compilation phase of synthesis.
PDC	Translated Xilinx timing and physical constraints	create_clock, define_attribute syn_loc …	Used in projects for Physical Plus.
SDC (Synopsys Standard)	FPGA timing constraints	create_clock, set_clock_latency, set_false_path …	Use sdc2fdc to convert constraints to an FDC file so that they can be passed to the synthesis tools.
SDC (Legacy)	Legacy timing constraints and non-timing (or design) constraints	define_clock, define_false_path define_attribute, define_collection …	Use sdc2fdc to convert the constraints to an FDC file so that they can be passed to the synthesis tools.

Copyright © 2013 Synopsys, Inc.

Timing Constraints

The synthesis tools have supported different timing formats in the past, and this section describes some of the details of standardization:

- Legacy SDC and Synopsys Standard SDC, on page 457
- FDC File Generation, on page 458
- Timing Constraint Precedence in Mixed Constraint Designs, on page 459

Legacy SDC and Synopsys Standard SDC

Releases prior to G-2012.09 had two types of constraint files that could be used in a design project:

- Legacy "Synplify-style" timing constraints (define_clock, define_false_path...) saved to an sdc file. This file also included non-timing design constraints, like attributes and compile points.
- Synopsys standard timing constraints (create_clock, set_false_path…). These constraints were also saved to an sdc file, which only contained timing constraints. Non-timing constraints were in a separate sdc file. The tool used the two files together, drawing timing constraints from one and non-timing constraints from the other.

Starting with the G-2012.09 release, Synopsys standard timing constraint format has replaced the legacy-style constraint format, and a new FDC (FPGA design constraint) file consolidates both timing and design formats. As a result of these updates, there are some changes in the use model:

- Timing constraints in the legacy format are converted and included in an FDC file, which includes both timing and non-timing constraints. The file uses the Synopsys standard syntax for timing constraints (create_clock, set_multicyle_path...). The syntax for non-timing design constraints is unchanged (define_attribute, define_io_standard…).
- The SCOPE editor has been enhanced to support the timing constraint changes, so that new constraints can be entered correctly.
- For older designs, use the sdc2fdc command to do a one-time conversion.
- If you are are targeting Xilinx 7-series technologies and Vivado, legacy-style timing constraints are not supported. You must use the sdc2fdc command to convert any existing legacy-style constraint files.

Copyright © 2013 Synopsys, Inc.

FDC File Generation

The following figure is a simplified summary of constraint-file handling and the generation of fdc.

It is not required that you convert Synopsys standard sdc constraints as the figure implies, because they are already in the correct format. You could have a design with mixed constraints, with separate Synopsys standard sdc and fdc files. The disadvantage to keeping them in the standard sdc format is that you cannot view or edit the constraints through the SCOPE interface.

Timing Constraint Precedence in Mixed Constraint Designs

Your design could include timing constraints in a Synopsys standard sdc file and others in an fdc file. With mixed timing constraints in the same design, the following order of precedence applies:

- The tool reads the file order listed in the project file and any conflicting constraint overwrites a previous constraint.

With the legacy timing constraints, it is strongly recommended that you convert them to the `fdc` format, but even if you retain the old format in an existing design, they must be used alone and cannot be mixed in the same design as `fdc` or Synopsys standard timing `sdc` constraints. Specifically, do not specify timing constraints using mixed formats. For example, do not define clocks with define_clock and create_clock together in the same constraint file or multiple SDC/FDC files.

For the list of FPGA timing constraints (FDC) and their syntax, see *FPGA Timing Constraints*, on page 1100.

Copyright © 2013 Synopsys, Inc.

FDC Constraints

The FPGA design constraints (FDC) file contains constraints that the tool uses during synthesis. This FDC file includes both timing constraints and non-timing constraints in a single file.

- Timing constraints define performance targets to achieve optimal results. The constraints follow the Synopsys standard format, such as create_clock, set_input_delay, and set_false_path.
- Non-timing (or design constraints) define additional goals that help the tool optimize results. These constraints are unique to the FPGA synthesis tools and include constraints such as define_attribute, define_io_standard, and define_compile_point.

The recommended method to define constraints is to enter them in the SCOPE editor, and the tool automatically generates the appropriate syntax. If you define constraints manually, use the appropriate syntax for each type of constraint (timing or non-timing), as described above. See *Methods for Creating Constraints*, on page 461 for details on generating constraint files.

Prior to release G-2012.09, designs used timing constraints in either legacy Synplify-style format or Synopsys standard format. You must do a one-time conversion on any existing SDC files to convert them to FDC files using the following command:

```
% sdc2fdc
```

sdc2fdc converts constraints as follows:

For legacy Synplify-style timing constraints	Converts timing constraints to Synopsys standard format and saves them to an FDC file.
For Synopsys standard timing constraints	Preserves Synopsys standard format timing constraints and saves them to an FDC file.
For non-timing or design constraints	Preserves the syntax for these constraints and saves them to an FDC file.

Once defined, the FDC file can be added to your project. Double-click this file from the Project view to launch the SCOPE editor to view and/or modify your constraints. See *Converting SDC to FDC*, on page 171 for details on how to run sdc2fdc.

Copyright © 2013 Synopsys, Inc.

Methods for Creating Constraints

Constraints are passed to the synthesis environment in FDC files using Tcl command syntax.

New Designs

For new designs, you can specify constraints using any of the following methods:

Definition Method	Description
SCOPE Editor (fdc file)–Recommended	Use this method to specify constraints wherever possible. The SCOPE editor automatically generates fdc constraints with the right syntax. You can use it for most constraints. See Chapter 6, *SCOPE Constraints Editor*, for information how to use SCOPE to automatically generate constraint syntax. Access: File->New->FPGA Design Constraints …
Manually-Entered Text Editor (fdc File, all other constraint files)	You can manually enter constraints in a text file. Make sure to use the correct syntax for the timing and design commands. The SCOPE GUI includes a TCL View with an advanced text editor, where you can manually generate the constraint syntax. For a description of this view, see *TCL View*, on page 494. You can also open any constraint file in a text editor to modify it.
Source Code Attributes/Directives (HDL files, cdc file)	Directives must be entered in the source code because they affect the compiler. Do not include any other constraints in the source code, as this makes the source code less portable. In addition, you must recompile the design for the constraints to take effect. You can specify some compiler directives in a cdc file with the New-> Compiler Directives command. Attributes can be entered through the SCOPE interface, as they affect the mapper, not the compiler

Copyright © 2013 Synopsys, Inc.

Definition Method	Description
Automatic— First Pass	Enable the Auto Constrain button in the Project view to have the tool automatically generate constraints based on inferred clocks. See *Using Auto Constraints*, on page 739 in the *User Guide* for details. Use this method as a quick first pass to get an idea of what constraints can be set.

If there are multiple timing exception constraints on the same object, the software uses the guidelines described in *Conflict Resolution for Timing Exceptions*, on page 513 to determine the constraint that takes precedence.

See Also

To specify the correct syntax for the timing and design commands, see:

- Chapter 14, *Timing Constraint Syntax*
- Chapter 15, *FPGA Design Constraint Syntax*
- Chapter 16, *Attribute and Directive Syntax*

Copyright © 2013 Synopsys, Inc.

Existing Designs

The SCOPE editor in this release does not save constraints to SDC files. For designs prior to G-2012.09, it is recommended that you migrate your timing constraints to FDC format to take advantage of the tool's enhanced handling of these types of constraints. To migrate constraints, use the sdc2fdc command (see *Converting SDC to FDC*, on page 1711) on your sdc files.

Note: If you need to edit an SDC file, either use a text editor, or double-click the file to open the legacy SCOPE editor. For information on editing older SDC files, see *SCOPE User Interface (Legacy)*, on page 517 or *Synplify-Style Timing Constraints (Legacy)*, on page 1135.

See Also

To use the current SCOPE editor, see:

- Chapter 6, *SCOPE Constraints Editor*
- Chapter 4, *Specifying Constraints*

Copyright © 2013 Synopsys, Inc.

Constraint Translation

The tool includes a number of scripts for converting constraints to the correct format. The sdc2fdc script translates sdc files to fdc files. For constraints from vendor tools, you must first use the appropriate utility to translate the constraints to sdc, and then migrate them to fdc with the sdc2fdc script.

This table lists the translation scripts:

Vendor	Command/Utility	For details, see
Synopsys	xdc2fdc	xdc2fdc Command, on page 1020
Synopsys	sdc2fdc	sdc2fdc Tcl Shell Command, on page 988
Altera	qsf2sdc	Altera qsf2sdc Utility, on page 994 Altera SOPC2Syn Utility, on page 998 Altera qsf2syn Utility, on page 1003
Xilinx	ucf2sdc	Xilinx ucf2sdc Utility, on page 1004 Xilinx ise2syn Utility, on page 1017

For information about using these utilities, see:

- *Translating XDC Constraints to FDC*, on page 190
- *Converting SDC to FDC*, on page 171
- *Translating Altera QSF Constraints*, on page 193
- *Converting and Using Xilinx UCF Constraints*, on page 200

Constraint Checking

The synthesis tools include several features to help you debug and analyze design constraints. Use the constraint checker to check the syntax and applicability of the timing constraints in the project. The synthesis log file includes a timing report as well as detailed reports on the compiler, mapper, and resource usage information for the design. A stand-alone timing analyzer (STA) generates a customized timing report when you need more details

Copyright © 2013 Synopsys, Inc.

about specific pathes or want to modify constraints and analyze, without resynthesizing the design. The following sections provide more information about these features.

Constraint Checker

Check syntax and other pertinent information on your constraint files using Run->Constraint Check or the Check Constraints button in the SCOPE editor. This command generates a report that checks the syntax and applicability of the timing constraints that includes 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

See *Constraint Checking Report*, on page 629 for details.

Timing Constraint Report Files

The results of running constraint checking, synthesis, and stand-alone timing analysis are provided in reports that help you analyze constraints.

Use these files for additional timing constraint analysis:

File	Description
`_cck.rpt`	Lists the results of running the constraint checker (see *Constraint Checking Report*, on page 629).
`.ta`	Reports timing analysis results (see *Generating Custom Timing Reports with STA*, on page 729).
`.srr` or `.htm`	Reports post-synthesis timing results as part of the text or HTML log file (see *Timing Reports*, on page 615 and *Log File*, on page 608).

Copyright © 2013 Synopsys, Inc.

Database Object Search

To apply constraints, you have to search the database to find the appropriate objects. Sometimes you might want to search for and apply the same constraint to multiple objects. The FPGA tools provide some Tcl commands to facilitate the search for database objects:

Commands	Common Commands	Description
Find	Tcl Find, open_design...	Lets you search for design objects to form collections that can apply constraints to the group. See *Using Collections*, on page 161 and *Tcl find Command*, on page 1774.
Collections	define_collection, c_union...	Create, copy, evaluate, traverse, and filter collections. See *Using Collections*, on page 161 and *Collection Commands*, on page 1794 for more information.

Copyright © 2013 Synopsys, Inc.

Forward Annotation

The tool can automatically generate vendor-specific constraint files for forward annotation to the place-and-route tools when you enable the Write Vendor Constraints switch (on the Implementation Results tab) or use the -write_apr_constraint option of the set_option command.

Vendor	File Extension
Achronix	SCF
Altera	ACF SCF TCL
Lattice	*_SYNPLIFY.LPF SCF
Microsemi	_SDC.SDC
Xilinx	NCD NCF UCF XDC

For information about how forward annotation is handled for your target technology, refer to the appropriate vendor chapter of the *FPGA Synthesis Reference Manual*.

Auto Constraints

Auto constraints are automatically generated by the synthesis tool, however, these do not replace regular timing constraints in the normal synthesis flow. Auto constraints are intended as a quick first pass to evaluate the kind of timing constraints you need to set in your design.

To enable this feature and automatically generate register-to-register constraints, use the Auto Constrain option on the left panel of the Project view. For details, see *Using Auto Constraints*, on page 739 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

CHAPTER 6

SCOPE Constraints Editor

The SCOPE (Synthesis Constraints OPtimization Environment®) editor automatically generates syntax for synthesis constraints. Enter information in the SCOPE tabs, panels, columns, and pulldowns to define constraints and parameter values. You can also drag-and-drop objects from the HDL Analyst UI to populate values in the constraint fields.

This interface creates Tcl-format *Synopsys Standard timing constraints* and *Synplify-style design constraints* and saves the syntax to an FPGA design constraints (FDC) file that can automatically be added to your synthesis project. See *Constraint Types*, on page 454 for definitions of synthesis constraints.

Topics in this section include:

- SCOPE User Interface, on page 470
- SCOPE Tabs, on page 471
- Industry I/O Standards, on page 496
- Delay Path Timing Exceptions, on page 500
- Specifying From, To, and Through Points, on page 506
- Conflict Resolution for Timing Exceptions, on page 513
- SCOPE User Interface (Legacy), on page 517

Copyright © 2013 Synopsys, Inc.

SCOPE User Interface

The SCOPE editor contains a number of panels for creating and managing timing constraints and design attributes. This GUI offers the easiest way to create constraint files for your project. The syntax is saved to a file using an FDC extension and can be included in your design project.

From this editor, you specify timing constraints for clocks, ports, and nets as well as design constraints such as attributes, collections, and compile points. However, you cannot set black-box constraints from the SCOPE window.

To bring up the editor, use one of the following methods from the Project view:

- For a new file (the project file is open and the design is compiled):
  - Choose File->New-> FPGA Design Constraints; select FPGA Constraint File (SCOPE).
  - Click the SCOPE icon in the toolbar; select FPGA Constraint File (SCOPE).
- You can also open the editor using an existing constraint file. Double-click on the constraint file (FDC), or use File->Open, specifying the file type as FPGA Design Constraints File (`*.fdc`).

See Also:

- *Using the SCOPE Editor*, on page 130 in the *User Guide*.
- *SCOPE User Interface (Legacy)*, on page 517

SCOPE Tabs

Here is a summary of the constraints created through the SCOPE editor:

SCOPE Panel	See...
Clocks	Clocks, on page 472
Generated Clocks	Generated Clocks, on page 478
Collections	Collections, on page 480
Inputs/Outputs	Inputs/Outputs, on page 482
Registers	Registers, on page 484
Delay Paths	Delay Paths, on page 486
Attributes	Attributes, on page 488
I/O Standards	I/O Standards, on page 489
Compile Points	Compile Points, on page 491
TCL View	TCL View, on page 494

If you choose an object from a SCOPE pull-down menu, it has the appropriate prefix appended automatically. If you drag and drop aìn object from an RTL view, for example, make sure to add the prefix appropriate to the language used for the module. See *Verilog Naming Syntax*, on page 1157 and *VHDL Naming Syntax*, on page 1158 for details.

Copyright © 2013 Synopsys, Inc.

Clocks

You use the Clocks panel of the SCOPE spreadsheet to define a signal as a clock.

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	Uncertainty	Comment
1										
2										
3										
4										

Clocks

The Clocks panel includes the following options:

Field	Description
Name	Specifies the clock object name. Clocks can be defined on the following objects: • Pins • Ports • Nets For virtual clocks, the field must contain a unique name not associated with any port, pin, or net in the design.
Period	Specifies the clock period in nanoseconds. This is the minimum time over which the clock waveform repeats. The period must be greater than zero.
Waveform	Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time in the list is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise and then fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of period/2.
Add Delay	Specifies whether to add this delay to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you use this option, you must also specify the clock, and clocks with the same source must have different names.

Copyright © 2013 Synopsys, Inc.

Field	Description
Clock Group	Assigns clocks to asynchronous clock groups. The clock grouping is inclusionary (for example, clk2 and clk3 can each be related to clk1 without being related to each other). For details, see *Clock Groups*, on page 473.
Latency	Specifies the clock latency applied to clock ports and clock aliases. Applying the latency constraint on a port can be used to model the off-chip clock delays in a multichip environment. Clock latency can only: • Apply to clocks defined on input ports. • Be used for source latency. • Apply to port clock objects.
Uncertainty	Specifies the clock uncertainty (skew characteristics) of the specified clock networks. You can only apply latency to clock objects.

Clock Groups

Clock grouping is associative; two clocks can be asynchronous to each other but both can be synchronous with a third clock.

The SCOPE GUI prompts you for a clock group for each clock that you define. By default, the tool assigns all clocks to the default clock group. When you add a name that differs from the default clock group name, the clock is assigned its own clock group and is asynchronous to the default clock group as well as all other named clock groups.

This section presents scenarios for defining clocks and includes the following examples:

- *Example 1 – SCOPE Definition*
- *Example 2 – Equivalent Tcl Syntax*
- *Example 3 – Establish Clock Relationships*
- *Example 4 – Using a Single Group Option*
- *Example 5 – UCF Forward Annotation*
- *Example 6 – Legacy Clock Grouping*

Copyright © 2013 Synopsys, Inc.

Example 1 – SCOPE Definition

A design has three clocks, clk1, clk2, clk3. You want clk1 and clk2 to be in the same clock group—synchronous to each other but asynchronous to clk3. You can do this by adding a name in the Clock Group column, as shown below:

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency
1	☑	clk1	clk1	7		☐	group1	
2	☑	clk2	n:clk2	10		☐	group1	
3	☑	clk3	clk3	12		☐	<default>	
4								
5								
6								

This definition assigns clk1 and clk2 to clock group group1, synchronous to each other and asynchronous to clk3. This is the equivalent command appears in the text editor window:

```
set_clock_groups -derive -asynchronous -name {group1}
                -group { {c:clk1} {c:clk2} }
```

Example 2 – Equivalent Tcl Syntax

A design has three clocks: clk1, clk2, clk3. Use the following commands to set clk2 synchronous to clk3, but asynchronous to clk1:

```
set_clock_groups –asynchronous –group [get_clocks {clk3 clk2}]

set_clock_groups –asynchronous -group [get_clocks {clk1}]
```

Copyright © 2013 Synopsys, Inc.

Example 3 – Establish Clock Relationships

A design has the following clocks defined:

```
create_clock -name {clka} {p:clka} -period 10 -waveform {0 5.0}
create_clock -name {clkb} {p:clkb} -period 20 -waveform {0 10.0}
create_clock -name {my_sys} {p:sys_clk} -period 200 -waveform {0
100.0}
```

You want to define clka and clkb as asynchronous to each other and clka and clkb as synchronous to my_sys.

For the tool to establish these relationships, multiple -group options are needed in a single set_clock_groups command. Clocks defined by the first –group option are asynchronous to clocks in the subsequent –group option. Therefore, you would use the following syntax to establish the relationships described above:

```
set_clock_groups -asynchronous -group [get_clocks {clka}]
            -group [get_clocks {clkb}]
```

Example 4 – Using a Single Group Option

set_clock_groups has a unique behavior when a single –group option is specified in the command. For example, in the following constraint specification:

```
set_clock_groups  -asynchronous -name {default_clkgroup_0} -group
[get_clocks {clka my_sys}]

set_clock_groups  -asynchronous -name {default_clkgroup_1} -group
[get_clocks {clkb my_sys}]
```

The first statement assigns clka AND my_sys as asynchronous to clkb, and the second statement assigns clkb AND my_sys as asynchronous to clka. Therefore, with this specification, all three clocks are established as asynchronous to each other.

Example 5 – UCF Forward Annotation

To forward-annotate clock groups to a Xilinx technology UCF for the Xilinx ISE place and route tool, each clock must belong to only one clock group. Clock relationships in UCF must be represented in synthesis using the *single group* option method for one or more set_clock_groups constraints, but any given clock may be present in only one of the constraints. This is due to the

Copyright © 2013 Synopsys, Inc.

linear nature of clock groups in UCF in that, if two clocks in a design are defined asynchronous to each other, a third clock in the design can only be defined as synchronous to one of the first two clocks, or asynchronous to both. The third clock can never be synchronous to both of the first two clocks.

The SCOPE method of defining clock groups as show in *Example 1 – SCOPE Definition*, is specifically designed to make this linear type grouping the default and avoid issues with clock groups when going from the Synopsys standard clock grouping to Xilinx UCF clock grouping. The Synplify Xilinx mappers detect when the target forward annotation is UCF for Xilinx designs and issues an error if clock grouping in synthesis is not compatible with UCF. It is recommended that you use SCOPE to define your clocks and establish clock relationships when targeting Xilinx technologies for ISE place and route.

An example of the SCOPE generated SDC Standard syntax for clock grouping for a design with three clocks—clka, clkb, and my_sys—is shown below:

```
set_clock_groups -asynchronous -group [get_clocks {clka my_sys}]
set_clock_groups -asynchronous -group [get_clocks {clkb}]
```

OR

```
set_clock_groups -asynchronous -group [get_clocks {clkb my_sys}]
set_clock_groups -asynchronous -group [get_clocks {clka}]
```

Example 6 – Legacy Clock Grouping

This section shows how the legacy clock group definitions (Synplify-style timing constraints) are converted to the Synopsys standard timing syntax (FDC). Legacy clock grouping can be represented through Synopsys standard constraints, but the multi-grouping in the Synopsys standard constraints cannot be represented in legacy constraints.

For example, the following table shows legacy clock definitions and their translated FDC equivalents:

Copyright © 2013 Synopsys, Inc.

Legacy Definition	define_clock -name{clka}{p:clka}-period 10 -clockgroup default_clkgroup_0 define_clock -name {clkb}{p:clkb} -freq 150 -clockgroup default_clkgroup_1 define_clock -name {clkc} {p:clkc} -freq 200 -clockgroup default_clkgroup_1
FDC Definition	###==== BEGIN Clocks - (Populated from SCOPE tab, 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} create_clock -name {clkc} {p:clkc} -period 5.0 -waveform {0 2.5} set_clock_groups -derive -name default_clkgroup_0 -asynchronous -group {c:clka} set_clock_groups -derive -name default_clkgroup_1 -asynchronous -group {c:clkb c:clkc} ###==== END Clocks

The create_generated_clock constraints used in legacy SDC are preserved in FDC. The -derive option directs the create_generated_clock command to inherit the -source clock group. This behavior is unique to FDC and is an extension of the Synopsys SDC standard functionality.

See Also

For equivalent Tcl syntax, see the following sections:

- create_clock, on page 1102
- set_clock_groups, on page 1108
- set_clock_latency, on page 1111
- set_clock_uncertainty, on page 1114

For information about other SCOPE panels, see *SCOPE Tabs*, on page 471.

Copyright © 2013 Synopsys, Inc.

Generated Clocks

Use the Generated Clocks panel of the SCOPE spreadsheet to define a signal as a generated clock. The equivalent Tcl constraint is create_generated_clock; its syntax is described in create_generated_clock, on page 1104.

	Enable	Name	Source	Object	Master Clock	Generate Type	Generate Parameters	Generate Modifier	Modifier Parameters	Invert	Add	Comment
1												
2												
3												
4												

Generated Clocks

The Generated Clocks panel includes the following options:

Field	Description
Name	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the source.
Source	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.
Object	Generated clocks can be defined on the following objects: • Pins • Ports • Nets • Instances—where instances have only one output (for example, BUFGs)
Master Clock	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.

Copyright © 2013 Synopsys, Inc.

Field	Description
Generate Type	Specifies any of the following: edges – Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.
	divide_by – Specifies the frequency division factor. If the divide factor value is *2*, the generated clock period is twice as long as the master clock period.
	multiply_by – Specifies the frequency multiplication factor. If the multiply factor value is *3*, the generated clock period is one-third as long as the master clock period.
Generate Parameters	Specifies integers that define the type of generated clock.
Generate Modifier	Defines the secondary characteristics of the generated clock.
Modify Parameters	Defines modifier values of the generated clock.
Invert	Specifies whether to use invert – Inverts the generated clock signal (in the case of frequency multiplication and division).
Add	Either add this clock to the existing clock or overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also specify the clock and master clock. The clocks with the same source must have different names.

For more information about other SCOPE options, see *SCOPE Tabs*, on page 471.

Copyright © 2013 Synopsys, Inc.

Collections

The Collections tab allows you to set 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 of the *User Guide*.

Field	Description
Collection Name	Enter the collection name.
Command	Select a collection creation command from the drop-down menu. See *Collection Commands*, on page 480 for descriptions of the commands.
Command Arguments	Specify the Tcl syntax for the constraint you want to apply to the collection.
Comment	Enter comments that are included in the constraints file.

You can crossprobe the collection results to an HDL Analyst view. To do this, right-click in the SCOPE cell and select the option Select in Analyst.

Collection Commands

You can use the collection commands on collections or Tcl lists. Tcl lists can be just a single element long.

To...	Use this command...
Create a collection	set modules To create and save a collection, assign it to a variable. You can also use this command to create a collection from any combination of single elements, TCL lists and collections: set modules [define_collection {v:top} {v:cpu} $mycoll $mylist] Once you have created a collection, you can assign constraints to it in the SCOPE interface.
Copy a collection	set modules_copy $modules This copies the collection, so that any change to $modules does not affect $modules_copy.
Evaluate a collection	c_print This command returns all objects in a column format. Use this for visual inspection. c_list This command returns a Tcl list of objects. Use this to convert a collection to a list. You can manipulate a Tcl list with standard Tcl list commands.
Concatenate a list to a collection	c_union
Identify differences between lists or collections	c_diff Identifies differences between a list and a collection or between two or more collections. Use the -print option to display the results.
Identify objects common to a list and a collection	c_intersect Use the -print option to display the results.
Identify objects common to two or more collections	c_sub Use the -print option to display the results.
Identify objects that belong exclusively to only one list or collection	c_symdiff Use this to identify unique objects in a list and a collection, or two or more collections. Use the -print option to display the results.

For information about all SCOPE panels, see *SCOPE Tabs*, on page 471.

Copyright © 2013 Synopsys, Inc.

Inputs/Outputs

The Inputs/Outputs panel models the interface of the FPGA with the outside environment. You use it to specify delays outside the device.

	Enable	Delay Type	Port	Rise	Fall	Max	Min	Clock	Clock Fall	Add Delay	Value	Comment
1												
2												
3												
4												

Inputs/Outputs

The Inputs/Outputs panel includes the following options:

Field	Description
Delay Type	Specifies whether the delay is an input or output delay.
Port	Specifies the name of the port.
Rise	Specifies that the delay is relative to the rising transition on specified port. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.
Fall	Specifies that the delay is relative to the falling transition on specified port Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
Max	Specifies that the delay value is relative to the longest path. **Note:** The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.

Copyright © 2013 Synopsys, Inc.

Field	Description
Min	Specifies that the delay value is relative to the shortest path. **Note:** The synthesis tool does not optimize for hold time violations and only reports -min delay values in the `synlog/topLevel_fpga_mapper.srr_Min` timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.
Clock	Specifies the name of a clock for which the specified delay is applied. If you specify the clock fall, you must also specify the name of the clock.
Clock Fall	Specifies that the delay relative to the falling edge of the clock. For examples, see *Input Delays*, on page 483 and *Output Delays*, on page 484.
Add Delay	Specifies whether to add delay information to the existing input delay or overwrite the input delay. For examples, see *Input Delays*, on page 483 and *Output Delays*, on page 484.
Value	Specifies the delay path value.

Input Delays

Here is how this constraint applies for input delays:

- Clock Fall – The default is the rising edge or rising transition of a reference pin. If you specify clock fall, you must also specify the name of the clock.
- Add Delay – Use this option to capture information about multiple paths leading to an input port relative to different clocks or clock edges.

  For example, set_input_delay 5.0 -max -rise -clock phi1 {A} removes all maximum rise input delay from A, because the -add_delay option is not specified. Other input delays with different clocks or with -clock_fall are removed.

  In this example, the -add_delay option is specified as set_input_delay 5.0 -max -rise -clock phi1 -add_delay {A}. If there is an input maximum rise delay for A relative to clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is maximum rise input delay relative to a different clock or different edge of the same clock, it remains with the new delay.

Copyright © 2013 Synopsys, Inc.

Output Delays

Here is how this constraint applies for output delays:

- Clock Fall – If you specify clock fall, you must also specify the name of the clock.
- Add Delay – By using this option, you can capture information about multiple paths leading from an output port relative to different clocks or clock edges.

  For example, the set_output_delay 5.0 -max -rise -clock phi1 {OUT1} command removes all maximum rise output delays from OUT1, because the -add_delay option is not specified. Other output delays with a different clock or with the -clock_fall option are removed.

  In this example, the -add_delay option is specified: set_output_delay 5.0 -max -rise -clock phi1 -add_delay {Z}. If there is an output maximum rise delay for Z relative to the clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is a maximum rise output delay relative to a different clock or different edge of the same clock, it remains with the new delay.

See Also

For equivalent Tcl syntax, see:

- set_input_delay, on page 1120
- set_output_delay, on page 1128

For information about all SCOPE panels, see *SCOPE Tabs*, on page 471.

Registers

This panel lets the advanced user add delays to paths feeding into/out of registers, in order to further constrain critical paths. You use this constraint to speed up the paths feeding a register. See set_reg_input_delay, on page 1131, and set_reg_output_delay, on page 1132 for the equivalent Tcl commands.

Copyright © 2013 Synopsys, Inc.

The Registers SCOPE panel includes the following fields:

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Delay Type	(Required) Specifies whether the delay is an input or output delay.
Register	(Required) Specifies the name of the register. If you have initialized a compiled design, you can choose from the pull-down list.
Route	(Required) Improves the speed of the paths to or from the register by the given number of nanoseconds. The value shrinks the effective period for the constrained registers without affecting the clock period that is forward-annotated to the place-and-route tool.
Comment	Lets you enter comments that are included in the constraints file.

Copyright © 2013 Synopsys, Inc.

Delay Paths

Use the Delay Paths panel to define the timing exceptions.

The Path Delay panel includes the following options:

Field	Description
Delay Type	Specifies the type of delay path you want the synthesis tool to analyze. Choose one of the following types: • Multicycle • False • Max Delay • Reset Path • Datapath Only
From	Starting point for the path. From points define timing start points and can be defined for clocks (c:), registers (i:), top-level input or bi-directional ports (p:), or black box output pins (i:). For details, see the following: • Defining From/To/Through Points for Timing Exceptions • Object Naming Syntax, on page 1157
Through	Specifies the intermediate points for the timing exception. Intermediate points can be combinational nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). If you click the arrow in a column cell, you open the Product of Sums (POS) interface where you can set through constraints. For details, see the following: • Product of Sums Interface • Defining From/To/Through Points for Timing Exceptions • Object Naming Syntax, on page 1157

Copyright © 2013 Synopsys, Inc.

Field	Description
To	Ending point of the path. To points must be timing end points and can be defined for clocks (c:), registers (i:), top-level output or bi-directional ports (p:), or black box input pins (i:). For details, see the following: • Defining From/To/Through Points for Timing Exceptions • Object Naming Syntax, on page 1157
Max Delay	Specifies the maximum delay value for the specified path in nanoseconds.
Setup	Specifies the setup (maximum delay) calculations used for specified path.
Start/End	Used for multicycle paths with different start and end clocks. This option determines the clock period to use for the multiplicand in the calculation for clock distance. If you do not specify a start or end clock, the end clock is the default.
Cycles	Specifies the number of cycles required for the multicycle path.

See Also

- For equivalent Tcl syntax, see:
  - set_multicycle_path, on page 1125
  - set_false_path, on page 1118
  - set_max_delay, on page 1122
  - reset_path, on page 1106
  - set_datapathonly_delay, on page 1156
- For more information on timing exception constraints and how the tool resolves conflicts, see:
  - *Delay Path Timing Exceptions*, on page 500
  - *Conflict Resolution for Timing Exceptions*, on page 513
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 471.

Copyright © 2013 Synopsys, Inc.

Attributes

You can assign attributes directly in the editor.

Here are descriptions for the Attributes columns:

Column	Description
Enabled	(Required) Turn this on to enable the constraint.
Object Type	Specifies the type of object to which the attribute is assigned. Choose from the pull-down list, to filter the available choices in the Object field.
Object	(Required) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field.
Attribute	(Required) Specifies the attribute name. You can choose from a pull-down list that includes all available attributes for the specified technology. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the Synopsys FPGA synthesis tool filters the available choices in the Object field. You must select an attribute before entering a value. If a valid attribute does not appear in the pull-down list, simply type it in this field and then apply appropriate values.
Value	(Required) Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate.

Copyright © 2013 Synopsys, Inc.

Val Type	Specifies the kind of value for the attribute. For example, string or boolean.
Description	Contains a one-line description of the attribute.
Comment	Lets you enter comments about the attributes.

Enter the appropriate attributes and their values, by clicking in a cell and choosing from the pull-down menu.

To specify an object to which you want to assign an attribute, you may also drag-and-drop it from the RTL or Technology view into a cell in the Object column. After you have entered the attributes, save the constraint file and add it to your project.

See Also

- For more information on specifying attributes, see *How Attributes and Directives are Specified*, on page 1166.
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 471.

I/O Standards

You can specify a standard I/O pad type to use in the design. Define an I/O standard for any port appearing in the I/O Standards panel.

	Enabled	Port	Type	I/O Standard	DCI	DV2	Slew Rate	Drive Strength	Termination	Description
1	☑	<input default>	input	LVCMOS_15			fast	8	pullup	1.5 volt - C..
2	☑	<output default>	output							
3	☐	<bidir default>	bidir							
4	☐	resetn	input							

I/O Standards

Copyright © 2013 Synopsys, Inc.

Field	Description
Enabled	(Required) Turn this on to enable the constraint, or off to disable a previous constraint.
Port	(Required) Specifies the name of the port. If you have initialized a compiled design, you can select a port name from the pull-down list. The first two entries let you specify global input and output delays, which you can then override with additional constraints on individual ports.
Type	(Required) Specifies whether the delay is an input or output delay.
I/O Standard	Supported I/O standards by Synopsys FPGA products. See *Industry I/O Standards*, on page 496 for a description of the standards.
DCI (Xilinx) DV2 (Xilinx) Slew Rate Drive Strength Termination Power Schmitt	The values for these parameters are based on the selected I/O standard.
Description	Describes the selected I/O Standard.
Comment	Enter comments about an I/O standard.

See Also

- The Tcl equivalent of this constraint is define_io_standard.
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 471.

Compile Points

Use the Compile Points panel to specify compile points in your design, and to enable/disable them. This panel, available only if the device technology supports compile points, is used to define a top-level constraint file.

Copyright © 2013 Synopsys, Inc.

Here are the descriptions of the fields in the Compile Points panel.

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Module	(Required) Specifies the name of the compile-point module. You must specify a view module, with a v: prefix to identify the module as a view. For example: v:alu.
Type	(Required) Specifies the type of compile point: • locked (default) – no timing reoptimization is done on the compile point. The hierarchical interface is unchanged and an interface logic model is constructed for the compile point. • locked, partition – locked compile point, for which compile point information is forward annotated to the place and route tool. This mode provides place and route runtime advantages and allows for obtaining stable results for a completed design. • soft – compile point is included in the top-level synthesis, boundary optimizations can occur. • hard – compile point is included in the top-level synthesis, boundary optimizations can occur, however, the boundary remains unchanged. Although, the boundary is not modified, instances on both sides of the boundary can be modified using top-level constraints. • black_box – the compile point is treated as a black box. The software ignores the contents of the compile point that includes all instances in the module; only its ports exist for synthesis. Black box compile point modules only write port definitions to the netlist files. For details, see *Compile Point Types,* on page 1092 in the *User Guide*.
Comment	Lets you enter a comment about the compile point.

Constraints for Compile Points

You can set constraints at the top-level or for modules to be used as the compile points from the Current Design pull-down menu shown below. Use the Compile Points tab to select compile points and specify their types.

See Also

- The Tcl equivalent is define_compile_point.
- For more information on compile points and using the Compile Points panel, see *Synthesizing Compile Points*, on page 1109 in the *User Guide*.
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 471.

Copyright © 2013 Synopsys, Inc.

TCL View

The TCL View is an advanced text file editor for defining 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
  - Validate commands and command syntax
  - Identifies 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

For information on how to use this Tcl text editor, see *Using the TCL View of SCOPE GUI*, on page 141.

See Also

- For Tcl timing constraint syntax, see *FPGA Timing Constraints*, on page 1100.
- For Tcl design constraint syntax, see Chapter 15, *FPGA Design Constraint Syntax*.
- You can also use the SCOPE editor to set attributes. See *How Attributes and Directives are Specified*, on page 1166 for details.

Copyright © 2013 Synopsys, Inc.

Industry I/O Standards

The synthesis tool lets you specify a standard I/O pad type to use in your design. You can define an I/O standard for any port supported from the industry standard and proprietary I/O standards.

For industry I/O standards, see *Industry I/O Standards*, on page 497.

For vendor-specific I/O standards, see:

- Altera I/O Standards, on page 1867
- Lattice iCE65/iCE40 I/O Standards, on page 1976
- Microsemi I/O Standards, on page 2029
- Xilinx I/O Standards, on page 2067

Copyright © 2013 Synopsys, Inc.

Industry I/O Standards

The following table lists industry I/O standards.

I/O Standard	Description
AGP1X	Intel Corporation Accelerated Graphics Port
AGP2X	Intel Corporation Accelerated Graphics Port
BLVDS_25	Bus Differential Transceiver
CTT	Center Tap Terminated - EIA/JEDEC Standard JESD8-4
DIFF_HSTL_15_Class_I	1.5 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
DIFF_HSTL_15_Class_II	1.5 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
DIFF_HSTL_18_Class_I	1.8 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-9A
DIFF_HSTL_18_Class_II	1.8 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-9A
DIFF_SSTL_18_Class_II	1.8 volt - Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-6
DIFF_SSTL_2_Class_I	2.5 volt - Pseudo Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9A
DIFF_SSTL_2_Class_II	2.5 volt - Pseudo Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9A
GTL	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+	Gunning Transceiver Logic Plus
GTL25	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+25	Gunning Transceiver Logic Plus
GTL33	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+33	Gunning Transceiver Logic Plus

Copyright © 2013 Synopsys, Inc.

I/O Standard	Description
HSTL_12	1.2 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_15_Class_II	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_I	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_II	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_III	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_IV	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_I	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_II	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_III	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_IV	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HyperTransport	2.5 volt - Hypertransport - HyperTransport Consortium

Copyright © 2013 Synopsys, Inc.

I/O Standard	Description
LVCMOS_12	1.2 volt - EIA/JEDEC Standard JESD8-16
LVCMOS_15	1.5 volt - EIA/JEDEC Standard JESD8-7
LVCMOS_18	1.8 volt - EIA/JEDEC Standard JESD8-7
LVCMOS_25	2.5 volt - EIA/JEDEC Standard JESD8-5
LVCMOS_33	3.3 volt CMOS - EIA/JEDEC Standard JESD8-B
LVCMOS_5	5.0 volt CMOS
LVDS	Differential Transceiver - ANSI/TIA/EIA-644-95
LVDSEXT_25	Differential Transceiver
LVPECL	Differential Transceiver - EIA/JEDEC Standard JESD8-2
LVTTL	3.3 volt TTL - EIA/JEDEC Standard JESD8-B
MINI_LVDS	Mini Differential Transceiver
PCI33	3.3 volt PCI 33MHz - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCI66	3.3 volt PCI 66MHz - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCI-X_133	3.3 volt PCI-X - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCML	3.3 volt - PCML
PCML_12	1.2 volt - PCML
PCML_14	1.4 volt - PCML
PCML_15	1.5 volt - PCML
PCML_25	2.5 volt - PCML
RSDS	Reduced Swing Differential Signalling
SSTL_18_Class_I	1.8 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-15
SSTL_18_Class_II	1.8 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-15
SSTL_2_Class_I	2.5 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9B
SSTL_2_Class_II	2.5 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9B
SSTL_3_Class_I	3.3 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-8
SSTL_3_Class_II	3.3 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-8
ULVDS_25	Differential Transceiver

Copyright © 2013 Synopsys, Inc.

Delay Path Timing Exceptions

For details about the following path types, see:

- Multicycle Paths, on page 500
- False Paths, on page 503

Multicycle Paths

Multicycle paths lets you specify paths with multiple clock cycles. The following table defines the parameters for this constraint. For the equivalent Tcl constraints, see set_multicycle_path, on page 1125. This section describes the following:

- Multi-cycle Path with Different Start and End Clocks, on page 500
- Multicycle Path Examples, on page 501

Multi-cycle Path with Different Start and End Clocks

The start/end option determines the clock period to use for the multiplicand in the calculation for required time. The following table describes the behavior of the multi-cycle path constraint using different start and end clocks. In all equations, *n* is number of clock cycles, and *clock_distance* is the default, single-cycle relationship between clocks that is calculated by the tool.

Basic required time for a multi-cycle path	clock_distance + [(n-1) * end_clock_period]
Required time with no end clock defined	clock_distance + [(n-1) * global_period]
Required time with -start option defined	clock_distance + [(n-1) * start_clock_period]
Required time with no start clock defined	clock_distance + [(n-1) * global_period]

Copyright © 2013 Synopsys, Inc.

If you do not specify a start or end option, by default the end clock is used for the constraint. Here is an example:

Multicycle Path Examples

Multicycle Path Example 1

If you apply a multicycle path constraint from D1 to D2, the allowed time is #cycles x normal time between D1 and D2. In the following figure, CLK1 has a period of 10 ns. The data in this path has only one clock cycle before it must reach D2. To allow more time for the signal to complete this path, add a multiple-cycle constraint that specifies two clock cycles (10 x 2 or 20 ns) for the data to reach D2.

Copyright © 2013 Synopsys, Inc.

Multicycle Path Example 2

The design has a multiplier that multiplies signal_a with signal_b and puts the result into signal_c. Assume that signal_a and signal_b are outputs of registers register_a and register_b, respectively. The RTL view for this example is shown below. On clock cycle 1, a state machine enables an input enable signal to load signal_a into register_a and signal_b into register_b. At the beginning of clock cycle 2, the multiply begins. After two clock cycles, the state machine enables an output_enable signal on clock cycle 3 to load the result of the multiplication (signal_c) into an output register (register_c).

The design frequency goal is 50 MHz (20 ns) and the multiply function takes 35 ns, but it is given 2 clock cycles. After optimization, this 35 ns path is normally reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To avoid reporting the paths as timing violations, use the SCOPE window to set 2-cycle constraints (From column) on register_a and register_b, or include the following in the timing constraint file:

```
# Paths from register_a use 2 clock cycles
set_multicycle_path -from register_a 2

# Paths from register_b use 2 clock cycles
set_multicycle_path -from register_b 2
```

Alternatively, you can specify a 2-cycle SCOPE constraint (To column) on register_c, or add the following to the constraint file:

```
# Paths to register_c use 2 clock cycles
set_multicycle_path -to register_c 2
```

Copyright © 2013 Synopsys, Inc.

False Paths

You use the Delay Paths constraint to specify clock paths that you want the synthesis tool to ignore during timing analysis and assign low (or no) priority during optimization. The equivalent Tcl constraint is described in set_false_path, on page 1118.

This section describes the following:

- Types of False Paths, on page 503
- Priority of False Path Constraints, on page 504

Types of False Paths

A false path is a path that is not important for timing analysis. There are two types of false paths:

- Architectural false paths

  These are false paths that the designer is aware of, like an external reset signal that feeds internal registers but which is synchronized with the clock. The following example shows an architectural false path where the primary input x is always 1, but which is not optimized because the software does not optimize away primary inputs.

- Code-introduced false paths

  These are false paths that you identify after analyzing the schematic.

Copyright © 2013 Synopsys, Inc.

Priority of False Path Constraints

False path constraints can be either explicit or *implicit*, and the priority of the constraint depends on the type of constraint it is.

- An explicit false path constraint is one that you apply to a path using the Delay Paths pane of the SCOPE GUI, or the following Tcl syntax:

  set_false_path {-from *point***} | {-to** *point***} | {-through** *point***}**

  This type of false path constraint has the highest priority of any of the types of constraints you can place on a path. Any path containing an explicit false path constraint is ignored by the software, even if you place a different type of constraint on the same path.

- Lower-priority false path constraints are those that the software automatically applies as a result of any of the following actions:
  - You assign clocks to different groups (Clocks pane of SCOPE GUI).
  - You assign an implicit false path (by selecting the false option in the Delay (ns) column of the SCOPE Clock to Clock panel). (This condition applies for legacy timing constraints.)
  - You disable the Use clock period for unconstrained IO option (Project -> Implementation Options->Constraints).

  Implicit false path constraints are overridden by any subsequent constraints you place on a path. For example, if you assign two clocks to different clock groups, then place a maximum delay constraint on a path that goes through both clocks, the delay constraint has priority.

False Path Constraint Examples

In this example, the design frequency goal is 50 MHz (20ns) and the path from register_a to register_c is a false path with a large delay of 35 ns. After optimization, this 35 ns path is normally reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To lower the priority of this path during optimization, define it as a false path. You can do this in many ways:

- If all paths from register_a to any register or output pins are not timing-critical, then add a false path constraint to register_a in the SCOPE interface (From), or put the following line in the timing constraint file:

Copyright © 2013 Synopsys, Inc.

```
#Paths from register_a are ignored
set_false_path -from {i:register_a}
```

- If all paths to register_c are not timing-critical, then add a false path constraint to register_c in the SCOPE interface (To), or include the following line in the timing constraint file:

```
#Paths to register_c are ignored
set_false_path -to {i:register_c}
```

- If only the paths between register_a and register_c are not timing-critical, add a From/To constraint to the registers in the SCOPE interface (From and To), or include the following line in the timing constraint file:

```
#Paths to register_c are ignored
set_false_path -from {i:register_a} -to {i:register_c}
```


Copyright © 2013 Synopsys, Inc.

Specifying From, To, and Through Points

The following section describes from, to, and through points for timing exceptions specified by the multicycle paths, false paths, and max delay paths constraints.

- Timing Exceptions Object Types, on page 506
- From/To Points, on page 506
- Through Points, on page 508
- Product of Sums Interface, on page 509
- Clocks as From/To Points, on page 511

Timing Exceptions Object Types

Timing exceptions must contain the type of object in the constraint specification. You must explicitly specify an object type, n: for a net, or i: for an instance, in the instance name parameter of all timing exceptions. For example:

```
set_multicycle_path -from {i:inst2.lowreg_output[7]}
   -to {i:inst1.DATA0[7]} 2
```

If you use the SCOPE GUI to specify timing exceptions, it automatically attaches the object type qualifier to the object name.

From/To Points

From specifies the starting point for the timing exception. To specifies the ending point for the timing exception. When you specify an object, use the appropriate prefix (see *syn_black_box*, on page 1235) to avoid confusion. The following table lists the objects that can serve as starting and ending points:

From Points	To Points
Clocks. See *Clocks as From/To Points*, on page 511 for more information.	Clocks. See *Clocks as From/To Points*, on page 511 for more information.
Registers	Registers
Top-level input or bi-directional ports	Top-level output or bi-directional ports
Instantiated library primitive cells (gate cells)	Instantiated library primitive cells (gate cells)
Black box outputs	Black box inputs

You can specify multiple from points in a single exception. This is most common when specifying exceptions that apply to all the bits of a bus. For example, you can specify constraints From A[0:15] to B – in this case, there is an exception, starting at any of the bits of A and ending on B.

Similarly, you can specify multiple to points in a single exception. If you specify both multiple starting points and multiple ending points such as From A[0:15] to B[0:15], there is actually an exception from any start point to any end point. In this case, the exception applies to all 16 * 16 = 256 combinations of start/end points.

Copyright © 2013 Synopsys, Inc.

Through Points

Through points are *limited to nets*; however, there are many ways to specify these constraints.

- Single Point
- Single List of Points
- Multiple Through Points
- Multiple Through Lists

You define these constraints in the appropriate SCOPE panels, or in the POS GUI (see *Product of Sums Interface*, on page 509). When a port and net have the same name, preface the name of the through point with n: for nets, t: for hierarchical ports, and p: for top-level ports. For example n:regs_mem[2] or t:dmux.bdpol. The n: prefix must be specified to identify nets; otherwise, the associated timing constraint will not be applied for valid nets.

Single Point

You can specify a single through point. In this case, the constraint is applied to any path that passes through regs_mem[2]:

```
set_false_path -through regs_mem[2]
```

Single List of Points

If you specify a list of through points, the through option behaves as an OR function and applies to any path that passes through any of the points in the list. In the following example, the constraint is applied to any path through regs_mem[2] OR prgcntr.pc[7] OR dmux.alub[0] with a maximum delay value of 5 ns (-max 5):

```
set_max_delay
-through {regs_mem[2], prgcntr.pc[7], dmux.alub[0]} 5
```

Copyright © 2013 Synopsys, Inc.

Multiple Through Points

You can specify multiple points for the same constraint by preceding each point with the -through option. In the following example, the constraint operates as an AND function and applies to paths through regs_mem[2] AND prgcntr.pc[7] AND dmux.alub[0]:

```
set_max_delay
-through regs_mem[2]
-through prgcntr.pc[7]
-through dmux.alub[0] 5
```

Multiple Through Lists

If you specify multiple -through lists, the constraint is applied as an AND/OR function and is applied to the paths through all points in the lists. The following constraint applies to all paths that pass through {A_1 or A_2 or...A_n} AND {B_1 or B_2 or B_3}:

```
set_false_path -through {A1 A2...An} -through {B1 B2 B3}
```

In this example,

```
set_multicycle_path
-through {net1, net2}
-through {net3, net4} 2
```

all paths that pass through the following nets are constrained at 2 clock cycles:

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

Product of Sums Interface

You can use the SCOPE GUI to format -through points for nets with multicycle path, false path, and max delay path constraints in the Product of Sums (POS) interface of the SCOPE editor. You can also manually specify constraints that use the -through option. For more information, see *Defining From/To/Through Points for Timing Exceptions*, on page 146 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

The POS interface is accessible by clicking the arrow in a Through column cell in the following SCOPE panels:

- Multi-Cycle Paths
- False Paths
- Delay Paths

Field	Description
Prod 1, 2, etc.	Type the first net name in a cell in a Prod row, or drag the net from a HDL Analyst view into the cell. Repeat this step along the same row, adding other nets in the Sum columns. The nets in each row form an OR list.
Sum 1, 2, etc.	Type the first net name in the first cell in a Sum column, or drag the net from a HDL Analyst view into the cell. Repeat this step down the same Sum column. The nets in each column form an AND list.
Drag and Drop Goes	Along Row - places objects in multiple Sum columns, utilizing only one Prod row. Down Column - places objects in multiple Prod rows, utilizing only one `Sum` column.
Drag and Drop	Inserts New Cells - New cells are created when dragging and dropping nets. Overwrites Cells - Existing cells are overwritten when dragging and dropping nets.
Save/Cancel	Saves or cancels your session.

Copyright © 2013 Synopsys, Inc.

Clocks as From/To Points

You can specify clocks as from/to points in your timing exception constraints. Here is the syntax:

set_*timing_exception* **-from** | **-to { c:***clock_name* [**:***edge*] **}**

where

- *timing_exception* is one of the following constraint types: multicycle path, false path, or max delay
- **c:***clock_name***:***edge* is the name of the clock and clock edge (r or f). If you do not specify a clock edge, by default both edges are used.

See the following sections for details and examples on each timing exception.

Multicycle Path Clock Points

When you specify a clock as a from or to point, the multicycle path constraint applies to all registers clocked by the specified clock.

The following constraint allows two clock periods for all paths from the rising edge of the flip-flops clocked by clk1:

```
set_multicycle_path -from {c:clk1:r} 2
```

You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint allows two clock periods for all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_multicycle_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]} 2
```


Copyright © 2013 Synopsys, Inc.

False Path Clock Points

When you specify a clock as a from or to point, the false path constraint is set on all registers clocked by the specified clock. False paths are ignored by the timing analyzer. The following constraint disables all paths from the rising edge of the flip-flops clocked by clk1:

```
set_false_path -from {c:clk1:r}
```

You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint disables all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net.

```
set_false_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]}
```

Path Delay Clock Points

When you specify a clock as a from or to point for the path delay constraint, the constraint is set on all paths of the registers clocked by the specified clock. This constraint sets a max delay of 2 ns on all paths to the falling edge of the flip-flops clocked by clk1:

```
set_max_delay -to {c:clk1:f} 2
```

You cannot specify a clock as a through point, but you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The next constraint sets a max delay of 0.2 ns on all paths from the rising edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_max_delay -from {c:clk1:r} -through (n:MYINST.mybus2[9]} .2
```

Copyright © 2013 Synopsys, Inc.

Conflict Resolution for Timing Exceptions

The term *timing exceptions* refers to the false path, max path delay, and multicycle path timing constraints. When the tool encounters conflicts in the way timing exceptions are specified through the constraint file, the software uses a set priority to resolve these conflicts. Conflict resolution is categorized into four levels, meaning that there are four different tiers at which conflicting constraints can occur, with one being the highest. The table below summarizes conflict resolution for constraints. The sections following the table provide more details on how conflicts can occur and examples of how they are resolved.

Conflict Level	Constraint Conflict	Priority	For Details, see ..
1	Different timing exceptions set on the same object.	1 – False Path 2 – Path Delay 3 – Multi-cycle Path	*Conflicting Timing Exceptions*, on page 514.
2	Timing exceptions of the same constraint type, using different semantics (from/to/through).	1 – From 2 – To 3 – Through	*Same Constraint Type with Different Semantics*, on page 515.
3	Timing exceptions of the same constraint type using the same semantic, but set on different objects.	1 – Ports/Instances/Pins 2 – Clocks	*Same Constraint and Semantics with Different Objects*, on page 516.
4	Identical timing constraints, except constraint values differ.	Tightest, or most constricting constraint.	*Identical Constraints with Different Values*, on page 516.

In addition to the four levels of conflict resolution for timing exceptions, there are priorities for the way the tool handles multiple I/O delays set on the same port and implicit and explicit false path constraints. For information on resolving these types of conflicts, see *Priority of Multiple I/O Constraints*, on page 530 and *Priority of False Path Constraints*, on page 504.

Copyright © 2013 Synopsys, Inc.

Conflicting Timing Exceptions

The first (and highest) level of resolution occurs when timing exceptions—false paths, max path delay, or multicycle path constraints—conflict with each other. The tool follows this priority for applying timing exceptions:

1. False Path
2. Path Delay
3. Multicycle Path

For example:

```
set_false_path -from {c:C1:r}
set_max_delay -from {i:A} -to {i:B} 10
set_multicycle_path -from {i:A} -to {i:B} 2
```

These constraints are conflicting because the path from A to B has three different constraints set on it. When the tool encounters this type of conflict, the false path constraint is honored. Because it has the highest priority of all timing exceptions, set_false_path is applied and the other timing exceptions are ignored.

Copyright © 2013 Synopsys, Inc.

Same Constraint Type with Different Semantics

The second level of resolution occurs when conflicts between timing exceptions that are of the same constraint type, use different semantics (from/to/through). The priority for these constraints is as follows:

1. From
2. To
3. Through

If there are two multicycle constraints set on the same path, one specifying a from point and the other specifying a to point, the constraint using -from takes precedence, as in the following example.

```
set_multicycle_path -from {i:A} 3
set_multicycle_path -to {i:B} 2
```

In this case, the tool uses:

```
set_multicycle_path -from {i:A} 3
```

The other constraint is ignored even though it sets a tighter constraint.

Copyright © 2013 Synopsys, Inc.

Same Constraint and Semantics with Different Objects

The third level resolves timing exceptions of the same constraint type that use the same semantic, but are set on different objects. The priority for design objects is as follows:

1. Ports/Instances/Pins
2. Clocks

If the same constraints are set on different objects, the tool ignores the constraint set on the clock for that path.

```
set_multicycle_path -from {i:mac1.datax[0]} -start 4
set_multicycle_path -from {c:clk1:r} 2
```

In the example above, the tool uses the first constraint set on the instance and ignores the constraint set on the clock from i:mac1.datax[0], even though the clock constraint is tighter.

For details on how the tool prioritizes multiple I/O delays set on the same port or implicit and explicit false path constraints, see *Priority of False Path Constraints*, on page 504 and *Priority of Multiple I/O Constraints*, on page 530.

Identical Constraints with Different Values

Where timing constraints are identical except for the constraint value, the tightest or most constricting constraint takes precedence. In the following example, the tool uses the constraint specifying two clock cycles:

```
set_multicycle_path -from {i:special_regs.trisa[7:0]} 2
set_multicycle_path -from {i:special_regs.trisa[7:0]} 3
```

Copyright © 2013 Synopsys, Inc.

SCOPE User Interface (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 want to edit your existing SDC file, to open the legacy SCOPE editor, double-click on your constraint file in the Project view.

For the Tcl command syntax created by the Legacy SCOPE editor, see:

- *Synplify-Style Timing Constraints (Legacy)*, on page 1135
- *FPGA Design Constraint Syntax*, on page 1161
- *How Attributes and Directives are Specified*, on page 1166

Copyright © 2013 Synopsys, Inc.

SCOPE Tabs (Legacy)

This section contains a description of the constraint panels in the SCOPE interface, and details about the constraints you can enter that are saved to an SDC file. SCOPE timing and design constraints include:

SCOPE Panel	See...
Clocks	Clocks (Legacy), on page 519
Clock to Clock	Clock to Clock (Legacy), on page 525
Collections	Collections, on page 480
Inputs/Outputs	Inputs/Outputs (Legacy), on page 527
Registers	Registers (Legacy), on page 531
Delay Paths	Delay Paths (Legacy), on page 532
Attributes	Attributes, on page 488
I/O Standards	I/O Standards, on page 489
Compile Points	Compile Points, on page 491
Other	Other (Legacy), on page 534

See Also:

- For the Tcl constraint syntax, see *Timing Constraint Summary (Legacy)*, on page 519
- For design constraints, see Chapter 15, *FPGA Design Constraint Syntax*.

Timing Constraint Summary (Legacy)

The following table shows the correlation between each SCOPE panel and the Tcl constraint syntax it creates:

SCOPE Panel/GUI	Tcl .sdc File
Clocks (Legacy)	define_clock
Clock to Clock (Legacy)	define_clock_delay
Delay Paths (Legacy)	define_false_path
Delay Paths (Legacy)	define_multicycle_path
Delay Paths (Legacy)	define_path_delay
Inputs/Outputs (Legacy)	define_input_delay
Inputs/Outputs (Legacy)	define_output_delay
Registers (Legacy)	define_reg_input_delay
Registers (Legacy)	define_reg_output_delay
Frequency in the Project view Implementation Options -> Constraints Panel	set_option -frequency

Clocks (Legacy)

You use the Clocks panel of the SCOPE spreadsheet to define a signal as a clock. The equivalent Tcl constraint is define_clock; its syntax is described in define_clock, on page 1136.

For more information on using the Clocks panel, see *Defining Clocks*, on page 179 of the *User Guide*. For information about setting clock constraints automatically in the Synplify Pro tool, see *Using Auto Constraints*, on page 739 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Clocks Panel Description

The Clocks panel includes the following fields:

SCOPE Cell	Description
Enabled	Turn this on to enable the constraint. Turn it off to disable an existing constraint.
Clock Object	(Required) Specifies the clock object name. Clocks *can* be defined on the following objects: • Top-level input ports • Nets • Output pins of instantiated cells For Xilinx technologies, see *Forward-annotation of Port Names in EDIF*, on page 2117 for details about specifying clock constraints on an instance. Clocks *cannot* be defined on the following objects: • Top-level output ports • Input pins of instantiated gates • Pins of inferred instances You can also assign a clock alias name to be used in the timing reports. See the Clock Alias option, below. For virtual clocks, the field must contain a unique name not associated with any port or instance in the design.
Clock Alias	Specifies a name for the clock if you want to use a name other than the object name. This alias name is used in the timing reports for the clock.
Frequency (MHz)	(Required) Specifies the clock frequency in MHz. If you fill in this field and click in the Period field, the Period value is filled in automatically. For frequencies other than that implied by the clock pin, refer to syn_reference_clock, on page 1490.
Period (ns)	(Required) Specifies the clock period in nanoseconds. If you fill in this field and click in the Frequency field, the frequency is filled in automatically.

Copyright © 2013 Synopsys, Inc.

SCOPE Cell	Description
Clock Group	Assigns a clock to a clock group. Clocks in the same clock group are related. • By default, the software assigns each clock to a group called default_clkgroup_*n*, where *n* is a sequential number. • The Synplify synthesis tool calculates the relationship between clocks in the same clock group and analyzes all paths between them. Paths between clocks in different groups are ignored (considered false paths). See *Clock Groups,* on page 522 for more information.
Rise At (ns) Fall At (ns)	Specifies non-default rising and falling edges for clocks.By default, the tool assumes that the clock is a 50% duty cycle clock, with rising edge at 0 and falling edge at Period/2. See *Rise and Fall Constraints,* on page 523 for details. Setting rise/fall calculates the duty cycle automatically.
Duty Cycle (%)	Specifies the clock duty cycle as a percentage of the clock period. If you have a duty cycle that is not 50% (the default), specify the rising and falling edge values (Rise At/Fall At) instead of Duty Cycle, and the duty cycle value is calculated automatically. There is no corresponding Tcl command option.
Route (ns)	Improves the path delays of registers controlled by the clock. The value shrinks the effective period for synthesis, without affecting the clock period that is forward-annotated to the place-and-route tool. This is an advanced user option. Before you use this option, evaluate the path delays on individual registers in the optimization timing report and try to improve the delays only on the registers that need them (Registers panel). See *Route Option,* on page 524 for a detailed explanation of this option.
Virtual Clock	Designates the specified clock as a virtual clock. It lets you specify arrival and required times on top level ports that are enabled by clocks external to the chip (or block) that you are synthesizing. The clock name can be a unique name not associated with any port or instance in the synthesized design.
Comment	Lets you enter comments that are included in the constraints file.

Copyright © 2013 Synopsys, Inc.

Clock Groups

The timing engine uses clock groups to analyze and optimize the design. It assumes that clocks in the same clock group are synchronized with each other and treats them as related clocks. Typically, clocks in a clock group are derived from the same base clock. The timing analyzer automatically calculates the relationships between the related clocks in a clock group and analyzes all paths between them. For paths between clocks of the same group, the timing engine calculates the time available based on the period of the two clocks. If you want to override the calculation, set a clock delay on the Clock to Clock tab (define_clock_delay).

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

The following figure shows how the timing analyzer calculates worst case edge-to-edge timing between all possible transitions of the two related clocks. In this example, CLK1 has a period of 5 ns and CLK2 has a period of 10 ns.

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

By default, all clocks in a design are assigned to separate clock groups. For most accurate results, re-assign clocks explicitly so that related clocks are in the same group, using the Clock Group field in the SCOPE spreadsheet.

For example, if your design has three clocks (clk1, clk2, and clk3), by default they are each assigned to different clock groups. If clk1 and clk3 are related, assign them to the same clock group so that paths between the two clocks are analyzed and optimized. For timing optimization, the software treats them as related clocks. Keep clk2 in a separate group, as it is unrelated to the other two clocks. The timing analyzer now only analyzes the relationship between clk1 and clk3. Paths between clk1 and clk2, and paths between clk3 and clk2, are considered false paths and are ignored.

Rise and Fall Constraints

The synthesis tool assumes an ideal clock network. By default, the constraints assume a 50% duty cycle clock with the rising edge at 0 and the falling edge at Period/2.

The synthesis tool computes relationships between the source clock and destination clock on a path by using the Rise At and Fall At numbers. When you enter or change these values, the Duty Cycle is calculated or recalculated automatically. The timing engine can calculate the rise/fall values from the Duty Cycle, but entering the rise/fall values directly is more accurate.

To understand how the relationships between source and destination clocks are computed using the Rise At and Fall At values, consider the following example.

	Enabled	Clock Object	Clock Alias	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock
1	☑	CLK1		100	10	default_clkgroup_2					☐
2	☑	CLK3		50	20	clkgroup_2	0	12	60		☐
3	☑	CLK4		50	20	default_clkgroup_1					☑

Clocks | Clock to Clock | Collections | Inputs/Outputs | Registers | Delay Paths | Attributes | I/O Standards | Compile Points

Copyright © 2013 Synopsys, Inc.

In this example:

- CLK1 is a clock with a period of 10 ns (100 MHz) in clock group default_clkgroup_2. Since none of the other fields are specified, this is a 50% duty cycle clock rising at 0 and falling at 5.

- CLK3 is a clock with a period of 20 ns (50 MHz) in clkgroup_2. This means all paths between CLK3 and CLK1 are automatically treated as false paths. In addition, CLK3 has a Rise At value of 0 and a Fall At value of 12, which means it has a 60% duty cycle.

- CLK4 is a virtual clock. This means that there can be no port or instance named CLK4 in this design. However, there may be top-level ports on the chip which are clocked by CLK4 outside the chip. Input arrival times and output required times for such ports can be specified relative to CLK4.

Check the Performance Summary for warnings about inferred clocks; an inferred clock in the summary may mean a declared clock is missing. The Performance Summary can also identify the alias names of derived clocks, for which timing exceptions (false path, multicycle path, or path delay) that target the derived clocks are needed.

Route Option

Use the Route (-route) option if you do not meet timing goals because the routing delay after placement and routing exceeds the delay predicted by the synthesis tool. The delay reported on an instance along a path from an input port is the amount of time by which the clock frequency goal is exceeded (positive slack), or not met (negative slack) because of the input delay. You can view slack times in the timing report section of the log file, or with the HDL Analyst tool.

Copyright © 2013 Synopsys, Inc.

Rerun synthesis with this option to include the actual route delay from place-and-route results, so that the tool can meet the required clock frequency. Using the Route option is equivalent to putting a register delay (set_reg_input delay or define_reg_input_delay) on all registers controlled by that clock.

Assume the frequency goal is 50 MHz (clock period goal of 20.0 ns). If the synthesis timing reports show the frequency goal is met, but the detailed timing report from placement and routing shows that the actual clock period is 21.8 ns because of paths through input_a, you can either tighten the constraints in the place-and-route tool, or you can rerun synthesis with a -route 1.8 option added to the define_input_delay constraint.

```
# In this example, input_a has a 10.0 ns input delay.
# Rerun synthesis with an added 1.8 ns constraint, to
# improve accuracy and match post-place-and-route results.

define_input_delay {input_a} 10.0 -route 1.8
```

Using this option adds 1.8 ns to the route calculations for paths to input_a and improves timing by 1.8 ns.

Clock to Clock (Legacy)

Defines the time available from data signals on all paths between the specified clock edges; only specified clock edge combinations are affected. For example, if you specify a new relationship for the rise-to-rise clock edge, other relationships are not affected. By default, the software automatically calculates clock delay based on the clock parameters defined in the Clock pane of the SCOPE UI. If you define a clock-to-clock constraint, this delay value overrides any automatic calculations made by the software and is reflected in the values shown in the Clock Relationships section of the timing report.

For the equivalent Tcl syntax, see define_clock_delay, on page 1139.

Copyright © 2013 Synopsys, Inc.

Note: Maintenance for the define_clock_delay command is limited. It is recommended that you use the define_false_path and define_path_delay commands instead, since these commands support clock aliases.

This table provides brief descriptions of the fields shown above.

Delay Parameter	Description
Enabled	Turn this on to enable the constraint. Turn it off to disable an existing constraint.
From Clock Edge	Clock edge that triggers the event of the starting point: *clock_name*, followed by a colon and the edge: (r for rising, f for falling). For example, CLK1:r.
To Clock Edge	Clock edge that triggers the event of the ending point: clock_name:r or f (rise or fall). For example, CLK3:f
Delay (ns)	Path delay in nanoseconds. Alternatively, you can use the keyword false. When you use the false keyword, the timing engine disables the delay field and places an implicit false path constraint on the path. See *Defining False Paths*, on page 151 in the *User Guide* for details.
False Path	You can enable the False Path checkbox to specify false paths for all paths between two clock edges. Clocks that you have assigned to different clock groups are unrelated, which means the software treats any paths between them as implicit false paths. You can use the constraint to override the implicit false path for paths between clocks in different clock groups.

Copyright © 2013 Synopsys, Inc.

The following is an example where a 5 ns delay constraint is placed on CLK4 rising to CLK3 rising, and an implicit false path constraint is placed on CLK3 rising to CLK2 falling.

Inputs/Outputs (Legacy)

This panel models the interface of the FPGA with the outside environment. You use it to specify delays outside the device. The equivalent Tcl constraints are define_input_delay, on page 1143 and define_output_delay, on page 1149. The default delay outside an FPGA is 0.0 ns.

You can specify multiple constraints on the same I/O port. See *Priority of Multiple I/O Constraints*, on page 530 for details.

For information about auto-constraining I/O paths, see *Using Auto Constraints*, on page 739 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

	Enabled	Port	Type	Clock Edge	Value (ns)	Route (ns)	Comment
1	☑	<input default>	input_delay				
2	☑	<output default>	output_delay				
3	☐	resetn	input_delay				
4	☐	porta[7:0]	input_delay				
5	☐	portb[7:0]	input_delay				
6	☐	portc[7:0]	input_delay				

Inputs/Outputs

The SCOPE Inputs/Outputs fields are shown in the following table:

Field	Description
Enabled	(Required) Turn this on to enable the constraint, or off to disable a previous constraint.
Port	(Required) Specifies the name of the port. If you have initialized a compiled design, you can select a port name from the pull-down list. The first two entries let you specify global input and output delays, which you can then override with additional constraints on individual ports.
Type	(Required) Specifies whether the delay is an input or output delay.
Clock Edge	(Recommended) The rising or falling edge that controls the event. The syntax for this field is the clock name, followed by a colon and the edge: r for rising, f for falling. For example, CLK1:r.
Value	(Required) Specifies the delay value. You do not need this value if you supply a Route value.
Route	Improves the delay of the paths to and from the port. The value shrinks the effective synthesis constraint without affecting the constraint that is forward-annotated to the place-and-route tool. This is an advanced user option. See *Route Option*, on page 529 for details.
Comment	Lets you enter comments that are included in the constraints file.

Route Option

Use this option if you do not meet timing goals because the routing delay after placement and routing exceeds the delay predicted by the Synopsys FPGA synthesis tool. The delay reported on an instance along a path from an input port is the amount of time by which the clock frequency goal is exceeded (positive slack), or not met (negative slack) because of the input delay. You can view slack times in the timing report section of the log file, or using the HDL Analyst tool.

Rerun synthesis using this option, which includes the actual route delay (from place-and-route results) so that the tool can meet the required clock frequency. Using the Route option is equivalent to putting a register delay (set_reg_input delay or define_reg_input_delay) on all registers controlled by that clock.

Assume the frequency goal is 50 MHz (clock period goal of 20.0 ns). If the synthesis timing reports show the frequency goal is met, but the detailed timing report from placement and routing shows that the actual clock period is 21.8 ns because of paths through input_a, you can either tighten the constraints in the place-and-route tool, or you can rerun optimization after entering 1.8 in the SCOPE Route column or adding the following define_input_delay -route constraint to the constraint file:

```
# In this example, input_a has a 10.0 ns input delay.
# Rerun synthesis with an added 1.8 ns constraint, to
# improve accuracy and match post-place-and-route results.

define_input_delay {input_a} 10.0 -route 1.8
```

Using this option adds 1.8 ns to the route calculations for paths to input_a and improves timing by 1.8 ns.

Copyright © 2013 Synopsys, Inc.

Priority of Multiple I/O Constraints

You can specify multiple input and output delays (define_input_delay and define_output_delay) constraints for the same I/O port. This is useful for cases where a port is driven by or feeds multiple clocks. The priority of a constraint and its use in your design is determined by a few factors:

- The software applies the tightest constraint for a given clock edge, and ignores all others. All applicable constraints are reported in the timing report.
- You can apply I/O constraints on three levels, with the most specific overriding the more global:
  - Global (top-level netlist), for all inputs and outputs
  - Port-level, for the whole bus
  - Bit-level, for single bits

  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. For example, take the following constraints:

  ```
  a[3:0]3 clk1:r
  a[3:0]3 clk2:r
  a[0]2 clk1:r
  ```

  In this case, port a[0] only gets one constraint of 2 ns. Ports a[1], a[2], and a[3] get two constraints of 3 ns each.
- If at any given level (bit, port, global) there is a constraint with a reference clock specified, then any constraint without a reference clock is ignored. In this example, the 1 ns constraint on port a[0] is ignored.

  ```
  a[0]2 clk1:r
  a[0]1
  ```

Registers (Legacy)

This panel lets the advanced user add delays to paths feeding into/out of registers, in order to further constrain critical paths. You use this constraint to speed up the paths feeding a register. See define_reg_input_delay, on page 1154, and define_reg_output_delay, on page 1155 for the equivalent Tcl commands.

The Registers SCOPE panel includes the following fields:

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Register	(Required) Specifies the name of the register. If you have initialized a compiled design, you can choose from the pull-down list.
Type	(Required) Specifies whether the delay is an input or output delay.
Route	(Required) Improves the speed of the paths to or from the register by the given number of nanoseconds. The value shrinks the effective period for the constrained registers without affecting the clock period that is forward-annotated to the place-and-route tool. For an explanation of this advanced user option, see *Route Option*, on page 529.
Comment	Lets you enter comments that are included in the constraints file.

Copyright © 2013 Synopsys, Inc.

Delay Paths (Legacy)

Use the Delay Paths panel to specify the following timing exceptions:

- 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.

For more details on specifying these constraints, see *Delay Path Timing Exceptions*, on page 500.

This table provides brief descriptions of the fields in the Delay Paths panel. You can use any combination of from, to, and through points: from–through, from, from–through–to, and so on.

SCOPE Cell	Description
Enabled	(Required) Turn this on to enable the constraint.
Delay Type	Specify the type of delay path you want the synthesis tool to analyze. Choose one of the following types: • Multicycle • False • Max Delay
From	Starting point for the path. From points define timing start points and can be clocks (c:), registers (i:), top-level input or bi-directional ports (p:), or black box outputs (i:). For details, see the following: • False Paths, on page 503 • syn_black_box, on page 1235 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*

SCOPE Cell	Description
To	Ending point of the path. To points must be timing end points and can be: clocks (c:), registers (i:), top-level output or bi-directional ports (p:), or black box inputs (i:). For details, see the following: • syn_black_box, on page 1235 • False Paths, on page 503 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*
Through	Specifies the intermediate points for the timing exception. Intermediate points can be combinational nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). If you click the arrow in a column cell, you open the Product of Sums (POS) interface where you can set through constraints. For details, see the following: • syn_black_box, on page 1235 • False Paths, on page 503 • Product of Sums Interface, on page 509 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*
Start/End	Used for multi-cycle paths with different start and end clocks. This option determines the clock period to use for the multiplicand in the calculation for clock distance. If you do not specify a start or end clock, the end clock is the default. For further details, see Multi-cycle Path with Different Start and End Clocks, on page 500 and *Defining Multicycle Paths*, on page 150 in the *User Guide*.
Cycles	Number of cycles for the required time calculated for the multi-cycle path.
Max Delay (ns)	Maximum delay value in nanoseconds.
Comment	Lets you enter comments that are included in the constraints file.

Copyright © 2013 Synopsys, Inc.

Other (Legacy)

The Other panel is intended for advanced users to enter newly-supported constraints. This panel contains the following fields

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Command	(Required) Specifies a command that you want to pass to the place-and-route tool.
Arguments	(Required) Specifies arguments to the command.
Comment	Lets you enter a comment about the commands that you are passing to the place-and-route tool.

See *Entering and Editing SCOPE Constraints (Legacy)*, on page 175 in the *User Guide* for information.

CHAPTER 7

Physical Synthesis User Interface

The Synplify Premier physical synthesis tools provide a graphical user interface (GUI) with windows and views that help you manage input and output files, direct the synthesis process, and analyze your design and its results. The following windows and menu commands are presented here:

- Design Planner User Interface, on page 536
- Design Planner User Interface Commands, on page 543
- Design Planner Tools Menu, on page 552
- Design Planner Popup Menus, on page 558
- Physical Analyst User Interface, on page 566
- Physical Analyst User Interface Commands, on page 574
- Physical Analyst View Popup Menus, on page 580

For details about the standard windows, views, and menu commands for all the Synopsys FPGA products, see also:

- Chapter 2, *User Interface Overview*
- Chapter 3, *User Interface Commands*

Copyright © 2013 Synopsys, Inc.

Design Planner User Interface

This section describes the graphical user interface (GUI) for the Synplify Premier Design Planner tool. Topics include:

- Design Planner Toolbar, on page 542
- Design Planner Keyboard Shortcuts, on page 542

Design Planner View

When the Synplify Premier Design Planner view opens, you can add a design plan file (`sfp`) to your project. The Design Planner consists of an RTL view and three sub-views. For information about the RTL view, see *RTL View*, on page 90. The other Design Planner components are described here:

- Design Plan Hierarchy View, on page 537
- Design Plan View, on page 539
- Design Plan Editor View, on page 539

For more information about using this tool, refer to *Using Design Planner*, on page 612 and for a description of how to create regions, see *Working with Regions*, on page 629 in the *User Guide*.

Design Plan Views

Design Plan Hierarchy View

The Design Plan Hierarchy view is a graphic display of the regions, unassigned design modules, pins, and temporary assigns for the specified device in a hierarchical arrangement. Click on the “+” and “-” signs to expand and contract the hierarchy.

Copyright © 2013 Synopsys, Inc.

The following graphic symbols are shown in the Design Plan Hierarchy view:

Symbol	Description
	Device (Package and Part Type)
	Chip Region (Allows logic replication for top-level, as well as regions)
	Regions Partial Module Assigned Full Module Assigned
	Unassigned Bin Partial Module Assigned Full Module Assigned
	Logic (Drag-and-drop logic assignments)
	Pins
	Temporary Assigns

Design Plan View

The Design Plan view provides a range of information about module assignment within the rows or regions of the device. As you make assignments, the information is updated automatically in this view. The columns displayed are user-defined and can include information such as area and register usage, net I/O definition, and number of connections. For a description of the symbols, see the table in *Design Plan Hierarchy View*, on page 537.

Design Plan Editor View

The Design Plan Editor view shows the physical layout of the specified device and the placement of constraints.

Copyright © 2013 Synopsys, Inc.

Pin Display

You can control pin display with commands from the View menu, as described in *Controlling Pin Display in the Design Plan Editor*, on page 615 in the *User Guide*. When your cursor is on an assigned pin, the tool displays the pin number and the port or net assignment.

Pin colors indicate status, as shown below:

Orange	Selected assigned pins
Red	Deselected assigned pins
Blue	Selected unassigned pins

Device Display

Depending on the technology specified, the image displays LAB (Altera technology) or CLB (Xilinx technology) structures and device-sensitive resources (for example, RAMs or DSPs). The view is context sensitive, so the information displayed in the pane depends on your settings and the objects selected in the Design Plan Hierarchy view. For example, assigned I/O pins display pin directions and the port or net assignments.

You use this view to place and edit regions and perform pin assignments.

The following figure shows examples of the Design Plan Editor view for an Altera device.

Copyright © 2013 Synopsys, Inc.

The following figure shows the Design Plan Editor view for a Xilinx device. Note the clock region boundary.

Copyright © 2013 Synopsys, Inc.

Design Planner Toolbar

In the Synplify Premier tool, the Design Planner toolbar allows you to create block regions on the device floorplan. You can assign critical path logic to these regions, then run physical synthesis and the place-and-route tool to help improve performance for the design.

The following table describes the default Design Planner icons.

Icon	Description
Block Region Tool	Create block regions on the device floorplan, to which you can assign critical path logic to help improve performance for your design.

Design Planner Keyboard Shortcuts

Keyboard shortcuts are key sequences that you type in order to run a command. Menus list keyboard shortcuts next to the corresponding commands. For example, you can press and hold the Ctrl key while you type the e key, instead of using the menu command View ->Expand Pin View.

The following table describes the Design Planner keyboard shortcuts.

Keyboard Shortcut	Description
Ctrl-e	In the Design Planner view, toggles Expanded Pin View on or off.
F9	In the Synplify Premier tool, obtains area utilization for the design. Same as Run->Estimate Area: (see *Run Menu*, on page 237).
Shift-F9	In the Synplify Premier tool, compiles the design to include regions as compile point modules in the compiled netlist. Same as Run->Compile Physical Hierarchy.

Copyright © 2013 Synopsys, Inc.

Design Planner User Interface Commands

The following sections describe the commands and ways to access these commands in the Design Planner graphical user interface (GUI).

- File Menu: Design Planner Commands, on page 543
- Edit Menu: Design Planner Commands, on page 546
- View Menu: Design Planner Commands, on page 547
- Implementation Options: Design Planner, on page 550
- Run Menu: Design Planner Commands, on page 551

File Menu: Design Planner Commands

Use the File menu for creating files. The following table describes the Design Planner File menu commands.

Command	Description
New...	Creates the following types of Design Planner files: Netlist Restructure and Xilinx Options. For example, see: • *Netlist Restructure File Command*, on page 543. • *Xilinx Options File Command*, on page 545

Netlist Restructure File Command

In the Synplify Premier tools, select File->New to display the New dialog box, where you can select the file type of Netlist Restructure File. You can choose to add this file to your project, specifying a file name and location. After you click OK, the netlist restructure file view opens along with the HDL Analyst RTL view.

Copyright © 2013 Synopsys, Inc.

The figure below shows the view for creating the netlist restructure file used to perform the bit-slicing feature. For more information, see *Bit Slicing*, on page 652 in the *User Guide*.

Xilinx Options File Command

Select File->New to display the New dialog box, where you can select the file type of Xilinx Options File. You can choose to add this file to your project, specifying a file name and location. After you click OK, the default Xilinx place-and-route options file is displayed. To run Xilinx place-and-route, see *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide*.

The following figure shows the partial contents of one of the default Xilinx place-and-route options files (lib/xilinx/xilinx_*name*.opt). Edit the contents of the file appropriately, as required, then save the file. To run Xilinx place-and-route, see *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Edit Menu: Design Planner Commands

The following table describes all the Edit menu Design Planner commands:

Edit Menu Commands — Design Planner

Paste - Replicate	Adds assignments from one region to another region.
Delete Region Assignments	Deletes all assignments from all regions.
Delete Selected Assignments	Deletes the instance(s) selected in the Design Plan Hierarchical, Design Plan, or Design Plan Editor view from the corresponding region.
Edit Regions	Enables regions in the Design Plan Editor for sizing, moving, and aligning (Xilinx only).
Delete Region	Deletes the selected region(s) (Xilinx only).
Align Regions	Aligns the selected regions (right, left, up, down) in the Design Plan Editor (Xilinx only).

Copyright © 2013 Synopsys, Inc.

View Menu: Design Planner Commands

These commands are available when the Design Planner window is active. These commands are available in addition to the commands described in *View Menu Commands: All Views*, on page 160 and *View Menu: Zoom Commands*, on page 161.

Command	Description
Show All Regions	Displays all regions.
Expanded Pin View	Enables or disables the expanded I/O pins on the device displayed in the Design Plan Editor.
Adjust Pin View...	Zooms in or out of the expanded pin view so that you can adjust the size of the I/O pins on the device displayed in the Design Plan Editor. See *Adjust Pin View Command*, on page 547.
Rats Nest->Show Rats Nest->Hide Rats Nest->Show Selected	Shows all, hides all, or shows selected rats nesting. Rats nesting displays connectivity between I/O pads and logic for all regions on the device in the Design Plan Editor.
Properties	Displays the Device or Region Properties or the Properties dialog box, showing the properties of the selected object. See *Design Planner Properties Command*, on page 548.

Adjust Pin View Command

Use this command to adjust the size of the pins in the Design Plan Editor for viewing and pin assignments.

Copyright © 2013 Synopsys, Inc.

Command	Description
Slider	Move the slider to increase or decrease the size of the pins.
Continuous view update	When enabled, dynamically displays the pin size changes in the Design Plan Editor as you drag the slider. If disabled, the pin size display updates only after you release the mouse button.

Design Planner Properties Command

Select View->Properties in the Synplify Premier Design Planner to display the Region Properties or Properties dialog box, depending on whether the selected object is a region or not. These two dialog boxes are similar; they both display the properties of the selected object in the Object Properties pane. The Region Properties dialog box also lets you change the region name in this pane.

The Region Properties dialog box has an additional pane, Assignments, where you can unassign selected logic instances from the region. Before exiting the dialog box, you can reassign any instances that you mistakenly unassigned.

Design PlannerToolbar Command

Select View->Toolbars to display the Toolbars dialog box, where you can:

- choose the toolbars to display
- customize their appearance

For details, see *Toolbar Command*, on page 163.

Copyright © 2013 Synopsys, Inc.

Implementation Options: Design Planner

You can use the Project menu to set implementation options or create new implementations. For the Project->Implementation Options... and Project->New Implementation... commands, the following features apply specifically to the Synplify Premier Design Planner:

- Design Planning Panel, on page 550
- GCC & Prototyping Tools Panel, on page 550

Design Planning Panel

In the Synplify Premier with Design Planner tool, you use the Design Planning panel to select the design plan file to use when you synthesize your design for the implementation. See *Using Design Planner*, on page 612 of the *User Guide* for details.

GCC & Prototyping Tools Panel

In the Synplify Premier software, the GCC & Prototyping Tools panel also allows you to specify one or more netlist prototyping file options (the Enable prototyping tools check box must be enabled) and any netlist restructure or Tcl files to compile with the design. Use the Add Prototype file button to add netlist restructure files and Tcl files to the project. See *Using Design Planner*, on page 612 of the *User Guide* for details.

Run Menu: Design Planner Commands

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

- Create a physical hierarchy.
- Estimate area required by each design module.

Command	Description
Estimate Area	Estimates the area required by each design module. Creates an estimation (est) file and an estimation log (_est.srr) file.
Compile Physical Hierarchy	Creates a netlist that includes regions as modules (Design Planner option.

Copyright © 2013 Synopsys, Inc.

Design Planner Tools Menu

When the Synplify Premier Design Planner is open, the Tools menu lets you:

- Display and assign/unassign replicated modules.
- Run area estimation for selected regions or all regions on the device.
- Set preferences for various tool features.

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

- Tools Menu Commands, on page 552
- Show Replicated Assignments Command, on page 553
- Preferences Command, on page 553

Tools Menu Commands

The following table lists the Tools menu commands.

Command	Description
Show Replicated Assignments	Displays all replicated modules assigned to various regions on the device. Assign or unassign replicated logic as necessary. See *Show Replicated Assignments Command*, on page 553 for further details.
Estimate Regions	Runs area estimation for selected regions on the device.
Estimate All Regions	Runs area estimation on all regions on the device.
Design Planner Preferences	Allows you to select preferences for the Design Planner general operations and region assignments. See *Preferences Command*, on page 553 for further details.

Copyright © 2013 Synopsys, Inc.

Show Replicated Assignments Command

Select Tools->Show Replicated Assignments to show all replicated modules assigned to various regions on the device. This displays the Replicated Assignments dialog box, where you can assign or unassign replicated logic, as necessary.

Preferences Command

Select Tools->Design Planner Preferences to define how the Synplify Premier Design Planner displays windows and rats nesting, and handles warnings and pin assignments in this view. The Preferences dialog box is displayed, with tabs to the following panels:

- General Panel
- Assignments Panel
- Pins Panel

General Panel

Sets preferences for displaying the Synplify Premier Design Planner windows and rats nesting.

Copyright © 2013 Synopsys, Inc.

Command	Description
Maximum Undo	For the series of operations you perform in the Design Planner view, specify the maximum number of operations that you can undo while in a session of this view.
Tile with Analyst when opening	If enabled, the HDL Analyst RTL view will also be tiled with the Design Planner view when it is opened.
Include Pins	When you turn on rats nesting, you can choose to display connections to I/O pins or not.
Show Number of Connections	When you turn on rats nesting, you can choose to display the number of connections between regions or I/O pins after making assignment to these resources.

Assignments Panel

Sets preferences for displaying assignment warnings, area estimate warnings, and rats nests when dragging and dropping assignments.

Command	Description
Show assignment warnings	When enabled, this option might display messages when you assign logic to regions. For example, see *Region Assignment Messages Dialog Box*, on page 556.
No area estimate warning	When enabled, this option displays a dialog box (see *Estimation Needed and Running Estimation Dialog Boxes*, on page 557) when you first open Design Planner.
Show rats' nest when dragging	When enabled, shows the attached nets to regions as you drag an instance or pin from the RTL view to the Design Planner view.

Pins Panel

Specifies if pin assignments can be edited and if multiple-pin assignments are allowed.

Copyright © 2013 Synopsys, Inc.

Command	Description
Pin assignments editable	When enabled, this option will not allow you to assign ports to pin locations. By default, this option is disabled.
Allow multi-pin assignments	When enabled, this option allows you to assign ports to multiple pin locations. Otherwise, you will be prohibited from doing this.

Region Assignment Messages Dialog Box

The Instance Replication dialog box is displayed when you try to assign the same logic to different regions, if you have enabled Show assignment warnings on the Assignments tab of Tools->Design Planner Preferences.

Estimation Needed and Running Estimation Dialog Boxes

The Estimation Needed dialog box opens when you first open the Design Planner, if you have enabled No area estimate warning on the Assignments tab of Tools->Design Planner Preferences.

The Running Estimation dialog box shows the time taken to get an area estimate. It opens if you click Yes on the Estimation Needed dialog box.

Copyright © 2013 Synopsys, Inc.

Design Planner Popup Menus

The commands available in the Synplify Premier Design Planner popup menu vary, depending on the technology, the current status of the design, what is selected, and the panel in which you are working. This section describes the following popup menus and dialog boxes:

- Design Plan Hierarchical Browser Popup Menu, on page 558
- Design Plan Popup Menu, on page 558
- Design Plan Editor Popup Menu, on page 559
- Properties Popup Menu Command (Design Planner), on page 560
- Show/Hide Columns Popup Menu Command, on page 561
- Assign Dialog Box, on page 564
- Instance Replication Dialog Box, on page 565

The following tables describe the Synplify Premier Design Planner popup menu commands. Popup menu commands from the Design Planner are added to the initial commands listed in these tables as you create physical regions and make assignments to regions and I/O pins. This lets you access the same commands from different locations within the synthesis tool.

Design Plan Hierarchical Browser Popup Menu

This table shows the popup menu commands available in the Hierarchical Browser of the Design Planner.

Command	Description
Properties	Displays the Properties dialog box, listing statistics (device, region, unassigned bins, or temporary assigns) for the object selected, depending on its technology.

Design Plan Popup Menu

This table shows the popup menu commands available in the Design Plan view of the Design Planner.

Copyright © 2013 Synopsys, Inc.

Command	Description
Auto-size columns	Automatically adjusts the column widths to fit in the Physical Report view.
Hide column *colname*	Hides the column called *colname.*
Show Column...	Lets you add columns by choosing from those in this submenu.
Show/Hide Columns...	Displays the Select Columns dialog box, where you can choose to enable or disable device options for reporting statistics in the column display for the current device. See *Show/Hide Columns Popup Menu Command*, on page 561.

Design Plan Editor Popup Menu

This table shows the popup menu commands available in the Design Plan Editor view of the Design Planner.

Command	Description
RTL View	Displays the RTL view for the selected or unselected physical regions, depending on which you choose.
Delete Region Assignments	Deletes assignments for the selected physical region.
Delete Region	Deletes the selected physical regions.
Estimate Regions	Estimates utilization of mapped resources, such as LUTs and registers, for the selected physical regions.
Region Type	• Soft (Tunneling On) – in or out assignments allowed. • Hard (Tunneling Off) – only in assignments, no out. • Keep-Out – ensures that no placement occurs in the region. Use this option to create decongestion areas for optimizing your design.
LogicLock ->Enabled ->Disabled	Enables or disables assignment of the selected regions as Altera LogicLock™ regions.
Show/Hide Region	Shows or hides the selected physical regions.

Copyright © 2013 Synopsys, Inc.

Command	Description
Paste	Pastes the assignments of the most recently copied physical region to the selected physical region.
Paste Replicate	Pastes the assignments of the most recently copied replicated logic to the selected physical region.
Edit Regions	Locks or unlocks the selected physical regions for changing its size or location.
Add	Creates a new physical region. The available commands in this submenu depend on the current device. See Chapter 12, *Floorplanning with Design Planner*.
Show All Regions	Displays all physical regions, including those that were hidden.
Estimate All Regions	Estimates utilization of mapped resources, such as LUTs and registers, for all physical regions.
Rats Nest	Displays the interconnects between the physical resources on the device.
Properties	Displays the Properties dialog box, listing statistics (device, region, unassigned bins, or temporary assigns) for the object selected, depending on its technology.

Properties Popup Menu Command (Design Planner)

The Properties dialog box displays properties of the currently selected object. The properties shown depend on the technology and the current status of your design.

Copyright © 2013 Synopsys, Inc.

Show/Hide Columns Popup Menu Command

In the Design Plan view, select the Show/Hide columns popup menu command to display the Select Columns dialog box. Customize your view by selecting or deselecting the available column headings.

The table below lists most of the resources, only some of which are available at any time, depending on the device selected.

Copyright © 2013 Synopsys, Inc.

Column Selection	Description
Area, Area Use, Area Use (%)	Total number of mapped resources, number of resources used, and percentage of resource utilization after you run Estimate Regions for the region.
Assigns	Number of assignments in the region.
BlockFifos, BlockFifo Use, BlockFifo Use (%)	Number of Block Fifos, number of Block Fifos used, and percentage of Block Fifos used after you run Estimate Regions in the region. Xilinx only.
BlockMults, BlockMult Use, BlockMult Use (%)	Number of Block Mults, number of Block Mults used, and percentage of Block Mults used after you run Estimate Regions in the region. Xilinx only.
BlockRams, BlockRams Use, BlockRams Use (%)	Number of Block RAMs, number of Block RAMs used, and percentage of Block RAMs used after you run Estimate Regions in the region. Xilinx only.
BlockFifo Dim	The dimension of the Block Fifo(s) for the region. Xilinx only.
BMULT Dim	The dimension of the Block Mult(s) for the region. Xilinx only.

Copyright © 2013 Synopsys, Inc.

Column Selection	Description
BRAM Dim	The dimension of the Block RAM(s) for the region. Xilinx only.
CLB Dim	The dimension of the CLB(s) for the region. Xilinx only.
Dimension	The dimension of the ROW regions.
DSP, DSP Use, DSP Use (%)	Number of DSPs, number of DSPs used, and percentage of DSPs used after you run Estimate Regions in the region. Altera Stratix only.
ESB Dim	The dimension of the ESB(s) in the region. Altera only.
ESB, ESB Use, ESB Use (%)	Number of ESBs, number of ESBs used, and percentage of ESBs used after you run Estimate Regions in the region. Altera only.
LAB Dim	The dimension of the LAB(s) for the region. Altera only.
Luts, Lut Use, Lut Use (%)	Number of LUTs, number of LUTs used, and percentage of LUTs used in the region after you run Estimate Regions.
M4K, M4K Use, M4K Use (%)	Number of M4Ks, number of M4Ks used, and percentage of M4Ks used after you run Estimate Regions in the region. Altera Stratix only.
M512, M512 Use, M512 Use (%)	Number of M512s, number of M512s used, and percentage of M512s used after you run Estimate Regions in the region. Altera Stratix only.
MaxChainLength, MaxChainLength Use, MaxChainLength Use (%)	Length of longest carry chain, length of the longest carry chain assigned to the region, and percentage of the longest carry chain assigned to the region after you run Estimate Regions.
MegaRAM, MegaRAM Use, MegaRAM Use (%)	Number of MegaRAMs, number of MegaRAMs used, and percentage of MegaRAMs used after you run Estimate Regions in the region. Altera Stratix only.

Copyright © 2013 Synopsys, Inc.

Column Selection	Description
Name	Name of the region.
Regs, Reg Use, Reg Use (%)	Number of registers, number of registers used, and percentage of registers used after you run Estimate Regions in the region.
Tunneling	Tunneling if enabled/disabled.

Assign Dialog Box

In the Synplify Premier tool, you can assign modules or primitives from an HDL Analyst, Design Plan Hierarchy Browser, or Design Plan view to regions you create in the Design Plan Editor by selecting Assign->*regionName* from the popup menu. The regions listed in the popup menu are in order of most recent use. If there are more than 25 regions, choosing More in the menu displays the Assign To dialog box, where you can choose from additional regions. The additional regions are listed alphabetically in the dialog box.

Instance Replication Dialog Box

When you assign the same instance logic from the HDL Analyst RTL view to different regions created in the Design Plan Editor, an Instance Replication dialog box appears as shown in the following figure. Confirm whether or not you want to replicate the selected logic instance to the specified region. You can choose to confirm replication for each instance separately (Yes or No) or simultaneously (Yes to All or No to All) for multiple instances.

Copyright © 2013 Synopsys, Inc.

Physical Analyst User Interface

This section describes the graphical user interface (GUI) for the Synplify Premier Physical Analyst tool. Topics include the:

- Physical Analyst View, on page 566
- Physical Analyst Toolbar, on page 571
- Physical Analyst Keyboard Shortcuts, on page 572

Physical Analyst View

The Physical Analyst tool provides a visual display of the placement and global routing of the design after running physical synthesis and place and route. It consists of an optional control panel and a window that displays a device.

Copyright © 2013 Synopsys, Inc.

Physical Analyst Control Panel (Objects Tab)

The Physical Analyst control panel is an embedded pane that is optionally displayed on the left side of the Physical Analyst view. You can toggle the display of this panel by clicking on the Physical Analyst Control Panel icon ( ) in the toolbar or selecting Options->Physical Analyst Control Panel.

Select the Objects tab on the control panel to determine the visibility and selectability of various objects on the device. The control panel also contains a context window. See *Physical Analyst Context Window*, on page 571.

Copyright © 2013 Synopsys, Inc.

Command	Description
Instances	Sets the visibility and selectability of instances in the device view. Cell instances (Core) are drawn at their placement location and orientation. Unplaced instances are not drawn, but can be located using the Find command. For more information about instances, see *Displaying Instances and Sites in Physical Analyst*, on page 672.
Instance Display	You can: • Set the visibility of signal pins to appear or not for core instances. • Display the enhanced view of core instances. Same as View->Configure Enhanced Instance Display.
Nets	Sets the visibility and selectability of nets after they have been routed using on-demand routes such as Expand or Route Selected Instances. Nets are routed on one metal layer, and display the point-to-point connections from output pins to input pins. • Signal Flow adds directional arrows to nets. • Pruned Signals disables the display of unconnected signals.
Sites	Sets visibility for the sites defined in the vendor-specific cell library files. Sites can be visible or hidden. Use tool tips to display the site row number and its boundaries (Instances must not be selectable).

The device display in the Physical Analyst view is based on the following input files:

Files	Description
`.lef` files	Provide physical cell library information for the various devices
`.def` files	Define the device floorplan
`.srm` file	Contains the netlist and describes instance placement

Physical Analyst Control Panel (Maps Tab)

The Physical Analyst control panel is an embedded pane that is optionally displayed on the left side of the Physical Analyst view. You can toggle the display of this panel by clicking on the Physical Analyst Control Panel icon ( ) in the toolbar or selecting Options->Physical Analyst Control Panel.

Select the Maps tab on the control panel to display color-coded maps which overlay the device providing information about the design after it has been synthesized. For more information about these implementation maps, see *Using Implementation Maps in Physical Analyst*, on page 710.

Copyright © 2013 Synopsys, Inc.

Command	Description
Select Maps	You can choose one of the following Implementation Maps: • Route Congestion • Block Component Utilization • Block Input Utilization • Slack Distribution • None—no maps are displayed; this is the default.
Show visible design objects	When enabled, both the map and instances are displayed in the Physical Analyst view. Otherwise, only the map without instances is shown. This option is turned off by default.

Slider Scale	Two slider scale modes are available for the Slack Distribution map; they are Linear and Log. The threshold, alert, and histogram can move at a linear or logarithmic rate.
Limits	Use the Threshold and Alert sliders adjust the threshold and the alert percentage levels, respectively, for the implementation maps.
Histogram	Displays 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

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. This is helpful when you perform operations in the Physical Analyst view, such as zooming in, for which the context window displays a rectangle around the relevant area on the device.

Physical Analyst Device View

The Physical Analyst device view shows cell placement and connectivity, based on the settings in the control panel. The device view displays

- Device and cell boundaries
- Placement site rows
- Nets

Physical Analyst Toolbar

Toolbars provide a quick way to access common menu commands by clicking their icons.

In the Synplify Premier tool, the Physical Analyst toolbar provides access to a visual display of the floorplan, placement, and global routing of the design after running physical synthesis and place-and-route. The following standard toolbars are available:

Copyright © 2013 Synopsys, Inc.

The following table describes the default Physical Analyst icons.

Icon	Description
Physical Analyst	Opens a new Physical Analyst view of your synthesized design. The view displayed is flat, although the hierarchy of the instance name is retained. Use the Physical Analyst commands to control how objects are displayed in this viewer. There are also a number of commands to use for analyzing the netlist that is generated.
Physical Analyst Control Panel	Toggles the display of the Physical Analyst Control panel. Same as selecting Options->Physical Analyst Control Panel
Reset Filter	Resets the original view to hide all nets in the display. Same as selecting Unfilter->Show All Instances, Hide All Nets

Physical Analyst Keyboard Shortcuts

Keyboard shortcuts are key sequences that you type in order to run a command. Menus list keyboard shortcuts next to the corresponding commands. For example, you can press and hold the Ctrl key while you type the g key, instead of using the menu command Go to Location.

The following table describes the Physical Analyst keyboard shortcuts.

Keyboard Shortcut	Description
Ctrl-b	In the Physical Analyst view, expands paths to all pins for two or more instances selected.
Ctrl-e	In the Design Planner view, toggles Expanded Pin View on or off.
Ctrl-g	In the Physical Analyst view, displays the Go to Location dialog box so you can find objects at a particular location.
Ctrl-h	In the Physical Analyst view, shows the drivers for selected nets.
Ctrl-i	In the Physical Analyst view, selects all instances for nets expanding them to their input pins.
Ctrl-k	In a Physical Analyst view, opens the Physical Analyst control panel. Same as Options->Physical Analyst Control Panel (see *Using Physical Analyst*, on page 665).
Ctrl-r	In the Physical Analyst view, routes selected instances connecting them to their output pins.
Ctrl-u	In the Physical Analyst view, refreshes the display for the current view.
Ctrl-Shift-h	In the Physical Analyst view, selects the drivers for the selected nets.
Ctrl-Shift-i	In the Physical Analyst view, only highlights instances expanded to input pins for selected nets.

Copyright © 2013 Synopsys, Inc.

Physical Analyst User Interface Commands

The following sections describe the commands and ways to access these commands in the Synplify Premier Physical Analyst graphical user interface (GUI).

- Edit Menu: Physical Analyst Commands, on page 574
- View Menu: Physical Analyst Commands, on page 574
- HDL Analyst Menu: Physical Analyst Commands, on page 579
- Options Menu: Physical Analyst Commands, on page 579

Edit Menu: Physical Analyst Commands

The following table describes all the Edit menu Physical Analyst commands:

Edit Menu Commands for the Physical Analyst View

Find...	Displays the Object Query dialog box, which lets you search your design for instances, symbols, and nets, by name and object type. See *Find Command (Physical Analyst View)*, on page 587.

View Menu: Physical Analyst Commands

Use the View menu to set the display and viewing options, choose toolbars, and display result files. The commands in the View menu might vary depending on what is selected within the active view.

The commands are available when the Physical Analyst view is active. These commands are available in addition to the commands described in *View Menu Commands: All Views*, on page 160 and *View Menu: Zoom Commands*, on page 161.

Copyright © 2013 Synopsys, Inc.

Command	Description
Filter	Shows only the desired objects in the Physical Analyst view. You can filter any of the following: • Show Selected – only show selected objects • Hide Selected – hide selected objects • Hide All – hide all objects • Hide All Instances – hide all instances • Hide All Nets – hide all nets • Forward – move forward from a previous filtered display
Unfilter	Expands the view to display the desired objects in the Physical Analyst view. You can show any of the following: • Reset – resets the design to show all instances and ports, but hide all nets • Reset All – resets the design to show all instances, ports, and nets • Back – moves back to a previous filtered display
Show Critical Path	Enters a filtered state where only instances and nets belonging to the critical timing path are shown in the Physical Analyst view.
Unselect All	Unselect all objects selected in the view.
Cross Probing	Enables/disables cross probing to/from the Physical Analyst tool. See *Cross Probing Command (Physical Analyst)*, on page 577.
Configure Enhanced Instance Display	Selects the enhanced display mode where core cells are displayed as fix-sized diamonds in the Physical Analyst view. See *Configure Enhanced Instance Display (Physical Analyst)*, on page 577.
Selection Transcription	When enabled, allows you to select an object in the Physical Analyst view, then displays the object's tool tip information in the TCL window. This gives you a running history of the objects you have examined. By default, this option is turned on.
Tool Tips	Controls the display of tool tips when the cursor is placed over objects in the view.
Physical Analyst Properties	Displays information about the design and device in the Physical Analyst view.

Copyright © 2013 Synopsys, Inc.

Command	Description
Zoom->Zoom Selected	Zooms into the selected object. For other zoom commands, refer to *View Menu: Zoom Commands*, on page 161.
Go to Location	Zooms in on the object at the specified coordinate pair location.
Refresh	When drawing large designs in the view, you can use the ESC key to cancel the current display. To restart drawing in the view again, use the Refresh option to start the display.

Physical Analyst Toolbar Command

Select View->Toolbars to display the Toolbars dialog box, where you can:

- choose the toolbars to display
- customize their appearance

For details, see *Toolbar Command*, on page 163.

Cross Probing Command (Physical Analyst)

View->Cross Probing is available in the Physical Analyst view of the Synplify Premier tool. The following enable/disable cross-probing options are available:

Command	Description
Send Crossprobes when selecting	Enables or disables cross probes from the Physical Analyst. For efficiency reasons, you might want to disable this option (i.e., not send cross probes).
Cross Probing from RTL Analyst	Enables or disables cross probing from the RTL Analyst view. Mapped or placed instances are highlighted in the Physical Analyst view in response to selection of objects in the RTL view.
Cross Probing from Tech Analyst	Enables or disables cross probing from the Technology view. Mapped or placed instances are highlighted in the Physical Analyst view in response to selection of objects in the Technology view.
Cross Probing to HDL Source	Enables or disables cross probing to the HDL source.
Auto route cross probed insts	Enables or disables instances that are cross probed to be automatically routed.

Configure Enhanced Instance Display (Physical Analyst)

The enhanced display mode, when enabled, causes core cells to be drawn as diamonds of fixed size regardless of zoom level. Selecting View->Configure Enhanced Instance Display in an active Physical Analyst view brings up the Enhanced Instance Display dialog box which is used to set the enhanced display parameters.

Copyright © 2013 Synopsys, Inc.

Option	Description
Enhance Instance Shape for Better Visibility	Enables enhanced instance display when checked.
Visible Instance Limit for Enhancement	Sets the maximum number of visible core cells that can be displayed in enhanced instance display mode.
Enhancement Size (pixels)	Sets the size (in pixels) of the instance; instances are drawn as diamonds to differentiate them from normal cell shapes.
Minimum Size for Normal Draw (pixels)	Sets the minimum size of an average core cell when the cell is drawn in normal mode and not enhanced

Copyright © 2013 Synopsys, Inc.

HDL Analyst Menu: Physical Analyst Commands

In the Project View, the HDL Analyst menu contains commands that provide project analysis in the Physical Analyst view.

Physical Analyst

The Physical Analyst feature is available only for the Synplify Premier product. Select HDL Analyst->Physical Analyst to display the Physical Analyst view. The Physical Analyst tool provides a visual display of the floorplan, placement, and global routing of the design after physical synthesis and place-and-route have been run. For more information, see *Using Physical Analyst*, on page 665 of the *User Guide*.

Options Menu: Physical Analyst Commands

Use the Options menu to set options for the Physical Analyst view. When using certain technologies, additional menu commands let you run technology-vendor software from this menu.

Options Menu Commands Specifically for the Physical Analyst View

Physical Analyst Control Panel	Toggles the display of the Physical Analyst control panel. See *Using Physical Analyst*, on page 665 in the *User Guide*.
Physical Analyst Color Schemes	Allows you to change to another pre-defined color scheme in the Physical Analyst view. Currently, only one standard color scheme is available.

Copyright © 2013 Synopsys, Inc.

Physical Analyst View Popup Menus

In the Synplify Premier tool, popup (right-click) menus in the different tool views are usually comprised of commonly-used menu or toolbar commands. Usually commands in the pop-ups are also available through UI menus or buttons. For example: right-click->Filter can also be run through View->Filter. However, there are a some commands that are only available through popup menus. Refer to the following tables for a complete list of Physical Analyst commands:

- Global Commands, on page 580
- Commands with Instances Selected, on page 582
- Commands with Nets Selected, on page 584
- Design, Instance, and Net Properties (Physical Analyst), on page 584
- Find Command (Physical Analyst View), on page 587
- Resolve Selection Dialog Box, on page 591

Global Commands

The following table describes global commands that are commonly used for selected objects and available through a popup (right-click) menu in the Physical Analyst view.

Command	Description
Properties	Displays the properties of the selected objects (for example, mapped instances or nets). See *Instance Properties*, on page 585.
Selection Tool	Cancels operations, such as when using the measurement tool in the Physical Analyst view, so that you can select other objects in the view thereafter. You can also click on the Selection Tool icon on the Physical Analyst toolbar to cancel this operation.
Signal Flow	Displays the signal flow of nets. When you select a net, the net is displayed with arrows showing the predominant direction of its signal flow (right, left, up, or down).

Copyright © 2013 Synopsys, Inc.

Command	Description
Measurement Tool	This measurement tool calculates the total distance in microns from a start location to an end location in the X and Y directions and displays these values in the Physical Analyst and Tcl windows.
Filter	Filters the design to display only the desired objects in the view. You can filter any of the following: • Show Selected – only show selected objects • Hide Selected – hide selected objects only • Hide All – hide all objects in the view • Hide All Instances – hide all instances in the view • Hide All Nets – hide all nets in the view • Forward – move forward to a filtered display in history of the current view
Unfilter	Expands the view to display the desired objects in the view. You can show any of the following: • Reset – resets the design to show all instances and ports, but hide all nets • Reset All – resets the design to show all instances, ports, and nets • Back – moves back to a previous filtered display in history of the current view
Find	See *Find Command (Physical Analyst View)*, on page 587.
Select ->All Instances and Nets ->All Instances ->All Nets ->Deselect All	Globally selects and highlights any of the following in the view: all instances and nets, all instances only, or all nets only. Conversely, you can globally deselect and unhighlight all selected objects in the view.
Critical Path -> Show Critical Path ->Expand Path Forward ->Expand Path Backward	Ways to display the critical path: • Enters a filtered state where only instances and nets belonging to the critical timing path are shown. • Traces the critical path from the start point to the end point. • Traces the critical path from the end point to the start point.

Copyright © 2013 Synopsys, Inc.

Command	Description
Zoom Selected	Zooms in on the selected object.
Markers ->Add marker ->Remove All ->Remove Selected ->Go to Next ->Go to Previous	Create any marker at a specified location. You can: • add a marker • remove all markers • remove a selected marker • move forward to the next marker in the order they were created • move backwards to the previous marker in the order they were created
Go to Location	Allows you to specify a coordinate pair location for an object and then zoom in on the object if requested.
Selection Transcription	When enabled, allows you to select an object in the Physical Analyst view, then displays the object's tool tip information in the TCL window. This gives you a running history of the objects you have examined. By default, this option is enabled.
Tool Tips	Enables/disables tool tips to be displayed as you move the mouse over objects in the view, providing additional information for these objects at-a-glance.
Refresh	When drawing large designs in the view, you can use the ESC key to cancel the current display. To restart drawing in the view again, use the Refresh option to start the display.
Physical Analyst Properties	Displays information about the design and device. See *Design, Instance, and Net Properties (Physical Analyst)*, on page 584.

Commands with Instances Selected

When one or more instances are selected, certain commands are included in popup menus, regardless of where you right-click. The following commands are available when instances are selected. Note that some commands are only available from this popup menu.

Instance-specific Command	Description
Properties (Core Cell)	Displays information about the cell you select. See *Instance Properties*, on page 585.
Expand ->Output Pins ->Input Pins ->All Pins ->Selected Pins	Expands logic starting from an instance or instances. In general, start with some visible instances and filter or hide the remainder. You can then expand either from: • all pins • selected pins • input pin • output pins Expansion reveals (unfilters and selects) instances to which the instance connects following the signal direction of pins. Expansion is stopped when reaching a register, black box, or port. As you expand, nets are shown connecting the instances. If you enable net pruning the portions of nets connecting to invisible instances are dimmed.
Expand to Register/Port ->Output Pins ->Input Pins ->All Pins ->Selected Pins	See all cells that are connected to a pin, output pins, input pins, or all pins, up to the next register/port.
Expand Paths ->All Pins ->Selected Pins	Shows all logic between two or more selected instances. Unfilters and selects instances and nets along the signal paths of the selected instances. As in the case of Expand you can control the pins to expand (all pins, selected pins).
Route Selected Instances ->Connected ->Output Pins ->Input ->All Pins	Shows nets connected to selected instances. As in the case of Expand you can control the pins to route (all pins, input pins, output pins).

Copyright © 2013 Synopsys, Inc.

Commands with Nets Selected

When one or more nets are selected, the following context-sensitive commands are included in popup menus, regardless of where you right-click.

Net-specific Command	Description
Properties (Net)	Displays information about the net you select. See *Instance Properties*, on page 585.
Select Net Instances ->Input Pins ->Output Pins ->All Pins	Selects all instances connected to the selected net. You can expand the net from input pins, output pins, or all pins. Unfilters and selects instances along the signal paths of the selected pins.
Highlight Visible Net Instances ->Input Pins ->Output Pins ->All Pins	Shows only nets connected to the selected instances. The net can be expanded from input pins, output pins, or all pins.
Select Net Driver	Selects the driver for a net.
Go to Net Driver	Shows and scrolls to the driver for a selected net.

Design, Instance, and Net Properties (Physical Analyst)

You can display information for the following in the Synplify Premier Physical Analyst view:

- Design Properties, on page 584
- Instance Properties, on page 585
- Net Properties, on page 586

Design Properties

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

The dialog box has a read-only Design pane, which lists 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.

Instance Properties

To display the properties for an instance, right-click on the instance and select Properties (Core Cell). The dialog box lists read-only information like the instance name, type, number of pins, placement location, device-specific location, delay, slack, and clock signal. It also indicates if the instance is included in the critical path.

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

Copyright © 2013 Synopsys, Inc.

Net Properties

To display the properties for a net, right-click on the net and select Properties (Net). The dialog box lists read-only information like the net name, logical nets, pin count, and fanout. It also indicates if the net is globally routed, and if the net is a clock.

Find Command (Physical Analyst View)

In the Physical Analyst view, right-click and select Find from the popup menu or press Ctrl-f to open the Find Object dialog box. The view displayed is flat, although the hierarchy of instance names is retained.

Copyright © 2013 Synopsys, Inc.

Objects listed in the Highlighted window of Find Object are also highlighted in the HDL Analyst and/or Physical Analyst views. The following table provides descriptions for the parameters in this dialog box. More sections about filtering and searching for objects are presented after the table.

Option	Description
Instances, Symbols, Nets, Ports	Specifies the design objects for which to search. In terms of memory consumption, searching for Instances requires the least amount, searching for Nets requires the most.
Search	Limits/expands the search to the entire design.
Search by Name	In the Physical Analyst tool, use with the Find 200 and Find All, for a case-sensitive search of the string that you specify in this field. The string can contain the following wildcard characters: • * (asterisk) – matches any sequence of characters; • ? (question mark) – matches any single character; • . (period) – does not match any characters, but indicates a change in hierarchical level.

Option	Description
Filter Search	In the Physical Analyst tool, use with the Find 200 and Find All, for a lists object names in the UnHighlighted area. Items in this field depend on the design object selected. See *Object Filter Search for Find Command*, on page 589.
UnHighlighted	Lists all objects in the design.
Highlighted	All objects in this section are highlighted in the current schematic or floorplan view. To highlight objects in the schematics, select them from the unhighlighted section and move them to the highlighted section using the arrows or by double-clicking the objects. You can select multiple objects using the Ctrl or Shift key. To deselect an object, press Ctrl and click on the object.
Highlight Search (*?)	In Physical Analyst, you can further define the search for instances, symbols, nets, or ports. See *Object Filter Search for Find Command*, on page 589.
Un-Highlight Selection (*?)	Uses wildcards for filtering objects to deselect in the schematic views.
Jump to location	Displays the target on any sheet in the schematic.
Find 200	Used with the Highlight Search (*?) filter string, displays the first 200 objects that match the filter string in the UnHighlighted list. Each time you click the button, an additional 200 objects are added to the list. Find 200 does not highlight objects in the schematic or floorplan views, but filters what is listed in the Unhighlighted field.
Find All	Used with the Highlight Search (*?) filter string, displays all objects that match the filter string in the UnHighlighted list. Find All does not highlight objects in the schematic or floorplan views, but filters what is listed in the Unhighlighted field.

Object Filter Search for Find Command

The Physical Analyst lets you fine-tune your search for design objects by filtering the kind of objects you are searching. In the Find Object dialog box (right-click->Find in a Physical Analyst view). The Highlight Search (*?) field has a pull-down menu that lists the additional object filters available for the current type of object, based on the Find object tab selected.

Copyright © 2013 Synopsys, Inc.

The following tables list the choices available for the Net, Instance, Port, and Symbol tabs. Design objects matching the search are listed in the Unhighlighted field in the dialog box. To highlight the objects in the view, use the arrows to move the desired objects to the Highlighted field.

Instances

From the Instances panel, the pull-down list contains the following filters:

Selection	Only searches....
Combinatorial	Combinatorial instances
Cover placement	Fixed macro placements that cannot be changed
Core cell	Cells (instances) in the core area
Critical path	Instances on a critical path
Critical path end	Instances at the end of a critical path
Critical path start	Instances at the start of a critical path
Fixed placement	Pre-placed or fixed macro placement
Macro cell	Macro cells (instance)
Module	Groups of instances that form logical modules
Negative slack	Instances with negative slack
Pad cell	Pad cells on the perimeter of the device
Placed	Placed instances
Placed placement	Pre-placed or fixed macro placement that can be changed by the program
Primitive	Primitives
Sequential	Sequential instances
Unplaced	Unplaced instances

Copyright © 2013 Synopsys, Inc.

Symbols

From the Symbols panel, the pull-down list contains the following filters:

Selection	Only searches....
Module	Hierarchical module
Primitive	Primitives

Nets

From the Nets panel, the pull-down list contains the following filters

Selection	Only searches....
Clock	Nets from clock ports

Ports

From the Ports panel, the pull-down list contains the following filters

Selection	Only searches....
Bidir	Bidirectional ports
Clock	Clocks
Input	Input ports
Output	Output ports

Resolve Selection Dialog Box

This box lists overlapping objects at a particular cursor location in the Physical Analyst and helps you resolve selection by selecting objects from the list. As you move the cursor over overlapping objects, you see a special cursor. When you click the cursor at that location, the Resolve Selection dialog box pops opens.

Copyright © 2013 Synopsys, Inc.

Resolve Selection
Select Object(s)
in3_ibuf
out2_obuf
All
Close
None

CHAPTER 8

Input and Result Files

This chapter describes the input and output files used by the synthesis tool.

- Input Files, on page 594
- Libraries, on page 599
- Output Files, on page 601
- Log File, on page 608
- Timing Reports, on page 615
- Hierarchical Area Report, on page 623
- Constraint Checking Report, on page 629

Copyright © 2013 Synopsys, Inc.

Input Files

The following table describes the input files used by the synthesis tool.

Extension	File	Description
`.adc`	Analysis Design Constraint	Contains timing constraints to use for stand-alone timing analysis. Constraints in this file are used only for timing analysis and do not change the result files from synthesis. Constraints in the `adc` file are applied in addition to `sdc` constraints used during synthesis. Therefore, `adc` constraints affect timing results only if there are no conflicts with `sdc` constraints. You can forward annotate `adc` constraints to your vendor constraint file without rerunning synthesis. See *Using Analysis Design Constraints*, on page 732 of the *User Guide* for details.
`.cdc`	Compiler Design Constraint	*Synplify Premier tools only* Specifies directives to be applied to the source code. See *Specifying Directives in a CDC File*, on page 248 of the *User Guide* for information about specifying the directives and using this file. See *Compiler Directives File Examples*, on page 1168 for examples.
`.fdc`	Synopsys FPGA Design Constraint	Create FPGA timing and design constraints with SCOPE. You can run the sdc2fdc utility to translate legacy FPGA timing constraints (SDC) to Synopsys FPGA timing constraints (FDC). For details, see the *sdc2fdc Tcl Shell Command*, on page 988.
`.ini`	Configuration and Initialization	Governs the behavior of the synthesis tool. You normally do *not* need to edit this file. For example, use the HDL Analyst Options dialog box, instead, to customize behavior. See *HDL Analyst Options Command*, on page 345. On the Windows 7 platforms, the `ini` file is in the C:\Users*userName*\AppData\Roaming\Synplicity directory, and on the Windows XP platforms, the `ini` file is in the C:\Documents and Settings*userName*\Application Data\Synplicity directory. On Linux workstations, the `ini` file is in the following directory: (~/.Synplicity, where ~ is your home directory, which can be set with the environment variable $HOME).

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
.ncf	Xilinx constraints file	Contains Xilinx constraints that can be translated to the sdc format and used to drive synthesis.
.nrf	Netlist Restructure	In the Synplify Premier software, displays the netlist restructure file view along with the HDL Analyst RTL view to perform bit slicing. For more information, see *Netlist Restructure File Command*, on page 543.
.opt	Xilinx P&R options file	An editable text file containing place-and-route option information to use when running Xilinx place-and-route with xflow. For the newer xtclsh flow, a Tcl file is used instead.
.prj	Project	Contains all the information required to complete a design. It is in Tcl format, and contains references to source files, compilation, mapping, and optimization switches, specifications for target technology and other runtime options.
.qsf	Altera constraints file	Contains Altera constraints that can be translated to the sdc format and used to drive synthesis.
.sdc	Constraint	Contains the timing constraints (clock parameters, I/O delays, and timing exceptions) in Tcl format. You can either create this file manually or generate it by entering constraints in the SCOPE window.
.sfp	Design Plan	In the Design Planner option of the Synplify Premier software, contains the design plan for an implementation in Tcl format. The sfp file is created when you assign a design plan to the critical path(s) in your design with the Design Planner. For example: • Region definitions: size and location. Some region definitions are determined by target device and the place-and-route tools. • Region assignments: assignment of logical instances to regions • Special assignments: assignment of logical instances to special chip resources like block RAMs or block Mults (Xilinx), or MACs (Altera DSPs) • Pin assignments: assignment of I/Os to specific FPGA device pin numbers

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
.sv	Source files (Verilog)	Design source files in SystemVerilog format. The sv source file is added to the Verilog directory in the Project view. For more information about the Verilog and SystemVerilog languages, and the synthesis commands and attributes you can include, see *Verilog*, on page 598, Chapter 9, *Verilog Language Support*, and Chapter 10, *SystemVerilog Language Support*. For information about using VHDL and Verilog files together in a design, see *Using Mixed Language Source Files*, on page 117 of the *User Guide*.
.tcl	Xilinx P&R options file	An editable text file containing place-and-route option information to use when running Xilinx place-and-route with the new xtclsh flow. With the older xflow, you use a opt file.
.ucf	Xilinx constraints file	Contains Xilinx constraints that can be translated to the sdc format and used to drive synthesis.
.vhd	Source files (VHDL)	Design source files in VHDL format. See *VHDL*, on page 597 and Chapter 11, *VHDL Language Support* for details. For information about using VHDL and Verilog files together in a design, see *Using Mixed Language Source Files*, on page 117 of the *User Guide*.
.v	Source files (Verilog)	Design source files in Verilog format. For more information about the Verilog language, and the synthesis commands and attributes you can include, see *Verilog*, on page 598, Chapter 9, *Verilog Language Support*, and Chapter 10, *SystemVerilog Language Support*. For information about using VHDL and Verilog files together in a design, see *Using Mixed Language Source Files*, on page 117 of the *User Guide*.

HDL Source Files

For the Synplify Pro or Synplify Premier products, the HDL source files for a project can be in either VHDL (vhd), Verilog (v), or SystemVerilog (sv) format. However, in Synplify, they must be all in one format.

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

VHDL

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

- numeric_bit
- numeric_std
- std_logic_1164

The synthesis tool also supports the following industry standards in the IEEE libraries:

- std_logic_arith
- std_logic_signed
- std_logic_unsigned

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

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


Copyright © 2013 Synopsys, Inc.

For more information about the VHDL language, and the synthesis commands and attributes you can include, see Chapter 11, *VHDL Language Support*.

Verilog

The Synopsys FPGA synthesis tool supports a synthesizable subset of Verilog 2001 and Verilog 95 (IEEE 1364) and SystemVerilog extensions. For more information about the Verilog language, and the synthesis commands and attributes you can include, see Chapter 9, *Verilog Language Support* and Chapter 10, *SystemVerilog Language Support*.

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

Copyright © 2013 Synopsys, Inc.

Libraries

You can instantiate components from a library, which can be either in Verilog or VHDL. For example, you might have technology-specific or custom IP components in a library, or you might have generic library components. The *installDirectory*/lib directory included with the software contains some component libraries you can use for instantiation.

There are two kinds of libraries you can use:

- Technology-specific libraries that contain I/O pad, macro, or other component descriptions. The lib directory lists these kinds of libraries under vendor sub-directories. The libraries are named for the technology family, and in some cases also include a version number for the version of the place-and-route tool with which they are intended to be used.

  For information about using vendor-specific libraries to instantiate LPMs, PLLs, macros, I/O pads, and other components, refer to the appropriate sections in Chapter 15, *Optimizing for Specific Targets*in the *User Guide*.

- Technology-independent libraries that contain common components. You can have your own library or use the one Synplicity provides. The Synplicity library is a Verilog library of common logic elements, much like the Synopsys® GTECH component library. See *The Synplicity Generic Technology Library*, on page 600 for a description of this library.

Copyright © 2013 Synopsys, Inc.

The Synplicity Generic Technology Library

The synthesis software includes this Verilog library for generic components under the *installDirectory*/lib/generic_technology directory. Currently, the library is only available in Verilog format. The library consists of technology-independent common logic elements, which help the designer to develop technology-independent parts. The library models extract the functionality of the component, but not its implementation. During synthesis, the mappers implement these generic components in implementations that are appropriate to the technology being used.

To use components from this directory, add the library to the project by doing either of the following:

- Add add_file -verilog "$LIB/generic_technology/gtech.v to your `prj` file or type it in the Tcl window.
- In the tool window, click the Add file button, navigate to the *installDirectory*/lib/generic_technology directory and select the gtech.v file.

When you synthesize the design, the tool uses components from this library.

You cannot use the Synplicity generic technology library together with other generic libraries, as this could result in a conflict. If you have your own GTECH library that you intend to use, do not use the Synplicity generic technology library.

Copyright © 2013 Synopsys, Inc.

Output Files

The synthesis tool generates reports about the synthesis run and files that you can use for simulation or placement and routing.The following table describes the output files, categorizing them as either synthesis result and report files, or output files generated as input for other tools.

Extension	File	Description
`.areasrr`	Hierarchical Area Report	Reports area-specific information such as sequential and combinational ATOMS, RAMs, DSPs, and Black Boxes on each module in the design. See *Hierarchical Area Report*, on page 623.
`_cck.rpt`	Constraint Checker Report	Checks the syntax and applicability of the timing constraints in the `sdc` file for your project and generates a report (*projectName*_cck.rpt). See *Constraint Checking Report*, on page 629 for more information.
`_conv.prj`	Project file for new project with converted constraints	Default name for the new project file generated after you run the Project->Vendor Constraints command in a Xilinx design. See *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 in the *User Guide* for details.
`_conv.sdc`	Constraints file (sdc) with translated UCF constraints	Contains converted UCF constraints.When you run the Project->Vendor Constraints command in a Xilinx design, one file is generated for each input Xilinx constraint file. See *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 in the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
`_edif.xdc`	Constraint file for Vivado place and route	Contains design constraints that include Synopsys timing mode and non-timing constraints. This file is used when the synthesis tool generates an `edif` netlist for Vivado place and route. For more information, see *Running Xilinx Vivado Place-and-Route*, on page 852.
`.est`	Module area estimation file	In the Synplify Premier product, area estimation file that contains an area estimate for each design module. This information appears in the RTL and Technology views.
`_est.srr`	Area estimation log	In the Synplify Premier product, logs area estimate information. This is the file displayed by View -> View Estimation Log.
`.fse`	FSM information file	Design-dependent. Contains information about encoding types and transition states for all state machines in the design.
`.info`	Design component files	Design-dependent. Contains detailed information about design components like state machines or ROMs.
`.ncf`	Xilinx constraints file	Generated by the tool after synthesis. It contains the physical constraints to be forward-annotated to Xilinx. See *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 in the *User Guide* for details. Physical constraints that are not forward-annotated are in the output `ucf` file.
`.opt`	Xilinx P&R options file	An editable text file containing place-and-route option information to use when running Xilinx place-and-route with Xflow. For the newer xtclsh flow, a Tcl file is used instead.

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
.pdc	Placement Constraints file	The Import Xilinx Constraints outputs the _placement.pdc and _timing.pdc files. This process translates placement and timing constraints from the Xilinx ISE ncd and ucf files into the FPGA synthesis pdc files.
.pfl	Message Filter criteria	Output file created after filtering messages in the Messages window. See *Updating the projectName.pfl file*, on page 530 in the *User Guide*.
Results file: • .edf • .acf • .vqm	Vendor-specific results file	Results file that contains the synthesized netlist, written out in a format appropriate to the technology and the place-and-route tool you are using. Generally, the format is EDIF, but there could be vendor-specific formats, such as: • acf or vqm for Altera • Xilinx edf format. Specify this file on the Implementation Results panel of the Implementation Options dialog box (*Implementation Results Panel*, on page 203).
run_options.txt	Project settings for implementations	This file is created when a design is synthesized and contains the project settings and options used with the implementations. These settings and options are also processed for displaying the Project Status view after synthesis is run. For details, see *Project Status Tab*, on page 69.

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
`.sap`	Synplify Annotated Properties	This file is generated after the Annotated Properties for Analyst option is selected in the Device panel of the Implementation Options dialog box. After the compile stage, the tool annotates the design with properties like clock pins. You can find objects based on these annotated properties using Tcl Find. For more information, see *Tcl find Command*, on page 1774*Using the Tcl Find Command to Define Collections*, on page 155 in the *User Guide*.
`.sar`	Archive file	Output of the Synopsys FPGA Archive utility in which design project files are stored into a single archive file. Archive files use Synplicity Proprietary Format. See *Archive Project Command*, on page 175 for details on archiving, unarchiving and copying projects.
`_scck.rpt`	Constraint Checker Report (Syntax Only)	Generates a report that contains an overview of the design information, such as, the top-level view, name of the constraints file, if there were any constraint syntax issues, and a summary of clock specifications.
`.srd`	Intermediate mapping files	Used to save mapping information between synthesis runs. You do not need to use these files.
`.srm`	Mapping output files	Output file after mapping. It contains the actual technology-specific mapped design. This is the representation that appears graphically in a Technology view.
`.srp`	RTL design plan view	(*Synplify Premier* tool) partitioned RTL-view file, containing regions and netlists. Double-clicking this file displays a design plan view of your design (same as HDL Analyst ->RTL->Floorplanned View).

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
`.srr`	Synthesis log file	Provides information on the synthesis run, as well as area and timing reports. See *Log File*, on page 608, for more information.
`.srs`	Compiler output file	Output file after the compiler stage of the synthesis process. It contains an RTL-level representation of a design. This is the representation that appears graphically in an RTL view.
`_synplify_sdc`	Logical constraints file for Physical Plus	The logic synthesis input constraints file that can be used and automatically forward-annotated to Physical Plus. Path names and instance names are designated at the gate- or netlist-level in this file and is formatted for Physical Plus.
`synplify.ucf`	UCF constraints file	One file generated by the synthesis tool after you run the Project->Vendor Constraints command in a Xilinx design. After logic synthesis it contains the supported UCF constraints for forward-annotation; after physical synthesis it contains both the supported and unsupported UCF constraints, including any physical constraints that are not being forward-annotated. See *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 in the *User Guide* for details.
`synplify_us.ucf`	UCF constraints file	One file generated by the synthesis tool after you run the Project->Vendor Constraints command in a Xilinx logic synthesis design. It contains the unsupported UCF constraints. See *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 in the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
`synlog folder`	Intermeditate technology mapping files	This folder contains intermediate netlists and log files after technology mapping has been run. Timestamp information is contained in these netlist files to manage jobs with up-to-date checks. For more information, see *Using Up-to-date Checking for Job Management*, on page 509.
`synwork folder`	Intermediate pre-mapping files	This folder contains intermediate netlists and log files after pre-mapping has been run. Timestamp information is contained in these netlist files to manage jobs with up-to-date checks. For more information, see *Using Up-to-date Checking for Job Management*, on page 509.
`.ta`	Customized Timing Report	*Synplify Pro* and *Synplify Premier* Contains the custom timing information that you specify through Analysis->Timing Analyst. See *Analysis Menu*, on page 284, for more information.
`_ta.srm`	Customized mapping output file	*Synplify Pro* and *Synplify Premier* Creates a customized output netlist when you generate a custom timing report with HDL Analyst->Timing Analyst. It contains the representation that appears graphically in a Technology view. See *Analysis Menu*, on page 284 for more information.

Extension	File	Description
`.tap`	Timing Annotated Properties	This file is generated after the Annotated Properties for Analyst option is selected in the Device panel of the Implementation Options dialog box. After the compile stage, the tool annotates the design with timing properties and the information can be analyzed in the RTL view and Design Planner. You can also find objects based on these annotated properties using Tcl Find. For more information, see *Using the Tcl Find Command to Define Collections*, on page 155 in the *User Guide*.
`.tcl`	Xilinx P&R options file	An editable text file containing place-and-route option information to use when running Xilinx place-and-route with the new xtclsh flow. With the older xflow, you use an `opt` file.
`.tlg`	Log file	This log file contains a list of all the modules compiled in the design.
vendor constraint file	Constraints file for forward annotation	Contains synthesis constraints to be forward-annotated to the place-and-route tool. The constraint file type varies with the vendor and the technology. Refer to the vendor chapters for specific information about the constraints you can forward-annotate. Check the Implementation Results dialog (Implementation Options) for supported files. See *Implementation Results Panel*, on page 203.
`.vif`	Verification Interface Format	For certain Altera families, the VIF file uses Tcl commands to forward-annotate sequential optimizations performed during synthesis to any formal verification tool that can read Tcl files. These commands are only available with the Synplify Pro tool. For more information, see *VIF Commands*, on page 1726 and *Overview of the VIF Flow*, on page 893 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Extension	File	Description
. vm . vhm	Mapped Verilog or VHDL netlist	Optional post-synthesis netlist file in Verilog (.vm) or VHDL (.vhm) format. This is a structural netlist of the synthesized design, and differs from the original RTL used as input for synthesis. Specify these files on the Implementation Results dialog box (Implementation Options). See *Implementation Results Panel*, on page 203. Typically, you use this netlist for gate-level simulation, to verify your synthesis results. Some designers prefer to simulate before and after synthesis, and also after place-and-route. This approach helps them to isolate the stage of the design process where a problem occurred. The Verilog and VHDL output files are for functional simulation only. When you input stimulus into a simulator for functional simulation, use a cycle time for the stimulus of 1000 time ticks.
.xdc	Constraint file for Vivado place and route	Contains design constraints that include Synopsys timing mode and non-timing constraints. This file is used when the synthesis tool generates a vm structural Verilog netlist for Vivado place and route. For more information, see *Running Xilinx Vivado Place-and-Route*, on page 852.

Log File

The log file report, located in the implementation directory, is written out in two file formats: text (*projectName*.srr), and HTML with an interactive table of contents (*projectName*.htm and *projectName*_srr.htm) where *projectName* is the name of your project. Select View Log File in HTML in the Options->Project View

Options dialog box to enable viewing the log file in HTML. Select the View Log button in the Project view (*Buttons and Options*, on page 131) to see the log file report.

The log file is written each time you compile or synthesize (compile and map) the design. When you compile a design without mapping it, the log file contains only compiler information. As a precaution, a backup copy of the log file (srr) is written to the backup sub-directory in the Implementation Results directory. Only one backup log file is updated for subsequent synthesis runs.

The log file contains detailed reports on the compiler, mapper, timing, and resource usage information for your design. Errors, notes, warnings, and messages appear in both the log file and on the Messages tab in the Tcl window.

For further details about different sections of the log file, see the following:

For information about...	See...
Compiled files, messages (warnings, errors, and notes), user options set for synthesis, state machine extraction information, including a list of reachable states.	*Compiler Report*, on page 610
Buffers added to clocks in certain supported technologies.	*Clock Buffering Report*, on page 611
Buffers added to nets.	*Net Buffering Report*, on page 611
Timing results. This section of the log file begins with "START TIMING REPORT" section. For the Synplify Pro and Synplify Premier products, if you use the Timing Analyst to generate a custom timing report, its format is the same as the timing report in the log file, but the customized timing report is in a `ta` file.	*Timing Reports*, on page 615
Compile point remapping (Synplify Pro and Synplify Premier products only).	*Compile Point Information*, on page 612

Copyright © 2013 Synopsys, Inc.

For information about...	See...
Resources used by synthesis mapping	*Resource Usage Report*, on page 612
Design changes made as a result of retiming (Synplify Pro and Synplify Premier products only).	*Retiming Report*, on page 613
Design changes made as a result of gated clock conversion (Synplify Pro and Synplify Premier products only).	*Errors, Warnings, Notes, and Messages*, on page 613

Compiler Report

This report starts with the compiler version and date, and includes the following:

- Project information: the top-level module.
- Design information: HDL syntax and synthesis checks, black box instantiations, FSM extractions and inferred RAMs/ROMs. It also includes informational or warning messages about unused ports, removal of redundant logic, and latch inference. See *Errors, Warnings, Notes, and Messages*, on page 613 for details about the kinds of messages.
- Netlist filter information: constant propagation.

Premap Report

This report begins with the pre-mapper version and date, and reports the following:

- File loading times and memory usage
- Clock summary

Mapper Report

This report begins with the mapper version and date, and reports the following:

- Project information: the names of the constraint files, target technology, and attributes set in the design.
- Design information such as flattened instances, extraction of counters, FSM implementations, clock nets, buffered nets, replicated logic, RTL optimizations, and informational or warning messages. See *Errors, Warnings, Notes, and Messages*, on page 613 for details about the kinds of messages.

Clock Buffering Report

This section of the log file reports any clocks that were buffered. For example:

```
Clock Buffers:
Inserting Clock buffer for port clock0,TNM=clock0
```

Net Buffering Report

Net buffering reports are generated for most all of the supported FPGAs and CPLDs. This information is written in the log file, and includes the following information:

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

Copyright © 2013 Synopsys, Inc.

Example: Net Buffering Report

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

Compile Point Information

In the Synplify Pro and Synplify Premier software, the Summary of Compile Points section of the log file (*projectName*.srr) lists each compile point, together with an indication of whether it was remapped, and, if so, why. Also, a timing report is generated for each compile point located in its respective results directories in the Implementation Directory. The compile point is the top-level design for this report file.

For more information on compile points and the compile-point synthesis flow, see *Synthesizing Compile Points*, on page 1109of the *User Guide.*

Timing Section

A default timing report is written to the log file (*projectName*.srr) in the "START OF TIMING REPORT" section. See *Timing Reports*, on page 615, for details.

For certain device technologies in the Synplify Pro and Synplify Premier tools, you can use the Timing Analyst to generate additional timing reports for point-to-point analysis (see *Analysis Menu*, on page 284). Their format is the same as the timing report.

Resource Usage Report

A resource usage report is written in the log file each time you compile or synthesize. The format of the resource usage report varies, depending on the architecture you are using. The report provides the following information:

- The total number of cells, and the number of combinational and sequential cells in the design

- The number of clock buffers and I/O cells
- Details of how many of each type of cell in the design

Retiming Report

Whenever retiming is enabled in the Synplify Pro or Synplify Premier tools, a retiming report is added to the log file (*projectName*.srr). It includes information about the design changes made as a result of retiming, such as the following:

- The number of flip-flops added, removed, or modified because of retiming. Flip-flops modified by retiming have a _ret suffix added to their names.
- Names of the flip-flops that were *moved* by retiming and no longer exist in the Technology view.
- Names of the flip-flops *created* as result of the retiming moves, that did not exist in the RTL view.
- Names of the flip-flops *modified* by retiming; for example, flip-flops that are in the RTL and Technology views, but have different fanouts because of retiming.

Errors, Warnings, Notes, and Messages

Throughout the log file, interactive error, note, warning, and informational messages appear.

- Error messages begin with "`@E:`"
- Warning messages begin with "`@W:`"
- Notes begin with "`@N:`"
- Advisories begin with "`@A:`"
- Informational messages begin with "`@I:`"

Copyright © 2013 Synopsys, Inc.

Colors distinguish different types of messages:

Color	Message Type	Example
Blue	Information (@I) Notes (@N)	@I: :"C:\designs\Designs6\module1\mychip.v” @N: CL201 |Trying to extract state machine for ...
Brown	Warnings (@W)	@W: CG146 |Creating black_box for empty module ...
Red	Errors(@E)	@E: CS106 |Reference to undefined module ...

In the Synplify Pro and Synplify Premier software, the errors, warnings, and notes are also displayed in the Messages tab of the Output window. To get help on a message, you can single click on the numeric ID at the beginning of the message in the log file or Messages window. To crossprobe to the corresponding HDL source code, single click on the source file name.

Copyright © 2013 Synopsys, Inc.

Timing Reports

Timing results can be written to one or more of the following files:

`.srr` or `.htm`	Log file that contains a default timing report. To find this information, after synthesis completes, open the log file (View -> Log File), and search for START OF TIMING REPORT.
`.ta`	Timing analysis file that contains timing information based on the parameters you specify in the stand-alone Timing Analyst (Analysis->Timing Analyst). This feature is only available in the Synplify Pro and Synplify Premier tools, for supported technologies.
*designName*_async_clk .rpt.scv	Asynchronous clock report file that is generated when you enable the related option in the stand-alone Timing Analyzer (Analysis->Timing Analyst). This feature is only available in the Synplify Pro and Synplify Premier tools, and only for supported technologies. This report can be displayed in a spreadsheet tool and contains information for paths that cross between multiple clock groups. See *Asynchronous Clock Report*, on page 622 for details on this report.

The timing reports in the `srr/htm` and `ta` files have the following sections:

- Timing Report Header, on page 616
- Performance Summary, on page 616
- Clock Relationships, on page 618
- Interface Information, on page 619
- Detailed Clock Report, on page 620
- Asynchronous Clock Report, on page 622

Copyright © 2013 Synopsys, Inc.

Timing Report Header

The timing report header lists the date and time, the name of the top-level module, the number of paths requested for the timing report, and the constraint files used.

```
00056 ##### START TIMING REPORT #####
00057 # Timing Report written on Fri Sep 06 13:38:15 2002
00058 #
00059
00060
00061 Top view:               mod2
00062 Paths requested:        5
00063 Constraint File(s):
00064 @N| This timing report estimates place and route data. Please look
00065 @N| Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF
```

You can control the size of the timing report by choosing Project -> Implementation Options, clicking the Timing Report tab of the panel, and specifying the number of start/end points and the number of critical paths to report. See *Timing Report Panel*, on page 205, for details.

Performance Summary

The Performance Summary section of the timing report reports estimated and requested frequencies for the clocks, with the clocks sorted by negative slack. The timing report has a different section for detailed clock information (see *Detailed Clock Report*, on page 620). The Performance Summary lists the following information for each clock in the design:

Performance Summary Column	Description
Starting Clock	Clock at the start point of the path. If the clock name is system, the clock is a collection of clocks with an undefined clock event. Rising and falling edge clocks are reported as one clock domain.
Requested/Estimated Frequency	Target frequency goal /estimated value after synthesis. See *Cross-Clock Path Timing Analysis*, on page 618 for information on how cross-clock path slack is reported.
Requested/Estimated Period	Target clock period/estimated value after synthesis.

Performance Summary Column	Description
Slack	Difference between estimated and requested period. See *Cross-Clock Path Timing Analysis*, on page 618 for information on how cross-clock path slack is reported.
Clock Type	The type of clock: inferred, declared, derived or system. For more information, see *Clock Types*, on page 617.
Clock Group	Name of the clock group that a clock belongs.

The synthesis tool does not report inferred clocks that have an unreasonable slack time. Also, a real clock might have a negative period. For example, suppose you have a clock going to a single flip-flop, which has a single path going to an output. If you specify an output delay of –1000 on this output, then the synthesis tool cannot calculate the clock frequency. It reports a negative period and no clock.

Clock Types

The synthesis timing reports include the following types of clocks:

- Declared Clocks

  User-defined clocks specified in the constraint file.

- Inferred Clocks

  These are clocks that the synthesis timing engine finds during synthesis, but which have not been constrained by the user. The tool assigns the default global frequency specified for the project to these clocks. The tool does not forward-annotate inferred clocks to the `synplicity.ucf` file. If you want to forward-annotate them to the `synplicity.ucf` file as P&R constraints, you must define these clocks in the top-level constraint file.

- Derived Clocks

  These are clocks that the synthesis tool identifies from a clock divider/multiplier such as DCM. The tool reports these clocks for timing purposes only and does not forward-annotate them in the `synplicity.ucf` file. The P&R tool derives these clocks from the master top-level clock, so the tool only forward-annotates the master clock

Copyright © 2013 Synopsys, Inc.

frequency to the `synplicity.ucf` file, to be used as a P&R clock constraint.

- System Clock

  The system clock is the delay for the combinatorial path. Additionally, a system clock can be reported if there are sequential elements in the design for a clock network that cannot be traced back to a clock. Also, the system clock can occur for unconstrained I/O ports. You must investigate these conditions.

Clock Relationships

For each pair of clocks in the design, the Clock Relationships section of the timing report lists both the required time (constraint) and the worst slack time for each of the intervals rise to rise, fall to fall, rise to fall, and fall to rise. See *Cross-Clock Path Timing Analysis*, on page 618 for details about cross-clock paths.

This information is provided for the paths between related clocks (that is, clocks in the same clock group). If there is no path at all between two clocks, then that pair is not reported. If there is no path for a given pair of edges between two clocks, then an entry of No paths appears.

For information about how these relationships are calculated, see *Clock Groups*, on page 522. For tips on using clock groups, see *Defining Other Clock Requirements*, on page 185 in the *User Guide*.

```
Clock Relationships
*******************

Clocks             |    rise  to  rise    |    fall  to  fall    |    rise  to  fall    |    fall  to  rise
-------------------------------------------------------------------------------------------------------------
Starting  Ending   |  constraint  slack   |  constraint  slack   |  constraint  slack   |  constraint  slack
-------------------------------------------------------------------------------------------------------------
clk1      clk1     |  25.000      15.943  |  25.000      17.764  |  No paths    -       |  No paths    -
clk1      clk2     |  1.000       -9.430  |  No paths    -       |  No paths    -       |  1.000       -1.531
clk2      clk1     |  No paths    -       |  1.000       -0.811  |  1.000       -1.531  |  No paths    -
clk2      clk2     |  8.000       0.764   |  8.000       -1.057  |  No paths    -       |  6.000       2.814
clk3      clk3     |  No paths    -       |  10.000      0.943   |  No paths    -       |  No paths    -
=============================================================================================================
```

Cross-Clock Path Timing Analysis

The following describe how the timing analyst calculates cross-clock path frequency and slack.

Cross-Clock Path Frequency

For each data path, the tool estimates the highest frequency that can be set for the clock(s) without a setup violation. It finds the largest scaling factor that can be applied to the clock(s) without causing a setup violation. If the start clock is not the same as the end clock, it scales both by the same factor.

scale = (*minimum time period* -(-*current slack*))/*minimum time period*

It assumes all other delays in the setup calculation (e.g., uncertainty) are fixed.

It applies relevant multicycle constraints to the setup calculation.

The estimated frequency for a clock is the minimum frequency over all paths that start or end on that clock, with the following exceptions:

- The tool does not consider paths between the system clock and another clock to estimate frequency.
- It considers paths with a path delay constraint to be asynchronous, and does not use them to estimate frequency.
- It considers paths between clocks in different domains to be asynchronous, and does not use them to estimate frequency.

Slack for Cross-Clock Paths

The slack reported for a cross-clock path is the worst slack for any path that starts on that clock. Note that this differs from the estimated frequency calculation, which is based on the worst slack for any path starting or ending on that clock.

Interface Information

The interface section of the timing report contains information on arrival times, required times, and slack for the top-level ports. It is divided into two subsections, one each for Input Ports and Output Ports. Bidirectional ports are listed under both. For each port, the interface report contains the following information.

Copyright © 2013 Synopsys, Inc.

Port parameter	**Description**
Port Name	Port name.
Starting Reference Clock	The reference clock.
User Constraint	The input/output delay. If a port has multiple delay records, the report contains the values for the record with the worst slack. The reference clock corresponds to the worst slack delay record.
Arrival Time	Input ports: define_input_delay, or default value of 0. Output ports: path delay (including clock-to-out delay of source register). For purely combinational paths, the propagation delay is calculated from the driving input port.
Required Time	Input ports: clock period – (path delay + setup time of receiving register + define_reg_input_delay value). Output ports: clock period – define_output_delay. Default value of define_output_delay is 0.
Slack	Required Time – Arrival Time

Detailed Clock Report

Each clock reported in the performance summary also has a detailed clock report section in the timing report. The clock reports are listed in order of negative slack.

General Critical Path Information

This section contains general information about the most critical paths in the design.

Clock Information	**Description**
N most critical start points	Start points can be input ports or registers. If the start point is a register, you see the starting pin in the report. To change the number of start points reported, choose Project -> Implementation Options, and set the number on the Timing Report panel.

Clock Information	Description
N most critical end points	End points can be output ports or registers. If the end point is a register, you see the ending pin in the report. To change the number of end points reported, select Project -> Implementation Options, and set the number on the Timing Report panel.
N worst path information (see the next table for details)	Starting with the most critical path, the worst path Information sections contain details of the worst paths in the design. Paths from clock A to clock B are reported as critical paths in the section for clock A. You can change the number of critical paths on the Timing Report panel of the Implementation Options dialog box.

Worst Path Information

For each critical path, the timing report has a detailed description. It starts with a summary of the information and is followed by a detailed pin-by-pin report. The summary reports information like requested period, actual period, start and end points, and logic levels. Note that the requested period here is period -route delay, while the requested period in the Performance Summary (*Performance Summary*, on page 616) is just the clock period.

The detailed path report uses this format: Output pin – Net – Input pin – Output pin – Net – Input pin. The following table describes the critical path information reported:

Critical path information	Description
Instance/Net Name	Technology view names for the instances and nets in the critical path
Type	Type of cell
Pin Name	Name of the pin
Pin Dir	Pin direction
Delay	The delay value.
Arrival Time	Clock delay at the source + the propagation delay through the path
Fan Out	Number of fanouts for the point in the path

Copyright © 2013 Synopsys, Inc.

Asynchronous Clock Report

You can generate a report for paths that cross between clock groups using the stand-alone Timing Analyst (Analysis->Timing Analyst, Generate Asynchronous Clock Report check box). Generally, paths in different clock groups are automatically handled as false paths. This option provides a file that contains information on each of the paths and can be viewed in a spreadsheet tool. To display the CSV-format report:

1. Locate the file in your results directory *projectName*_async_clk.rpt.csv .
2. Open the file in your spreadsheet tool.

Column	Description
Index	Path number.
Path Delay	Delay value as reported in standard timing (`ta`) file.
Logic Levels	Number of logic levels in the path (such as LUTs, cells, and so on) that are between the start and end points.
Types	Cell types, such as LUT, logic cell, and so on.
Route Delay	As reported for each path in `ta`
Source Clock	Start clock.
Destination Clock	End clock.
Data Start Pin	Sequential device output pin at start of path.
Data End Pin	Setup check pin at destination.

	A	B	C	D	E	F	G	H	I
1	Index	Path Delay	Logic Levels	Types	Route Delay	Source Clock	Destination Clock	Data Start Pin	Data End Pin
2	1	1.533	1	LUT1_L	0.632	Clock_A	Clock_B	reg_A.Q	reg_B.D
3	2	2.176	1	LUT1_L	0.884	Clock_B	Clock_C	reg_B.Q	reg_C.D
4									

Copyright © 2013 Synopsys, Inc.

Hierarchical Area Report

An area report is created during synthesis which contains the percentage utilization for elements in the design, as well as, total sequential utilization for elements of specific modules. For instance, elements can include sequential, combinational, or memory elements. They can also include the following types of technology-specific elements for ROMs, I/O pads, DSPs, or black boxes.

This report is only generated for Altera and Xilinx devices, for which technology-specific area information is reflected in the output depending upon the specified device. The report is written to the *projectName*.areasrr file. You can view the file with the log viewer or any text editor.

Altera Hierarchical Area Report

```
#### START OF AREA REPORT #####[
Part:       EP3SE50FC780-2 (Altera)


-----------------------------------------------------------------------
########   Utilization report for  Top level view:   eight_bit_uc
=======================================================================


SEQUENTIAL ATOMS
****************

Name          Total elements     Utilization     Notes
-------------------------------------------------------
REGISTERS     181                100 %
=======================================================
Total SEQUENTIAL ATOMS in block eight_bit_uc:181 (39.09 % Utilization)


COMBINATIONAL ATOMS
*******************

Name                        Total elements     Utilization     Notes
-------------------------------------------------------------------
ATOMS                       198                100 %
SHARED ARITHMATIC MODE      0                  0.0 %
COMB ARITHMETIC MODE        20                 100 %
===================================================================
Total COMBINATIONAL ATOMS in block eight_bit_uc:218 (47.08 % Utilization)
```


Copyright © 2013 Synopsys, Inc.

```
RAMS
****

Name          Total elements     Number of bits     Utilization
---------------------------------------------------------------------
SYNC RAMS     1                  1                  100 %
LPMs          1                  1                  100 %
=====================================================================
Total RAMS in the block eight_bit_uc:2 (0.43 % Utilization)

DSPs
****

Name     Total elements     Utilization     Notes
-------------------------------------------------
MACs     0                  0.0 %
=================================================
Total DSPs in the block eight_bit_uc:0 (0.00 % Utilization)

Black Boxes
***********

Name            Total elements     Utilization     Notes
-------------------------------------------------------
BLACK BOXES     1                  100 %
=======================================================
Total Black Boxes in the block eight_bit_uc:1 (0.22 % Utilization)

-------------------------------------------------------
########   Utilization report for  cell:   alu   ########
Instance path:   eight_bit_uc.alu
=======================================================

SEQUENTIAL ATOMS
****************

Name          Total elements     Utilization     Notes
-------------------------------------------------------
REGISTERS     10                 5.52 %
=======================================================
Total SEQUENTIAL ATOMS in the block eight_bit_uc.alu:10 (2.16 % Utilization)

COMBINATIONAL ATOMS
*******************

Name                       Total elements     Utilization     Notes
---------------------------------------------------------------------
ATOMS                      38                 19.2 %
SHARED ARITHMATIC MODE     0                  0.0 %
COMB ARITHMETIC MODE       10                 50. %
=====================================================================
```

```
Total COMBINATIONAL ATOMS in the block eight_bit_uc.alu:48 (10.37 % Utilization)

RAMS
****

Name          Total elements     Number of bits     Utilization
-----------------------------------------------------------------------
SYNC RAMS     0                  0                  0.0 %
LPMs          0                  0                  0.0 %
=======================================================================
Total RAMS in the block eight_bit_uc.alu:0 (0.00 % Utilization)


DSPs
****

Name     Total elements     Utilization     Notes
-----------------------------------------------
MACs     0                  0.0 %
===============================================
Total DSPs in the block eight_bit_uc.alu:0 (0.00 % Utilization)


Black Boxes
***********
Name            Total elements     Utilization     Notes
-------------------------------------------------------
BLACK BOXES     0                  0.0 %
=======================================================
Total Black Boxes in the block eight_bit_uc.alu:0 (0.00 % Utilization)


-------------------------------------------------------------
########   Utilization report for  cell:   data_mux   ########
Instance path:   eight_bit_uc.data_mux
=============================================================
```

... (All modules of the design are listed similarly.)

```
##### END OF AREA REPORT #####]
```


Copyright © 2013 Synopsys, Inc.

Xilinx Hierarchical Area Report

```
#### START OF AREA REPORT #####[

Part:               XC5VLX30FF324-2 (Xilinx)

---------------------------------------------------------------------
########   Utilization report for  Top level view:   eight_bit_uc
=====================================================================

SEQUENTIAL ELEMENTS
*******************

Name          Total elements     Utilization     Notes
-----------------------------------------------------
REGISTERS     170                100 %
LATCHES       0                  0.0 %
=====================================================
Total SEQUENTIAL ELEMENTS in the block eight_bit_uc:170 (35.71 % Utilization)


COMBINATIONAL LOGIC
*******************

Name      Total elements     Utilization     Notes
-------------------------------------------------
LUTS      231                100 %
MUXCY     18                 100 %
XORCY     19                 100 %
DSP48     0                  0.0 %
=================================================
Total COMBINATIONAL LOGIC in the block eight_bit_uc:268 (56.30 % Utilization)


MEMORY ELEMENTS
***************

Name          Total elements     Utilization     Notes
-----------------------------------------------------
SYNC RAMS     0                  0.0 %
=====================================================
Total MEMORY ELEMENTS in the block eight_bit_uc:0 (0.00 % Utilization)


Name     Total elements     Number of bits     Utilization     Notes
----------------------------------------------------------------------
ROMS     0                  0                  0.0 %
======================================================================
Total  in the block eight_bit_uc:0 (0.00 % Utilization)
```

```
Distributed RAM
***************


Name                Total elements     Number of LUTs     Utilization
-------------------------------------------------------------------
DISTRIBUTED RAM     4                  4                  100 %
===================================================================
Total Distributed RAM in the block eight_bit_uc:4 (0.84 % Utilization)


IO PADS
*******

Name     Total elements     Utilization     Notes
-------------------------------------------------
PADS     26                 100 %
=================================================
Total IO PADS in the block eight_bit_uc:26 (5.46 % Utilization)


-----------------------------------------------------------
########   Utilization report for  cell:   alu   ########
Instance path:   eight_bit_uc.alu
===========================================================


SEQUENTIAL ELEMENTS
*******************

Name          Total elements     Utilization     Notes
-----------------------------------------------------
REGISTERS     10                 5.88 %
LATCHES       0                  0.0 %
=====================================================
Total SEQUENTIAL ELEMENTS in the block eight_bit_uc.alu:10 (2.10 % Utilization)


COMBINATIONAL LOGIC
*******************

Name      Total elements     Utilization     Notes
-------------------------------------------------
LUTS      37                 16. %
MUXCY     8                  44.4 %
XORCY     9                  47.4 %
DSP48     0                  0.0 %
=================================================
Total COMBINATIONAL LOGIC in the block eight_bit_uc.alu:54 (11.34 % Utilization)
```


Copyright © 2013 Synopsys, Inc.

```
MEMORY ELEMENTS
***************

Name          Total elements     Utilization     Notes
-----------------------------------------------------
SYNC RAMS     0                  0.0 %
=====================================================
Total MEMORY ELEMENTS in the block eight_bit_uc.alu:0 (0.00 % Utilization)


Name     Total elements     Number of bits     Utilization     Notes
-------------------------------------------------------------------
ROMS     0                  0                  0.0 %
===================================================================
Total  in the block eight_bit_uc.alu:0 (0.00 % Utilization)


Distributed RAM
***************

Name                Total elements     Number of LUTs     Utilization
-------------------------------------------------------------------
DISTRIBUTED RAM     0                  0                  0.0 %
===================================================================
Total Distributed RAM in the block eight_bit_uc.alu:0 (0.00 % Utilization)



-------------------------------------------------------------
########   Utilization report for  cell:   data_mux   ########
Instance path:   eight_bit_uc.data_mux
=============================================================
```

... (All modules of the design are listed similarly.)

```
##### END OF AREA REPORT #####]
```

Constraint Checking Report

Use the Run->Constraint Check command to generate a report on the constraint files in your project. The *projectName*_cck.rpt file provides information such as invalid constraint syntax, constraint applicability, and any warnings or errors. For details about running Constraint Check, see *Tcl Syntax Guidelines for Constraint Files*, on page 123 in the *User Guide*.

This section describes the following topics:

- Reporting Details, on page 629
- Inapplicable Constraints, on page 630
- Applicable Constraints With Warnings, on page 631
- Sample Constraint Check Report, on page 632

Reporting Details

This constraint checking file reports the following:

- 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

It contains the following sections:

Summary	Statement which summarizes the total number of issues defined as an error or warning (x) out of the total number of constraints with issues (y) for the total number of constraints (z) in the sdc file. `Found <x> issues in <y> out of <z> constraints`
Clock Relationship	Standard timing report clock table, without slack.
Unconstrained Start/End Points	Lists I/O ports that are missing input/output delays.

Copyright © 2013 Synopsys, Inc.

Unapplied constraints	Constraints that cannot be applied because objects do not exist or the object type check is not valid. See *Inapplicable Constraints*, on page 630 for more information.
Applicable constraints with issues	Constraints will be applied either fully or partially, but there might be issues that generate warnings which should be investigated, such as some objects/collections not existing. Also, whenever at least one object in a list of objects is not specified with a valid object type a warning is displayed. See *Applicable Constraints With Warnings*, on page 631 for more information.
Constraints with matching wildcard expressions	Lists constraints or collections using wildcard expressions up to the first 1000, respectively.

Inapplicable Constraints

Refer to the following table for constraints that were not applied because objects do not exist or the object type check was not valid:

For these constraints...	Objects must be...
Attributes	Valid definitions
define_clock	• Ports • Nets • Pins • Registers • Instantiated buffers
define_clock_delay	Clocks
define_compile_point	• Region • View
define_current_design	v:*view*

For these constraints...	Objects must be...
define_false_path define_multicycle_path define_path_delay	For -to or -from objects: • i:sequential instances • p:ports • i:black boxes For -through objects • n:nets • t:hierarchical ports • t:pins
define_multicycle_path	Specified as a positive integer
define_input_delay	• Input ports • bidir ports
define_output_delay	• Output ports • Bidir ports
define_reg_input_delay define_reg_output_delay	Sequential instances

Applicable Constraints With Warnings

The following table lists reasons for warnings in the report file:

For these constraints...	Objects must be...
define_clock	• Ports • Nets • Pins • Registers • Instantiated buffers
define_clock_delay	A single object. Multiple objects are not supported.
define_compile_point	A single object. Multiple objects are not supported.
define_current_design	v:*view*

Copyright © 2013 Synopsys, Inc.

For these constraints...	Objects must be...
define_false_path define_multicycle_path define_path_delay	For -to or -from objects: • i:sequential instances • p:ports • i:black boxes For -through objects: • n:nets • t:hierarchical ports • t:pins
define_input_delay	A single object. Multiple objects are not supported.
define_output_delay	A single object. Multiple objects are not supported.
define_reg_input_delay define_reg_output_delay	A single object. Multiple objects are not supported.

Sample Constraint Check Report

The following is a sample report generated by constraint checking:

```
# Synopsys Constraint Checker, version maprc, Build 658R, built Aug 25 2011
# Copyright (C) 1994-2011, Synopsys, Inc.

# Written on Thu Oct 20 09:42:22 2011
##### DESIGN INFO #######################################################

Top View:                "decode_top"
Constraint File(s):      "C:\timing_88\FPGA_decode_top.sdc"
##### SUMMARY ############################################################

Found 3 issues in 2 out of 27 constraints
```

```
##### DETAILS ##########################################################

Clock Relationships
*******************

Starting  Ending  |  rise to rise  |  fall to fall  |  rise to fall  |  fall to rise
--------------------------------------------------------------------------------
clk2x     clk2x   |  24.000        |  24.000        |  12.000        |  12.000
--------------------------------------------------------------------------------
clk2x     clk     |  24.000        |  No paths      |  No paths      |  12.000
clk       clk2x   |  24.000        |  No paths      |  12.000        |  No paths
clk       clk     |  48.000        |  No paths      |  No paths      |  No paths
================================================================================
Note:
'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in
different clock groups

Unconstrained Start/End Points
******************************
p:test_mode

Inapplicable constraints
************************

define_false_path -from p:next_synd -through i:core.tab1.ram_loader
@E:|object "i:core.tab1.ram_loader" does not exist
@E:|object "i:core.tab1.ram_loader" is incorrect type; "-through" objects must be of
type net (n:), or pin (t:)

Applicable constraints with issues
**********************************

define_false_path -from {core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.omega_inst.omega_tmp_d_lch[7:0]}
@W:|object "core.decoder.root_mult*.root_prod_pre[*]" is missing qualifier which may
result in undesired results; "-from" objects must be of type clock (c:), inst (i:), port
(p:), or pin (t:)

Constraints with matching wildcard expressions
**********************************************

define_false_path -from {core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.omega_inst.omega_tmp_d_lch[7:0]}
@N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]
```


Copyright © 2013 Synopsys, Inc.

```
define_false_path -from {i:core.decoder.*.root_prod_pre[*]} -to
{i:core.decoder.t_*_[*]}
@N:|expression "core.decoder.*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]
@N:|expression "core.decoder.t_*_[*]" applies to objects:
core.decoder.t_20_[7:0]
core.decoder.t_19_[7:0]
core.decoder.t_18_[7:0]
core.decoder.t_17_[7:0]
core.decoder.t_16_[7:0]
core.decoder.t_15_[7:0]
core.decoder.t_14_[7:0]
core.decoder.t_13_[7:0]
core.decoder.t_12_[7:0]
core.decoder.t_11_[7:0]
core.decoder.t_10_[7:0]
core.decoder.t_9_[7:0]
core.decoder.t_8_[7:0]
core.decoder.t_7_[7:0]
core.decoder.t_6_[7:0]
core.decoder.t_5_[7:0]
core.decoder.t_4_[7:0]
core.decoder.t_3_[7:0]
core.decoder.t_2_[7:0]
core.decoder.t_1_[7:0]
core.decoder.t_0_[7:0]

define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.err[7:0]}
N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.omega_inst.deg_omega[4:0]}
@N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.omega_inst.omega_tmp[0:7]}
@N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.root[7:0]}
@N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]
```

Copyright © 2013 Synopsys, Inc.

```
define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.root_inst.count[3:0]}
N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.root_inst.q_reg[7:0]}
@N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult*.root_prod_pre[*]} -to
{i:core.decoder.root_inst.q_reg_d_lch[7:0]}
@N:|expression "core.decoder.root_mult*.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult.root_prod_pre[*]} -to
{i:core.decoder.error_inst.den[7:0]}
@N:|expression "core.decoder.root_mult.root_prod_pre[*]" applies to objects:
core.decoder.root_mult.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.root_mult1.root_prod_pre[*]} -to
{i:core.decoder.error_inst.num1[7:0]}
@N:|expression "core.decoder.root_mult1.root_prod_pre[*]" applies to objects:
core.decoder.root_mult1.root_prod_pre[14:0]

define_false_path -from {i:core.decoder.synd_reg_*_[7:0]} -to
{i:core.decoder.b_*_[7:0]}
@N:|expression "core.decoder.synd_reg_*_[7:0]" applies to objects:
core.decoder.un1_synd_reg_0_[7:0]
core.decoder.synd_reg_20_[7:0]
core.decoder.synd_reg_19_[7:0]
core.decoder.synd_reg_18_[7:0]
core.decoder.synd_reg_17_[7:0]
core.decoder.synd_reg_16_[7:0]
core.decoder.synd_reg_15_[7:0]
core.decoder.synd_reg_14_[7:0]
core.decoder.synd_reg_13_[7:0]
core.decoder.synd_reg_12_[7:0]
core.decoder.synd_reg_11_[7:0]
core.decoder.synd_reg_10_[7:0]
core.decoder.synd_reg_9_[7:0]
core.decoder.synd_reg_8_[7:0]
core.decoder.synd_reg_7_[7:0]
core.decoder.synd_reg_6_[7:0]
core.decoder.synd_reg_5_[7:0]
core.decoder.synd_reg_4_[7:0]
core.decoder.synd_reg_3_[7:0]
core.decoder.synd_reg_2_[7:0]
```


Copyright © 2013 Synopsys, Inc.

```
core.decoder.synd_reg_1_[7:0]
@N:|expression "core.decoder.b_*_[7:0]" applies to objects:
core.decoder.un1_b_0_[7:0]
core.decoder.b_calc.un1_lambda_0_[7:0]
core.decoder.b_20_[7:0]
core.decoder.b_19_[7:0]
core.decoder.b_18_[7:0]
core.decoder.b_17_[7:0]
core.decoder.b_16_[7:0]
core.decoder.b_15_[7:0]
core.decoder.b_14_[7:0]
core.decoder.b_13_[7:0]
core.decoder.b_12_[7:0]
core.decoder.b_11_[7:0]
core.decoder.b_10_[7:0]
core.decoder.b_9_[7:0]
core.decoder.b_8_[7:0]
core.decoder.b_7_[7:0]
core.decoder.b_6_[7:0]
core.decoder.b_5_[7:0]
core.decoder.b_4_[7:0]
core.decoder.b_3_[7:0]
core.decoder.b_2_[7:0]
core.decoder.b_1_[7:0]
core.decoder.b_0_[7:0

Library Report
**************

# End of Constraint Checker Report
```

Gated Clock Conversion Report

When using the Clock Conversion option in the Synplify Pro or Synplify Premier tool, both the sequential element that could not be converted and the reason why the conversion did not occur are reported.

For more information about using gated clocks, see *Working with Gated Clocks*, on page 466.

CHAPTER 9

Verilog Language Support

This chapter discusses Verilog support in the synthesis tool. SystemVerilog support is described separately, in Chapter 10, *SystemVerilog Language Support*. This chapter includes the following topics:

- Support for Verilog Language Constructs, on page 639
- Verilog 2001 Support, on page 642
- Verilog Synthesis Guidelines, on page 667
- Verilog Module Template, on page 685
- Scalable Modules, on page 686
- Built-in Gate Primitives, on page 689
- Combinational Logic, on page 690
- Sequential Logic, on page 695
- Verilog State Machines, on page 705
- Verilog Guidelines for RAM Inference, on page 709
- Verilog Coding for Altera RAM Inference, on page 717
- Verilog Coding for Xilinx RAM Inference, on page 738
- RAM Instantiation with SYNCORE, on page 745
- ROM Inference, on page 746
- Instantiating Black Boxes in Verilog, on page 749
- PREP Verilog Benchmarks, on page 750

Copyright © 2013 Synopsys, Inc.

- Hierarchical or Structural Verilog Designs, on page 751
- Verilog Attribute and Directive Syntax, on page 758

Support for Verilog Language Constructs

This section describes support for various Verilog language constructs:

- Supported and Unsupported Verilog Constructs, on page 639
- Ignored Verilog Language Constructs, on page 641

Supported and Unsupported Verilog Constructs

The following table lists the supported and unsupported Verilog constructs. If the tool encounters an unsupported construct, it generates an error message and stops.

Supported Verilog Constructs	Unsupported Verilog Constructs
Net types: wire, tri, tri0, tri1, supply1, supply0	Net types: trireg, triand, trior, wand, wor, charge strength
Register types: • reg, integer, time (64-bit reg) • arrays of reg	Register types: real
Gate primitive, module, and macromodule instantiations	Built-in unidirectional and bidirectional switches, and pull-up/pull-down
always blocks, user tasks, user functions	Named events and event triggers
inputs, outputs, and inouts to a module	UDPs and specify blocks
All operators -, -, *, /, %, <, >, <=, >=, ==, !=, ===, !==, &&, ||, !, ~, &, ~&, |, ~|, ^~, ~^, ^, <<, >>, ?:, { }, {{ }} (See *Verilog Operator Support*, on page 640 for additional details.)	Net names: force, release, and hierarchical net names (for simulation only)
Procedural statements: assign, if-else-if, case, casex, casez, for, repeat, while, forever, begin, end, fork, join	Procedural statements: deassign, wait

Copyright © 2013 Synopsys, Inc.

Procedural assignments: • Blocking assignments = • Non-blocking assignments <=	Do not use = with <= for the same register. Use parameter override: # and defparam (down one level of hierarchy only).
Continuous assignments	
Compiler directives: 'define, 'ifdef, 'ifndef, 'else, 'elsif, 'endif, 'include, 'undef	
Miscellaneous: • Parameter ranges • Local declarations to begin-end block • Variable indexing of bit vectors on the left and right sides of assignments	

Verilog Operator Support

For Verilog operator support:

- the / and % operators are supported for compile-time constants and constant powers of two.
- when an equality (==) or inequality (!=) operator includes unknown bits (e.g., A==4'b10x1 or A!=4'b111z), the Synopsys FPGA Verilog compiler assumes that the output is always False. This assumption contradicts the LRM which states that the output should be x (unknown) and can result in a possible simulation mismatch.

Verilog Net Type Support

For Verilog net type support:

- While the Synopsys FPGA Verilog compiler allows the use of tri0 and tri1 nets, these nets are treated as wire net types during synthesis, and any variable declared as a tri0 or tri1 net type behaves as a wire net type. A warning is issued in the log file alerting you that a tri0 or tri1 variable is being treated as a wire net type and that a simulation mismatch is possible.

Copyright © 2013 Synopsys, Inc.

Ignored Verilog Language Constructs

When it encounters certain Verilog constructs, the tool ignores them and continues the synthesis run. The following constructs are ignored:

- delay, delay control, and drive strength
- scalared, vectored
- initial block
- Compiler directives (except for 'define, 'ifdef, 'ifndef, 'else, 'elsif, 'endif, 'include, and 'undef, which are supported)
- Calls to system tasks and system functions (they are only for simulation)

Copyright © 2013 Synopsys, Inc.

Verilog 2001 Support

You can choose the Verilog standard to use for a project or given files within a project: Verilog '95 or Verilog 2001. See *File Options Popup Menu Command*, on page 381 and *Setting Verilog and VHDL Options*, on page 239 of the *User Guide*. The synthesis tool supports the following Verilog 2001 features:

Feature	Description
Combined Data, Port Types (ANSI C-style Modules)	Module data and port type declarations can be combined for conciseness.
Comma-separated Sensitivity List	Commas are allowed as separators in sensitivity lists (as in other Verilog lists).
Wildcards (*) in Sensitivity List	Use @* or @(*) to include all signals in a procedural block to eliminate mismatches between RTL and post-synthesis simulation.
Signed Signals	Data types net and reg, module ports, integers of different bases and signals can all be signed. Signed signals can be assigned and compared. Signed operations can be performed for vectors of any length.
Inline Parameter Assignment by Name	Assigns values to parameters by name, inline.
Constant Function	Builds complex values at elaboration time.
Configuration Blocks	Specifies a set of rules that defines the source description applied to an instance or module.
Localparams	A constant that cannot be redefined or modified.
$signed and $unsigned Built-in Functions	Built-in Verilog 2001 function that converts types between signed and unsigned.
$clog2 Constant Math Function	Returns the value of the log base-2 for the argument passed.
Generate Statement	Creates multiple instances of an object in a module. You can use generate with loops and conditional statements.
Automatic Task Declaration	Dynamic allocation and release of storage for tasks.

Copyright © 2013 Synopsys, Inc.

Feature	Description
Multidimensional Arrays	Groups elements of the declared element type into multi-dimensional objects.
Variable Partial Select	Supports indexed part select expressions (+: and -:), which use a variable range to provide access to a word or part of a word.
Cross-Module Referencing	Accesses elements across modules.
ifndef and elsif Compiler Directives	'ifndef and 'ilsif compiler directive support.

Combined Data, Port Types (ANSI C-style Modules)

In Verilog 2001, you can combine module data and port type declarations to be concise, as shown below:

Verilog '95

```
module adder_16 (sum, cout, cin, a, b);
output [15:0] sum;
output cout;
input [15:0] a, b;
input cin;
reg [15:0] sum;
reg cout;
wire [15:0] a, b;
wire cin;
```

Verilog 2001

```
module adder_16(output reg [15:0] sum, output reg cout,
input wire cin, input wire [15:0] a, b);
```


Copyright © 2013 Synopsys, Inc.

Comma-separated Sensitivity List

In Verilog 2001, you can use commas as separators in sensitivity lists (as in other Verilog lists).

Verilog '95

```
always @(a or b or cin)
   sum = a - b - cin;
always @(posedge clock or negedge reset)
   if (!reset)
     q <= 0;
   else
     q <= d;
```

Verilog 2001

```
always @(a, b or cin)
   sum = a - b - cin;
always @(posedge clock, negedge reset)
   if (!reset)
     q <= 0;
   else
     q <= d;
```

Wildcards (*) in Sensitivity List

In Verilog 2001, you can use @* or @(*) to include all signals in a procedural block, eliminating mismatches between RTL and post-synthesis simulation.

Copyright © 2013 Synopsys, Inc.

Verilog '95

```
always @(a or b or cin)
   sum = a - b - cin;
```

Verilog 2001

```
// Style 1:
always @(*)
   sum = a - b - cin;
// Style 2:
always @*
   sum = a - b - cin;
```

Signed Signals

In Verilog 2001, data types net and reg, module ports, integers of different bases and signals can all be signed. You can assign and compare signed signals, and perform signed operations for vectors of any length.

Declaration

```
module adder (output reg signed [31:0] sum,
   wire signed input [31:0] a, b;
```

Assignment

```
wire signed [3:0] a = 4'sb1001;
```

Comparison

```
wire signed [1:0] sel;
parameter p0 = 2'sb00, p1 = 2'sb01, p2 = 2'sb10, p3 = 2'sb11;
case sel
   p0: ...
   p1: ...
   p2: ...
   p3: ...
endcase
```


Copyright © 2013 Synopsys, Inc.

Inline Parameter Assignment by Name

In Verilog 2001, you can assign values to parameters by name, inline:

```
module top( /* port list of top-level signals */ );
   dff #(.param1(10), .param2(5)) inst_dff(q, d, clk);
endmodule
```

where:

```
module dff #(parameter param1=1, param2=2) (q, d, clk);
   input d, clk;
   output q;
...
endmodule
```

Constant Function

In Verilog 2001, you can use constant functions to build complex values at elaboration time.

Example – Constant function

```
module ram
// Verilog 2001 ANSI parameter declaration syntax
   #(parameter depth= 129,
   parameter width=16 )
// Verilog 2001 ANSI port declaration syntax
(input clk, we,
   // Calculate addr width using Verilog 2001 constant function
   input [clogb2(depth)-1:0] addr,
   input [width-1:0] di,
   output reg [width-1:0] do );
function integer clogb2;
input [31:0] value;
      for (clogb2=0; value>0; clogb2=clogb2+1)
      value = value>>1;
   endfunction
reg [width-1:0] mem[depth-1:0];

always @(posedge clk) begin
   if (we)
      begin
         mem[addr]<= di;
         do<= di;
```

```
        end
        else
           do<= mem[addr];
        end
    endmodule
```

Configuration Blocks

Verilog configuration blocks define a set of rules that explicitly specify the exact source description to be used for each instance in a design. A configuration block is defined outside the module. Currently, support is limited to single configuration blocks.

Syntax

config *configName*;
  design *libraryIdentifier.moduleName*;
  default liblist *listofLibraries*;
  configurationRule;
endconfig

Design Statement

The design statement specifies the library and module for which the configuration rule is to defined.

design *libraryIdentifier.moduleName*;
  libraryIdentifier **:-** Library Name
  moduleName **:-** Module Name

Default Statement

The default liblist statement lists the library from which the definition of the module and sub-modules can be selected. A use clause cannot be used in this statement.

default liblist *listof_Libraries*;
  listofLibraries **:-** List of Libraries

Configuration Rule Statement

In this section, rules are defined for different instances or cells in the design. The rules are defined using instance or cell clauses.

- Instance Clause – specifies the particular source description for a given instance in the design.

Copyright © 2013 Synopsys, Inc.

- Cell Clause – specifies the source description to be picked for a particular cell/module in a given design.

A configuration rule can be defined as any of the following:

- instance clause with liblist

  `instance` *moduleName*`.`*instance* `liblist` *listofLibraries*`;`

- instance clause with use clause

  `instance` *moduleName*`.`*instance* `use` *libraryIdentifier*`.`*cellName*`;`

- cell clause with liblist

  `cell` *cellName* `liblist` *listofLibraries*`;`

- cell clause with use clause

  `cell` *cellName* `use` *libraryIdentifier*`.`*cellName*`;`

Configuration Block Examples

The following examples illustrate Verilog 2001 configuration blocks.

Example – Configuration with instance clause

The following example has different definitions for the leaf module compiled into the multlib and xorlib libraries; configuration rules are defined specifically for instance u2 in the top module to have the definition of leaf module as XOR (by default the leaf definition is multiplier). This example uses an instance clause with liblist to define the configuration rule.

```
//********Leaf module with the Multiplication definition

// Multiplication definition is compiled to the library "multlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib multlib "leaf_mult.v"

module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output reg [15:0] dout
);
```

```
always@*
   dout = d1 * d2;
endmodule //EndModule

//********Leaf module with the XOR definition

// XOR definition is compiled to the library "xorlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib xorlib "leaf_xor.v"

module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output reg[15:0] dout
);

always@(*)
   dout = d1 ^ d2;
endmodule //EndModule

//********Top module definition

// Top module definition is compiled to the library "TOPLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib TOPLIB "top.v"

module top
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
   input [7:0] d3,
   input [7:0] d4,
//Output Port
   output [15:0] dout1,
   output [15:0] dout2
);
```


Copyright © 2013 Synopsys, Inc.

```
leaf
u1
(
   .d1(d1),
   .d2(d2),
   .dout(dout1)
);

leaf
u2
(
   .d1(d3),
   .d2(d4),
   .dout(dout2)
);

endmodule //End Module

//********Configuration Definition

// Configuration definition is compiled to the library "TOPLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib TOPLIB "cfg.v"

config cfg;
   design TOPLIB.top;
   default liblist multlib xorlib TOPLIB; //By default the leaf
      // definition is Multiplication definition
   instance top.u2 liblist xorlib; //For instance u2 the default
      // definition is overridden and the "leaf" definition is
      // picked from "xorlib" which is XOR.
endconfig //EndConfiguration
```

Example – Configuration with cell clause

In the following example, different definitions of the leaf module are compiled into the multlib and xorlib libraries; a configuration rule is defined for cell leaf that picks the definition of the cell from the multlib library. This example uses a cell clause with a use clause to define the configuration rule.

```
//********Leaf module with the Multiplication definition

// Multiplication definition is compiled to the library "multlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib multlib "leaf_mult.v"
```

```
module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output reg [15:0] dout
);

always@*
   dout = d1 * d2;
endmodule //EndModule

//********Leaf module with the XOR definition

// XOR definition is compiled to the library "xorlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
//add_file -verilog -lib xorlib "leaf_xor.v"

module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output reg[15:0] dout
);

always@(*)
   dout = d1 ^ d2;
endmodule //EndModule

//********Top module definition

// Top module definition is compiled to the library "TOPLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib TOPLIB "top.v"

module top
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
   input [7:0] d3,
```


Copyright © 2013 Synopsys, Inc.

```
   input [7:0] d4,
//Output Port
   output [15:0] dout1,
   output [15:0] dout2
);

leaf
u1
(
   .d1(d1),
   .d2(d2),
   .dout(dout1)
);

leaf
u2
(
   .d1(d3),
   .d2(d4),
   .dout(dout2)
);
endmodule //End Module

//********Configuration Definition

// Configuration definition is compiled to the library "TOPLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib TOPLIB "cfg.v"

config cfg;
   design TOPLIB.top;
   default liblist xorlib multlib TOPLIB; //By default the leaf
      // definition uses the XOR definition
   cell leaf use multlib.leaf; //Definition of the instances u1 and
u2
      // will be Multiplier which is picked from "multlib"
endconfig //EndConfiguration
```

Example – Hierarchical reference of the module inside the configuration

In the following example, different definitions of leaf are compiled into the multlib, addlib, and xorlib libraries; the configuration rule is defined for instance u2 that is referenced in the hierarchy as the lowest instance module using an instance clause.

```
//********Leaf module with the Multiplication definition
```

```
// Multiplication definition is compiled to the library "multlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib multlib "leaf_mult.v"

module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output reg [15:0] dout
);

always@*
   dout = d1 * d2;
endmodule //EndModule

//********Leaf module with the XOR definition

// XOR definition is compiled to the library "xorlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib xorlib "leaf_xor.v"

module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output reg[15:0] dout
);

always@(*)
   dout = d1 ^ d2;
endmodule //EndModule

//********Leaf module with the ADDER definition

// ADDER definition is compiled to the library "addlib"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib xorlib "leaf_add.v"
```


Copyright © 2013 Synopsys, Inc.

```
module leaf
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
//Output Port
   output [15:0] dout
);

assign dout = d1 + d2;
endmodule

//********Sub module definition

// Sub module definition is compiled to the library "SUBLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib SUBLIB "sub.v"

module sub
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
   input [7:0] d3,
   input [7:0] d4,
//Output Port
   output [15:0] dout1,
   output [15:0] dout2
);

leaf
u1
(
   .d1(d1),
   .d2(d2),
   .dout(dout1)
);

leaf
u2
(
   .d1(d3),
   .d2(d4),
   .dout(dout2)
);
endmodule //End Module
```

```
//********Top module definition

// Top module definition is compiled to the library "TOPLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib TOPLIB "top.v"

module top
(
//Input Port
   input [7:0] d1,
   input [7:0] d2,
   input [7:0] d3,
   input [7:0] d4,
//Output Port
   output [15:0] dout1,
   output [15:0] dout2
);

sub
u1
(
   .d1(d1),
   .d2(d2),
   .d3(d3),
   .d4(d4),
   .dout1(dout1),
   .dout2(dout2)
);
endmodule //End Module

//********Configuration Definition

// Configuration definition is compiled to the library "TOPLIB"
// Command to be added in the synplify project file to compile a
// specific HDL to a specific library is
// add_file -verilog -lib TOPLIB "cfg.v"

config cfg;
   design TOPLIB.top;
   default liblist addlib multlib xorlib TOPLIB SUBLIB; //By
default,
      //the leaf definition uses the ADDER definition
   instance top.u1.u2 liblist xorlib multlib; //For instances u2 is
      //referred hierarchy to lowest instances and the default
definition
      //is overridden by XOR definition for this instanceendconfig
//EndConfiguration
```


Copyright © 2013 Synopsys, Inc.

Limitations

- Verilog configuration support is limited to single configuration.
- Nested configuration is not supported.
- Top-level design name in the project file must match the top-level design name in the design clause of the configuration construct.
- A use clause with the cell name or library name omitted is not supported.
- The case where the configuration name and the module name are the same is not supported.
- Mixed HDL configuration is not supported.
- Multiple top levels in the design clause are not supported.
- Compiling the same configuration file to multiple libraries is not supported.

Localparams

In Verilog 2001, localparams (constants that cannot be redefined or modified) follow the same parameter rules in regard to size and sign.

Example:

```
parameter ONE = 1
localparam TWO=2*ONE
localparam [3:0] THREE=TWO+1;
localparam signed [31:0] FOUR=2*TWO;
```

$signed and $unsigned Built-in Functions

In Verilog 2001, the built-in Verilog 2001 functions can be used to convert types between signed and unsigned.

```
c = $signed (s); /* Assign signed valued of s to c. */
d = $unsigned (s); /* Assign unsigned valued of s to d. */
```

Copyright © 2013 Synopsys, Inc.

$clog2 Constant Math Function

Verilog-2005 includes the $clog2 constant math function which returns the value of the log base-2 for the argument passed. This system function can be used to compute the minimum address width necessary to address a memory of a given size or the minimum vector width necessary to represent a given number of states.

Syntax

$clog2(*argument*)

In the above syntax, argument is an integer or vector.

Example 1 – Constant Math Function Counter

```
module top
#( parameter COUNT = 256 )
//Input
(  input clk,
   input rst,
//Output
//Function used to compute width based on COUNT value of counter:
   output [$clog2(COUNT)-1:0] dout );
reg[$clog2(COUNT)-1:0]count;

always@(posedge clk)
begin
   if(rst)
      count = 'b0;
   else
      count = count + 1'b1;
   end
assign dout = count;
endmodule
```

Example 2 – Constant Math Function RAM

```
module top
#
(  parameter DEPTH = 256,
   parameter WIDTH = 16 )
(
//Input
```


Copyright © 2013 Synopsys, Inc.

```
   input clk,
   input we,
   input rst,
//Function used to compute width of address based on depth of RAM:
   input [$clog2(DEPTH)-1:0] addr,
   input [WIDTH-1:0] din,
//Output
   output reg[WIDTH-1:0] dout );
reg[WIDTH-1:0] mem[(DEPTH-1):0];

always @ (posedge clk)
   if (rst == 1)
      dout = 0;
   else
      dout = mem[addr];

always @(posedge clk)
   if (we) mem[addr] = din;

endmodule
```

Generate Statement

The newer Verilog 2005 generate statement is now supported in Verilog 2001. Defparams, parameters, and function and task declarations within generate statements are supported. In addition, the naming scheme for registers and instances is enhanced to include closer correlation to specified generate symbolic hierarchies. Generated data types have unique identifier names and can be referenced hierarchically. Generate statements are created using one of the following three methods: generate-loop, generate-conditional, or generate-case.

```
// for loop
generate
begin:G1
   genvar i;
   for (i=0; i<=7; i=i+1)
   begin :inst
      adder8 add (sum [8*i+7 : 8*i], c0[i+1],
      a[8*i+7 : 8*i], b[8*i+7 : 8*i], c0[i]);
   end
end
endgenerate
```

```
// if-else
generate
   if (adder_width < 8)
      ripple_carry # (adder_width) u1 (a, b, sum);
   else
      carry_look_ahead # (adder_width) u1 (a, b, sum);
endgenerate

// case
parameter WIDTH=1;
generate
   case (WIDTH)
      1: adder1 x1 (c0, sum, a, b, ci);
      2: adder2 x1 (c0, sum, a, b, ci);
      default: adder # width (c0, sum, a, b, ci);
   endcase
endgenerate
```

Automatic Task Declaration

In Verilog 2001, tasks can be declared as automatic to dynamically allocate new storage each time the task is called and then automatically release the storage when the task exits. Because there is no retention of tasks from one call to another as in the case of static tasks, the potential conflict of two concurrent calls to the same task interfering with each other is avoided. Automatic tasks make it possible to use recursive tasks.

This is the syntax for declaring an automatic task:

task automatic *taskName* **(***argument* **[,** *argument* **, ...]) ;**

Arguments to automatic tasks can include any language-defined data type (reg, wire, integer, logic, bit, int, longint, or shortint) or a user-defined datatype (typedef, struct, or enum). Multidimensional array arguments are not supported.

Automatic tasks can be synthesized but, like loop constructs, the synthesis tool must be able to statically determine how many levels of recursive calls are to be made. Automatic (recursive) tasks are used to calculate the factorial of a given number.

Copyright © 2013 Synopsys, Inc.

Example

```
module automatic_task (input byte in1,
   output bit [8:0] dout);
parameter FACT_OP = 3;
bit [8:0] dout_tmp;

task automatic factorial(input byte operand,
   output bit [8:0] out1);
integer nFuncCall = 0;
begin
   if (operand == 0)
   begin
      out1 = 1;
   end
   else
   begin
      nFuncCall++;
      factorial((operand-1), out1);
      out1 = out1 * operand;
   end
end
endtask

always_comb
factorial(FACT_OP,dout_tmp);
assign dout = dout_tmp + in1 ;
endmodule
```

Multidimensional Arrays

In Verilog 2001, arrays are declared by specifying the element address ranges after the declared identifiers. Use a constant expression, when specifying the indices for the array. The constant expression value can be a positive integer, negative integer, or zero. Refer to the following examples.

2-dimensional wire object	my_wire is an eight-bit-wide vector with indices from 5 to 0. `wire [7:0] my_wire [5:0];`
3-dimensional wire object	my_wire is an eight-bit-wide vector with indices from 5 to 0 whose indices are from 3 down to 0. `wire [7:0] my_wire [5:0] [3:0];`
3-dimensional wire object	my_wire is an eight-bit-wide vector (-4 to 3) with indices from -3 to 1 whose indices are from 3 down to 0. `wire [-4:3] my_wire [-3:1] [3:0];`

These examples apply to register types too:

```
reg [3:0] mem[7:0]; // A regular memory of 8 words with 4
   //bits/word.

reg [3:0] mem[7:0][3:0]; // A memory of memories.
```

There is a Verilog restriction which prohibits bit access into memory words. Verilog 2001 removes all such restrictions. This applies equally to wires types. For example:

```
wire[3:0] my_wire[3:0];

assign y = my_wire[2][1]; // refers to bit 1 of 2nd word (word
   //does not imply storage here) of my_wire.
```

Variable Partial Select

In Verilog 2001, indexed partial select expressions (+: and -:), which use a variable range to provide access to a word or part of a word, are supported. The software extracts the size of the operators at compile time, but the index expression range can remain dynamic. You can use the partial select operators to index any non-scalar variable.

The syntax to use these operators is described below.

vectorName [*baseExpression* **+:** *widthExpression*]
vectorName [*baseExpression* **-:** *widthExpression*]

Copyright © 2013 Synopsys, Inc.

vectorName	Name of vector. Direction in the declaration affects the selection of bits
baseExpression	Indicates the starting point for the array. Can be any legal Verilog expression.
+:	The +: expression selects bits starting at the *baseExpression* while adding the *widthExpression*. Indicates an upward slicing.
-:	The -: expression selects bits starting at the *baseExpression* while subtracting the *widthExpression*. Indicates a downward slicing.
widthExpression	Indicates the width of the slice. It must evaluate to a constant at compile time. If it does not, you get a syntax error.

This is an example using partial select expressions:

```
module part_select_support (down_vect, up_vect, out1, out2, out3);
output [7:0] out1;
output [1:0] out2;
output [7:0] out3;
input [31:0] down_vect;
input [0:31] up_vect;
wire [31:0] down_vect;
wire [0:31] up_vect;
wire [7:0] out1;
wire [1:0] out2;
wire [7:0] out3;
wire [5:0] index1;
assign index1 = 1;
assign out1 = down_vect[index1+:8]; // resolves to [8:1]
assign out2 = down_vect[index1-:8]; // should resolve to [1:0],
   // but resolves to constant 2'b00 instead
assign out3 = up_vect[index1+:8]; // resolves to [1:8]
endmodule
```

For the Verilog code above, the following description explains how to validate partial select assignments to out2:

- The compiler first determines how to slice down_vect.
  - down_vect is an array of [31:0]

  - assign out2 = down_vect [1 -: 8] will slice down_vect starting at value 1 down to -6 as [1 : -6], which includes [1, 0, -1, -2, -3, -4, -5, -6]

- Then, the compiler assigns the respective bits to the outputs.
  - out2 [0] = down_vect [-6]
    out2 [1] = down_vect [-5]
  - Negative ranges cannot be specified, so out2 is tied to “00”.
  - Therefore, change the following expression in the code to:
    assign out2 = down_vect [1 -: 2], which resolves to down_vect [1,0]

Cross-Module Referencing

Cross-module referencing is a method of accessing an element across modules in Verilog and SystemVerilog. Verilog supports accessing elements across different scopes using the hierarchical reference (.) operator. Cross-module referencing can also be done on the variable of any of the data types available in SystemVerilog.

Cross-module referencing can be downward or upward, starting with the top module.

Downward Cross-Module Referencing

In downward cross-module referencing, you reference elements of lower-level modules in the higher-level modules through instantiated names. This is the syntax for a downward cross-module reference:

port/variable = *inst1*.*inst2*.*value*; // XMR Read

inst1.*inst2*.*port/variable* = *value*; // XMR Write

In this syntax, *inst1* is the name of an instance instantiated in the top module and *inst2* is the name of an instance instantiated in *inst1*. *Value* can be a constant, parameter, or variable. *Port/variable* is defined/declared once in the current module.

Copyright © 2013 Synopsys, Inc.

Example – Downward Read Cross-Module Reference

```
module top (
   input a,
   input b,
   output c,
   output d );

sub inst1 (.a(a), .b(b), .c(c) );
assign d = inst1.a;
endmodule

module sub (
   input a,
   input b,
   output c );

assign c = a  & b;
endmodule
```

Example – Downward Write Cross-Module Reference

```
module top
(  input a,
   input b,
   output c,
   output d
);

sub inst1 (.a(a), .b(b), .c(c), .d(d) );
assign top.inst1.d = a;
endmodule

module sub
(  input a,
   input b,
   output c,
   output d
);

assign c = a & b;
endmodule
```

Upward Cross-Module Referencing

In upward cross-module referencing, a lower-level module references items in a higher-level module in the hierarchy through the name of the top module.

This is the syntax for an upward reference from a lower module:

port/variable = *top.inst1.inst2.value*; // XMR Read

top.inst1.inst2.port/variable = *value*; // XMR Write

The starting reference is the top-level module. In this syntax, *top* is the name of the top-level module, *inst1* is the name of an instance instantiated in top module and *inst2* is the name of an instance instantiated in *inst1*. Value can be a constant, parameter, or variable. *Port/variable* is the one defined/declared in the current module.

Example – Upward Read Cross-Module Reference

```
module top (
   input a,
   input b,
   output c,
   output d );

sub inst1 (.a(a), .b(b), .c(c), .d(d) );
endmodule

module sub (
   input a,
   input b,
   output c,
   output d );

assign c = a & b;
assign d = top.a;
endmodule
```

Limitations

The following limitations currently exist with cross-module referencing:

- Cross-module referencing through an array of instances is not supported.
- Cross-module referencing into generate blocks is not supported.
- In upward cross-module referencing, the reference must be an absolute path (an absolute path is always from the top-level module).
- Functions and tasks cannot be accessed through cross-module reference notation.

Copyright © 2013 Synopsys, Inc.

- You can only use cross-module referencing with Verilog/SystemVerilog elements. You cannot access VHDL elements with hierarchical references.

ifndef and elsif Compiler Directives

Verilog 2001 supports the `ifndef and `elsif compiler directives. Note that the `ifndef directive is the opposite of `ifdef.

```
module top(output out);
   `ifndef a
      assign out = 1'b01;
   `elsif b
      assign out = 1'b10;
   `else
      assign out = 1'b00;
   `endif
endmodule
```

Verilog Synthesis Guidelines

This section provides guidelines for synthesis using Verilog and covers the following topics:

- General Synthesis Guidelines, on page 667
- Library Support in Verilog, on page 668
- Constant Function Syntax Restrictions, on page 672
- Multi-dimensional Array Syntax Restrictions, on page 672
- Signed Multipliers in Verilog, on page 674
- Verilog Language Guidelines: always Blocks, on page 675
- Initial Values in Verilog, on page 676
- Cross-language Parameter Passing in Mixed HDL, on page 683
- Library Directory Specification for the Verilog Compiler, on page 683

General Synthesis Guidelines

Some general guidelines are presented here to help you synthesize your Verilog design. See *Verilog Module Template*, on page 685 for additional information.

- Top-level module – The synthesis tool picks the last module compiled that is not referenced in another module as the top-level module. Module selection can be overridden from the Verilog panel of the Implementation Options dialog box.
- Simulate your design before synthesis to expose logic errors. Logic errors that you do not catch are passed through the synthesis tool, and the synthesized results will contain the same logic errors.
- Simulate your design after placement and routing – Have the place-and-route tool generate a post placement and routing (timing-accurate) simulation netlist, and do a final simulation before programming your devices.
- Avoid asynchronous state machines – To use the synthesis tool for asynchronous state machines, make a netlist of technology primitives from your target library.

Copyright © 2013 Synopsys, Inc.

- Level-sensitive latches – For modeling level-sensitive latches, use continuous assignment statements.

Library Support in Verilog

Verilog libraries are used to compile design units; this is similar to VHDL libraries. Use the libraries in Verilog to support mixed-HDL designs, where the VHDL design includes instances of a Verilog module that is compiled into a specific library. Library support in Verilog can be used with Verilog 2001 and SystemVerilog designs.

Compiling Design Units into Libraries

By default, the Verilog source files are compiled into the work library. You can compile these Verilog source files into any user-defined library.

To compile a Verilog file into a user-defined library:

1. Select the file in the Project view.

   The library name appears next to the filename; it directly follows the filename.

2. Right-click and select File Options from the popup menu. Specify the name for your library in the Library Names field. You can:
   - Compile multiple files into the same library.
   - Also compile the same file into multiple libraries.

Searching for Verilog Design Units in Mixed-HDL Designs

When a VHDL file references a Verilog design unit, the compiler first searches the corresponding library for which the VHDL file was compiled. If the Verilog design unit is not found in the user-defined library for which the VHDL file was compiled, the compiler searches the work library and then all the other Verilog libraries.

Therefore, to use a specific Verilog design unit in the VHDL file, compile the Verilog file into the same user-defined library for which the corresponding VHDL file was compiled. You cannot use the VHDL library clause for Verilog libraries.

Specifying the Verilog Top-level Module

To set the Verilog top-level module for a user-defined library, use *libraryName.moduleName* in the Top Level Module field on the Verilog tab of the Implementation Options dialog box. You can also specify the following equivalent Tcl command:

```
set_option -top_module "signed.top"
```


Copyright © 2013 Synopsys, Inc.

Limitations

The following functions are not supported:

- Direct Entity Instantiation
- Configuration for Verilog Instances

Example 1: Specifying Verilog Top-level Module—Compiled to the Non-work Library

```
//top_unsigned.v compiled into a user defined library - "unsigned"
//add_file -verilog -lib unsigned "./top_unsigned.v"
module top ( input unsigned [7:0] a, b,
output unsigned [15:0] result );
assign result = a * b;
endmodule

//top_signed.v compiled into a user defined library - "signed"
//add_file -verilog -lib signed "./top_signed.v"
module top ( input signed [7:0] a, b,
output signed [15:0] result );
assign result = a * b;
endmodule
```

To set the top-level module from the signed library:

- Specify the prefix library name for the module in the Top Level Module option in the Verilog panel of the Implementation Options dialog box.
- `set_option -top_module "signed.top"`

Example 2: Referencing Verilog Module from VHDL

This example includes two versions of the Verilog `sub` module that are compiled into the signed_lib and unsigned_lib libraries. The compiler uses the sub module from unsigned_lib when the top.vhd is compiled into unsigned_lib.

```
//Sub module sub in sub_unsigned is compiled into unsigned_lib
//add_file -verilog -lib unsigned_lib "./sub_unsigned.v"
module sub ( input unsigned [7:0] a, b,
output unsigned [15:0] result );
assign result = a * b;
endmodule

//Sub module sub in sub_signed is compiled into signed_lib
//add_file -verilog -lib signed_lib "./sub_signed.v"
module sub ( input signed [7:0] a, b,
output signed [15:0] result );
assign result = a * b;
endmodule

//VHDL Top module top is compiled into unsigned_lib library
// add_file -vhdl -lib unsigned_lib "./top.vhd"
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY top IS
GENERIC(
size_t : integer := 8
);
   PORT( a_top : IN std_logic_vector(size_t-1 DOWNTO 0);
   b_top : IN std_logic_vector(size_t-1 DOWNTO 0);
   result_top : OUT std_logic_vector(2*size_t-1 DOWNTO 0)
);
END top;

ARCHITECTURE RTL OF top IS
component sub
   PORT(a : IN std_logic_vector(7 DOWNTO 0);
   b : IN std_logic_vector(7 DOWNTO 0);
   result : OUT std_logic_vector(15 DOWNTO 0));
END component;
BEGIN
U1 : sub
   PORT MAP (
```


Copyright © 2013 Synopsys, Inc.

```
      a => a_top,
      b => b_top,
   result => result_top
);
END ARCHITECTURE RTL;
```

Constant Function Syntax Restrictions

For Verilog 2001, the syntax for constant functions is identical to the existing function definitions in Verilog. Restrictions on constant functions are as follows:

- No hierarchal references are allowed
- Any function calls inside constant functions must be constant functions
- System tasks inside constant functions are ignored
- System functions inside constant functions are illegal
- Any parameter references inside a constant function should be visible
- All identifiers, except arguments and parameters, should be local to the constant function
- Constant functions are illegal inside the scope of a generate statement

Multi-dimensional Array Syntax Restrictions

For Verilog 2001, the following examples show multi-dimensional array syntax restrictions.

```
reg [3:0] arrayb [7:0][0:255];

arrayb[1] = 0;
// Illegal Syntax - Attempt to write to elements [1][0]..[1][255]

arrayb[1][12:31] = 0;
// Illegal Syntax - Attempt to write to elements [1][12]..[1][31]

arrayb[1][0] = 0;
// Okay. Assigns 32'b0 to the word referenced by indices [1][0]

Arrayb[22][8] = 0;
// Semantic Error, There is no word 8 in 2nd dimension.
```

When using multi-dimension arrays, the association is always from right-to-left while the declarations are left-to-right.

Example 1

```
module test (input a,b, output z, input clk, in1, in2);
reg tmp [0:1][1:0];

always @(posedge clk)
begin
   tmp[1][0] <= a ^ b;
   tmp[1][1] <= a & b;
   tmp[0][0] <= a | b;
   tmp[0][1] <= a &~ b;
end
assign z = tmp[in1][in2];

endmodule
```

Example 2

```
module bb(input [2:0] in, output [2:0] out)
   /* synthesis syn_black_box */;
endmodule

module top(input [2:0] in, input [2:1] d1, output [2:0] out);
wire [2:0] w1[2:1];
wire [2:0] w2[2:1];

generate
begin : ABCD
   genvar i;
   for(i=1; i < 3; i = i+1)
   begin : CDEF
      assign w1[i] = in;
      bb my_bb(w1[i], w2[i]);
   end
end
endgenerate
assign out = w2[d1];

endmodule
```


Copyright © 2013 Synopsys, Inc.

Signed Multipliers in Verilog

This section applies only to those using Verilog compilers earlier than version 2001.

The software contains an updated signed multiplier module generator. A signed multiplier is used in Verilog whenever you multiply signed numbers. Because earlier versions of Verilog compilers do not support signed data types, an example is provided on how to write a signed multiplier in your Verilog design:

```
module smul4(a, b, clk, result);
input [3:0]a;
input [3:0]b;
input clk;
output [7:0]result;
wire [3:0] inputa_signbits, inputb_signbits;
reg [3:0]inputa;
reg [3:0]inputb;
reg [7:0]out, result;
assign inputa_signbits = {4{inputa[3]}};
assign inputb_signbits = {4{inputb[3]}};

always @(inputa or inputb or inputa_signbits or inputb_signbits)
begin
   out = {inputa_signbits,inputa} * {inputb_signbits,inputb};
end

always @(posedge clk)
begin
   inputa = a;
   inputb = b;
   result = out;
end

endmodule
```

Verilog Language Guidelines: always Blocks

An always block can have more than one event control argument, provided they are all edge-triggered events or all signals; these two kinds of arguments cannot be mixed in the same always block.

Examples

```
// OK: Both arguments are edge-triggered events
always @(posedge clk or posedge rst)

// OK: Both arguments are signals
always @(A or B)

// No good: One edge-triggered event, one signal
always @(posedge clk or rst)
```

An always block represents either sequential logic or combinational logic. The one exception is that you can have an always block that specifies level-sensitive latches and combinational logic. Avoid this style, however, because it is error prone and can lead to unwanted level-sensitive latches.

An event expression with posedge/negedge keywords implies edge-triggered sequential logic; and without posedge/negedge keywords implies combinational logic, a level-sensitive latch, or both.

Each sequential always block is triggered from exactly one clock (and optional sets and resets).

You must declare every signal assigned a value inside an always block as a reg or integer. An integer is a 32-bit quantity by default, and is used with the Verilog operators to do two's complement arithmetic.

Syntax:

integer [*msb*:*lsb*] *identifier* **;**

Avoid combinational loops in always blocks. Make sure all signals assigned in a combinational always block are explicitly assigned values every time the always block executes, otherwise the synthesis tool needs to insert level-sensitive latches in the design to hold the last value for the paths that do not assign values. This is a common source of errors, so the tool issues a warning message that latches are being inserted into your design.

Copyright © 2013 Synopsys, Inc.

You will get an error message if you have combinational loops in your design that are not recognized as level-sensitive latches by the synthesis tool (for example if you have an asynchronous state machine).

It is illegal to have a given bit of the same reg or integer variable assigned in more than one always block.

Assigning a 'bx to a signal is interpreted as a “don't care” (there is no 'bx value in hardware); the synthesis tool then creates the hardware with the most efficient design.

Initial Values in Verilog

In Verilog, you can now store and pass initial values that the synthesis software previously ignored. Initial values specified in Verilog only affect the compiler output. This ensures that the synthesis results match the simulation results. You declare the initial values differently for registers and RAMs.

Initial Values for Registers

The synthesis compiler reads the procedural assign statements with initial values. It then stores the values, propagates them to inferred logic, and passes them down stream. The initial values only affect the output of the compiler; initial value properties are not forward-annotated to the final netlist.

If synthesis removes an unassigned register that has an initial value, the initialization values are still propagated forward. If bits of a register are unassigned, the compiler removes the unassigned bits and propagates the initial value.

Copyright © 2013 Synopsys, Inc.

In the following example (without an initial value specified), register one does not get any input. If it is not initialized, it is removed during the optimization process:

However, if the register is initialized to a value of 1, the compiler ensures that the initial value is used in synthesis.

```
module top(a, b, c, clk);
input clk;
input [3:0]a, b;
output [3:0]c;
reg [3:0]c =3'b1100;

always@(posedge clk)
   begin
      c<= a & b;
   end
endmodule
```


Copyright © 2013 Synopsys, Inc.

Technology View

Initial Values for RAMs

You can specify initial values for a RAM in a data file and then include the appropriate task enable statement, $readmemb or $readmemh, in the initial statement of the RTL code for the module. The inferred logic can be different due to the initial statement. The syntax for these two statements is as follows:

$readmemh ("*fileName***", *memoryName* [, *startAddress* [, *stopAddress*]]);**

$readmemb ("*fileName***", *memoryName* [, *startAddress* [, *stopAddress*]]);**

$readmemb	Use this with a binary data file.
$readmemh	Use this with a hexadecimal data file.
fileName	Name of the data file that contains initial values. See *Initialization Data File*, on page 680 for format examples.
memoryName	The name of the memory.
startAddress	Optional starting address for RAM initialization; if omitted, defaults to first available memory location.
stopAddress	Optional stopping address for RAM initialization; *startAddress* must be specified

RAM Initialization Example

This example shows a single-port RAM that is initialized using the $readmemb binary task enable statement which reads the values specified in the binary mem.ini file. See *Initialization Data File*, on page 680 for details of the binary and hexadecimal file formats.

```
module ram_inference (data, clk, addr, we, data_out);
input [27:0] data;
input clk, we;
input [10:0] addr;
output [27:0] data_out;
reg [27:0] mem [0:2000] /* synthesis syn_ramstyle = "no_rw_check" */;
reg [10:0] addr_reg;

initial
begin
   $readmemb ("mem.ini", mem, 2, 1900) /* Initialize RAM with contents */
      /* from locations 2 thru 1900*/;
end

always @(posedge clk)
begin
   addr_reg <= addr;
end
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge clk)
begin
   if(we)
   begin
      mem[addr] <= data;
   end
end

assign data_out = mem[addr_reg];
endmodule
```

Initialization Data File

The initialization data file, read by the $readmemb and $readmemh system tasks, contains the initial values to be loaded into the memory array. This initialization file can reside in the project directory or can be referenced by an include path relative to the project directory. The system $readmemb or $readmemh task first looks in the project directory for the named file and, if not found, searches for the file in the list of directories on the Verilog tab in include-path order.

If the initialization data file does not contain initial values for every memory address, the unaddressed memory locations are initialized to 0. Also, if a width mismatch exists between an initialization value and the memory width, loading of the memory array is terminated; any values initialized before the mismatch is encountered are retained.

Unless an internal address is specified (see *Internal Address Format*, on page 682), each value encountered is assigned to a successive word element of the memory. If no addressing information is specified either with the $readmem task statement or within the initialization file itself, the default starting address is the lowest available address in the memory. Consecutive words are loaded until either the highest address in the memory is reached or the data file is completely read.

If a start address is specified without a finish address, loading starts at the specified start address and continues upward toward the highest address in the memory. In either case, loading continues upward. If both a start address and a finish address are specified, loading begins at the start address and continues until the finish address is reached (or until all initialization data is read).

Copyright © 2013 Synopsys, Inc.

For example:

```
initial
begin
//$readmemh ("mem.ini", ram_bank1)
   /* Initialize RAM with contents from locations 0 thru 31*/;

//$readmemh ("mem.ini", ram_bank1,0)
   /* Initialize RAM with contents from locations 0 thru 31*/;

$readmemh ("mem.ini", ram_bank1, 0, 31)
   /* Initialize RAM with contents from locations 0 thru 31*/;

$readmemh ("mem.ini", ram_bank2, 31, 0)
   /* Initialize RAM with contents from locations 31 thru 0*/;
```

The data initialization file can contain the following:

- White space (spaces, new lines, tabs, and form-feeds)
- Comments (both comment formats are allowed)
- Binary values for the $readmemb task, or hexadecimal values for the $readmemh tasks

In addition, the data initialization file can include any number of hexadecimal addresses (see *Internal Address Format*, on page 682).

Binary File Format

The binary data file mem.ini that corresponds to the example in *RAM Initialization Example*, on page 679 looks like this:

```
11111111111111111111100110111   /* data for address 0 */
11111111111111111111101100111   /* data for address 1 */
11111111111111111111111000010
11111111111111111111100100001
11111111111111111111101110000
11111111111111111111011100110
... /* continues until Address 1999 */
```


Copyright © 2013 Synopsys, Inc.

Hex File Format

If you use $readmemh instead of $readmemb, the hexadecimal data file for the example in *RAM Initialization Example*, on page 679 looks like this:

```
FFFFF37   /* data for address 0 */
FFFFF63   /* data for address 1 */
FFFFFC2
FFFFF21
.../* continues until Address 1999 */
```

Internal Address Format

In addition to the binary and hex formats described above, the initialization file can include embedded hexadecimal addresses. These hexadecimal addresses must be prefaced with an at sign (@) as shown in the example below.

```
FFFFF37  /* data for address 0 */
FFFFF63  /* data for address 1 */
@0EA     /* memory address 234
FFFFFC2  /* data for address 234*/
FFFFF21  /* data for address 235*/
...
@0A7     /* memory address 137
FFFFF77  /* data for address 137*/
FFFFF7A  /* data for address 138*/
...
```

Either uppercase or lowercase characters can be used in the address. No white space is allowed between the @ and the hex address. Any number of address specifications can be included in the file, and in any order. When the $readmemb or $readmemh system task encounters an embedded address specification, it begins loading subsequent data at that memory location.

When addressing information is specified both in the system task and in the data file, the addresses in the data file must be within the address range specified by the system task arguments; otherwise, an error message is issued, and the load operation is terminated.

Copyright © 2013 Synopsys, Inc.

Forward Annotation of Initial Values

Initial values for RAMs and sequential shift components are forward annotated to the netlist. The compiler currently generates netlist (`srs`) files with seqshift, ram1, ram2, and nram components. If initial values are specified in the HDL code, the synthesis tool attaches an attribute to the component in the `srs` file.

For Altera RAMS, $readmemb or $readmemh passes the initialization data to the compiler and mapper with statements like the following:

`srs` File	ai .syn_initval "3 00001001 2 00000000 1 00010101 0 00111101 ";
`srm` File	ai .syn_initval "3 00001001 2 00000000 1 00010101 0 00111101 ";

It also creates a hex file with address and data as shown in the following example. The sixth and seventh digits are the address, and the third and fourth digits are the corresponding data values.

```
:0100030009F3
:0100020000FD
:0100010015E9
:010000003DC2
:00000001FF
```

This hex file is associated with the altsyncram in the `vqm` file as follows:

```
defparam mem_Z.init_file = "mem.hex";
```

Cross-language Parameter Passing in Mixed HDL

The compiler supports the passing of parameters for integers, natural numbers, real numbers, and strings from Verilog to VHDL. The compiler also supports the passing of these same generics from VHDL to Verilog.

Library Directory Specification for the Verilog Compiler

Currently, if a module is instantiated in a module top without a module definition, the Verilog compiler errors out. Verilog simulators provide a command line switch (-y *libraryDirectory*) to specify a set of library directories which the compiler searches.

Copyright © 2013 Synopsys, Inc.

Library directories are specified in the Library Directories section in the Verilog panel of the Implementations Options dialog box.

Example:

If the project has one Verilog file specified

```
module foo(input a, b, output z);

foobar u1 (a, b, z);

endmodule
```

and the project directories D:/libdir and D:/lib2dir are specified as the library directories, the following is passed

c_ver *some options* **-y "D:/libdir" -y "D:/lib2dir"** *more options* **foo.v**

to the Verilog compiler. Then, if foobar.v exists in one of the specified directories, it is loaded into the compiler.

Verilog Module Template

Hardware designs can include combinational logic, sequential logic, state machines, and memory. These elements are described in the Verilog module. You also can create hardware by directly instantiating built-in gates into your design (in addition to instantiating your own modules).

Within a Verilog module you can describe hardware with one or more continuous assignments, always blocks, module instantiations, and gate instantiations. The order of these statements within the module is irrelevant, and all execute concurrently. The following is the Verilog module template:

```
module <top_module_name>(<port_list>);

/* Port declarations. followed by wire,
   reg, integer, task and function declarations */

/* Describe hardware with one or more continuous assignments,
   always blocks, module instantiations and gate instantiations */

// Continuous assignment
wire <result_signal_name>;
assign <result_signal_name> = <expression>;

// always block
always @(<event_expression>)

begin
   // Procedural assignments
   // if statements
   // case, casex, and casez statements
   // while, repeat and for loops
   // user task and user function calls
end

// Module instantiation
<module_name> <instance_name> (<port_list>);

// Instantiation of built-in gate primitive
gate_type_keyword (<port_list>);

endmodule
```

The statements between the begin and end statements in an always block execute sequentially from top to bottom. If you have a fork-join statement in an always block, the statements within the fork-join execute concurrently.

Copyright © 2013 Synopsys, Inc.

You can add comments in Verilog by preceding your comment text with // (two forward slashes). Any text from the slashes to the end of the line is treated as a comment, and is ignored by the synthesis tool. To create a block comment, start the comment with /* (forward slash followed by asterisk) and end the comment with */ (asterisk followed by forward slash). A block comment can span any number of lines but cannot be nested inside another block comment.

Scalable Modules

This section describes creating and using scalable Verilog modules. The topics include:

- Creating a Scalable Module, on page 686
- Using Scalable Modules, on page 687

Creating a Scalable Module

You can create a Verilog module that is scalable, so that it can be stretched or shrunk to handle a user-specified number of bits in the port list buses.

Declare parameters with default parameter values. The parameters can be used to represent bus sizes inside a module.

Syntax

parameter *parameterName* **=** *value* **;**

You can define more than one parameter per declaration by using comma-separated *parameterName* **=** *value* pairs.

Example

```
parameter size = 1;
parameter word_size = 16, byte_size = 8;
```

Using Scalable Modules

To use scalable modules, instantiate the scalable module and then override the default parameter value with the defparam keyword. Give the instance name of the module you are overriding, the parameter name, and the new value.

Syntax

defparam *instanceName*.*parameterName* **=** *newValue* **;**

Example

```
big_register my_register (q, data, clk, rst);
defparam my_register.size = 64;
```

Combine the instantiation and the override in one statement. Use a # (hash mark) immediately after the module name in the instantiation, and give the new parameter value. To override more than one parameter value, use a comma-separated list of new values.

Syntax

moduleName **#** **(***newValuesList***)** *instanceName* **(***portList***)** **;**

Example

```
big_register #(64) my_register (q, data, clk, rst);
```

Creating a Scalable Adder

```
module adder(cout, sum, a, b, cin);

/* Declare a parameter, and give a default value */
parameter size = 1;
output cout;

/* Notice that sum, a, and b use the value of the size parameter */
output [size-1:0] sum;
input [size-1:0] a, b;
input cin;
assign {cout, sum} = a - b - cin;
endmodule
```


Copyright © 2013 Synopsys, Inc.

Scaling by Overriding a Parameter Value with defparam

```
module adder8(cout, sum, a, b, cin);
output cout;
output [7:0] sum;
input [7:0] a, b;
input cin;
adder my_adder (cout, sum, a, b, cin);

// Creates my_adder as an eight bit adder
defparam my_adder.size = 8;
endmodule
```

Scaling by Overriding the Parameter Value with

```
module adder16(cout, sum, a, b, cin);
output cout;
```

You can define a parameter at this level of hierarchy and pass that value down to a lower-level instance. In this example, a parameter called my_size is declared. You can declare a parameter with the same name as the lower level name (size) because this level of hierarchy has a different name range than the lower level and there is no conflict – but there is no correspondence between the two names either, so you must explicitly pass the parameter value down through the hierarchy.

```
parameter my_size = 16;    // I want a 16-bit adder
output [my_size-1:0] sum;
input [my_size-1:0] a, b;
input cin;

/* my_size overrides size inside instance my_adder of adder */
// Creates my_adder as a 16-bit adder
adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
```

Copyright © 2013 Synopsys, Inc.

Built-in Gate Primitives

You can create hardware by directly instantiating built-in gates into your design (in addition to instantiating your own modules.) The built-in Verilog gates are called primitives.

Syntax

gateTypeKeyword [*instanceName*] **(***portList***) ;**

The gate type keywords for simple and tristate gates are listed in the following tables. The *instanceName* is a unique instance name, and is optional. The signal names in the *portList* can be given in any order with the restriction that all outputs must come before the inputs. For tristate gates, outputs come first, then inputs, and then enable. The following tables list the available keywords.

Keyword (Simple Gates)	Definition
buf	buffer
not	inverter
and	and gate
nand	nand gate
or	or gate
nor	nor gate
xor	exclusive or gate
xnor	exclusive nor gate

Keyword (Tristate Gates)	Definition
bufif1	tristate buffer with logic one enable
bufif0	tristate buffer with logic zero enable
notif1	tristate inverter with logic one enable
notif0	tristate inverter with logic zero enable

Copyright © 2013 Synopsys, Inc.

Combinational Logic

Combinational logic is hardware with output values based on some function of the current input values. There is no clock, and no saved states. Most hardware is a mixture of combinational and sequential logic.

You create combinational logic with an always block and/or continuous assignments.

Combinational Logic Examples

The following combinational logic synthesis examples are included in the *installDirectory*/examples/verilog/common_rtl/combinat directory:

- Adders
- ALU
- Bus Sorter
- 3-to-8 Decoder
- 8-to-3 Priority Encoders
- Comparator
- Multiplexers (concurrent signal assignments, case statements, or if-then-else statements can be used to create multiplexers; the tool automatically creates parallel multiplexers when the conditions in the branches are mutually exclusive)
- Parity Generator
- Tristate Drivers

Copyright © 2013 Synopsys, Inc.

always Blocks for Combinational Logic

Use the Verilog always blocks to model combinational logic as shown in the following template.

```
always @(event_expression)
begin
   // Procedural assignment statements,
   // if, case, casex, and casez statements
   // while, repeat, and for loops
   // task and function calls
end
```

When modeling combinational logic with always blocks, keep the following in mind:

- The always block must have exactly one event control (@(*event_ expression*)) in it, located immediately after the always keyword.
- List all signals feeding into the combinational logic in the event expression. This includes all signals that affect signals that are assigned inside the always block. List all signals on the right side of an assignment inside an always block. The tool assumes that the sensitivity list is complete, and generates the desired hardware. However, it will issue a warning message if any signals on the right side of an assignment inside an always block are not listed, because your pre- and post-synthesis simulation results might not match.
- You must explicitly declare as reg or integer all signals you assign in the always block.

Note: Make sure all signals assigned in a combinational always block are explicitly assigned values each time the always block executes. Otherwise, the synthesis tool must insert level-sensitive latches in your design to hold the last value for the paths that do not assign values. This will occur, for instance, if there are combinational loops in your design. This often represents a coding error. The synthesis tool issues a warning message that latches are being inserted into your design because of combinational loops. You will get an error message if you have combinational loops in your design that are not recognized as level-sensitive latches by the synthesis tool.

Copyright © 2013 Synopsys, Inc.

Event Expression

Every always block must have one event control (@(*event_expression*)), that specifies the signal transitions that trigger the always block to execute. This is analogous to specifying the inputs to logic on a schematic by drawing wires to gate inputs. If there is more than one signal, separate the names with the or keyword.

Syntax

always @ (*signal1* **or** *signal2* ...**)**

Example

```
/* The first line of an always block for a multiplexer that
   triggers when 'a', 'b' or 'sel' changes */
always @(a or b or sel)
```

Locate the event control immediately after the always keyword. Do not use the posedge or negedge keywords in the event expression; they imply edge-sensitive sequential logic.

Example: Multiplexer

See also *Comma-separated Sensitivity List*, on page 644.

```
module mux (out, a, b, sel);
output out;
input a, b, sel;
reg out;

always @(a or b or sel)
begin
   if (sel)
      out = a;
   else
      out = b;
end
endmodule
```

Continuous Assignments for Combinational Logic

Use continuous assignments to model combinational logic. To create a continuous assignment:

1. Declare the assigned signal as a wire using the syntax:

   wire [*msb* **:** *lsb*] *result_signal* **;**

2. Specify your assignment with the assign keyword, and give the expression (value) to assign.

   assign *result_signal* = *expression* **;**

   or ...

   Combine the wire declaration and assignment into one statement:

   wire [*msb* **:** *lsb*] *result_signal* = *expression* **;**

Each time a signal on the right side of the equal sign (=) changes value, the expression re-evaluates, and the result is assigned to the signal on the left side of the equal sign. You can use any of the built-in operators to create the expression.

The bus range [msb **:** lsb] is only necessary if your signal is a bus (more than one bit wide).

All outputs and inouts to modules default to wires; therefore the wire declaration is redundant for outputs and inouts and assign *result_signal* = *expression* is sufficient.

Example: Bit-wise AND

```
module bitand (out, a, b);
output [3:0] out;
input [3:0] a, b;
/* This wire declaration is not required because "out" is an
   output in the port list */
wire [3:0] out;
assign out = a & b;
endmodule
```


Copyright © 2013 Synopsys, Inc.

Example: 8-bit Adder

```
module adder_8 (cout, sum, a, b, cin);
output cout;
output [7:0] sum;
input cin;
input [7:0] a, b;
assign {cout, sum} = a - b - cin;
endmodule
```

Signed Multipliers

A signed multiplier is inferred whenever you multiply signed numbers in Verilog 2001 or VHDL. However, Verilog 95 does not support signed data types. If your Verilog code does not use the Verilog 2001 standard, you can implement a signed multiplier in the following way:

```
module smul4(a, b, clk, result);
input [3:0]a;
input [3:0]b;
input clk;
output [7:0]result;
reg [3:0]inputa;
reg [3:0]inputb;
reg [7:0]out, result;

always @(inputa or inputb)
begin
   out = {{4{inputa[3]}},inputa} * {{4{inputb[3]}},inputb};
end

always @(posedge clk)
begin
   inputa = a;
   inputb = b;
   result = out;
end

endmodule
```

Copyright © 2013 Synopsys, Inc.

Sequential Logic

Sequential logic is hardware that has an internal state or memory. The state elements are either flip-flops that update on the active edge of a clock signal or level-sensitive latches that update during the active level of a clock signal.

Because of the internal state, the output values might depend not only on the current input values, but also on input values at previous times. A state machine is sequential logic where the updated state values depend on the previous state values. There are standard ways of modeling state machines in Verilog. Most hardware is a mixture of combinational and sequential logic.

You create sequential logic with always blocks and/or continuous assignments.

Sequential Logic Examples

The following sequential logic synthesis examples are included in the *installDirectory*/examples/verilog/common_rtl/sequentl directory:

- Flip-flops and level-sensitive latches
- Counters (up, down, and up/down)
- Register file
- Shift registers
- State machines

For additional information on synthesizing flip-flops and latches, see these topics:

- Flip-flops Using always Blocks, on page 696
- Level-sensitive Latches, on page 697
- Sets and Resets, on page 699
- SRL Inference, on page 704

Copyright © 2013 Synopsys, Inc.

Flip-flops Using always Blocks

To create flip-flops/registers, assign values to the signals in an always block, and specify the active clock edge in the event expression.

always Block Template

```
always @(event_expression)
begin
   // Procedural statements
end
```

The always block must have one event control (@(*event_expression*)) immediately after the always keyword that specifies the clock signal transitions that trigger the always block to execute.

Syntax

always @ (*edgeKeyword clockName*)

where *edgeKeyword* is posedge (for positive-edge triggered) or negedge (for negative-edge triggered).

Example

```
always @(posedge clk)
```

Assignments to Signals in always Blocks

When assigning signals in an always block:

- Explicitly declare, as a reg or integer, any signal you assign inside an always block.
- Any signal assigned within an edge-triggered always block will be implemented as a register; for instance, signal q in the following example.

Example

```
module dff_or (q, a, b, clk);
output q;
input a, b, clk;
reg q; // Declared as reg, since assigned in always block
```

```
always @(posedge clk)
begin
   q <= a | b;
end
endmodule
```

In this example, the result of a | b connects to the data input of a flip-flop, and the q signal connects to the q output of the flip-flop.

Level-sensitive Latches

The preferred method of modeling level-sensitive latches in Verilog is to use continuous assignment statements.

Example

```
module latchor1 (q, a, b, clk);
output q;
input a, b, clk;

assign q = clk ? (a | b) : q;
endmodule
```

Whenever clk, a, or b change, the expression on the right side re-evaluates. If your clk becomes true (active, logic 1), a | b is assigned to the q output. When the clk changes and becomes false (deactivated), q is assigned to q (holds the last value of q). If a or b changes and clk is already active, the new value a | b is assigned to q.

Although it is simpler to specify level-sensitive latches using continuous assignment statements, you can create level-sensitive latches from always blocks. Use an always block and follow these guidelines for event expression and assignments.

always Block Template

```
always@(event_expression)
begin   // Procedural statements
end
```

Whenever the assignment to a signal is incompletely defined, the event expression specifies the clock signal and the signals that feed into the data input of the level-sensitive latch.

Copyright © 2013 Synopsys, Inc.

Syntax

always @ (*clockName* **or** *signal1* **or** *signal2* ... **)**

Example

```
always @(clk or data)
begin
   if (clk)
      q <= data;
end
```

The always block must have exactly one event control (@(*event_expression*)) in it, and must be located immediately after the always keyword.

Assignments to Signals in always Blocks

You must explicitly declare as reg or integer any signal you assign inside an always block.

Any incompletely-defined signal that is assigned within a level-triggered always block will be implemented as a latch.

Whenever level-sensitive latches are generated from an always block, the tool issues a warning message, so that you can verify if a given level-sensitive latch is really what you intended. (If you model a level-sensitive latch using continuous assignment then no warning message is issued.)

Example: Creating Level-sensitive Latches You Want

```
module latchor2 (q, a, b, clk);
output q;
input a, b, clk;
reg q;

always @(clk or a or b)
begin
   if (clk)
      q <= a | b;
end
endmodule
```

If clk, a, or b change, and clk is a logic 1, then set q equal to a|b.

What to do when clk is a logic zero is not specified (there is no else in the if statement), so when clk is a logic 0, the last value assigned is maintained (there is an implicit q=q). The synthesis tool correctly recognizes this as a level-sensitive latch, and creates a level-sensitive latch in your design. The tool issues a warning message when you compile this module (after examination, you may choose to ignore this message).

Example: Creating Unwanted Level-sensitive Latches

```
module mux4to1 (out, a, b, c, d, sel);
output out;
input a, b, c, d;
input [1:0] sel;
reg out;

always @(sel or a or b or c or d)
begin
   case (sel)
      2'd0: out = a;
      2'd1: out = b;
      2'd3: out = d;
   endcase
end
endmodule
```

In the above example, the sel case value 2'd2 was intentionally omitted. Accordingly, out is not updated when the select line has the value 2'd2, and a level-sensitive latch must be added to hold the last value of out under this condition. The tool issues a warning message when you compile this module, and there can be mismatches between RTL simulation and post-synthesis simulation. You can avoid generating level-sensitive latches by adding the missing case in the case statement; using a “default” case in the case statement; or using the Verilog full_case directive.

Sets and Resets

A set signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic one. Asynchronous sets take place independent of the clock, whereas synchronous sets only occur on an active clock edge.

Copyright © 2013 Synopsys, Inc.

A reset signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic zero. Asynchronous resets take place independent of the clock, whereas synchronous resets take place only at an active clock edge.

Asynchronous Sets and Resets

Asynchronous sets and resets are independent of the clock. When active, they set flip-flop outputs to one or zero (respectively), without requiring an active clock edge. Therefore, list them in the event control of the always block, so that they trigger the always block to execute, and so that you can take the appropriate action when they become active.

Event Control Syntax

always @ (*edgeKeyword clockSignal* **or** *edgeKeyword resetSignal* **or** *edgeKeyword setSignal* **)**

EdgeKeyword is posedge for active-high set or reset (or positive-edge triggered clock) or negedge for active-low set or reset (or negative-edge triggered clock).

You can list the signals in any order.

Example: Event Control

```
// Asynchronous, active-high set (rising-edge clock)
always @(posedge clk or posedge set)

// Asynchronous, active-low reset (rising-edge clock)
always @(posedge clk or negedge reset)

// Asynchronous, active-low set and active-high reset
// (rising-edge clock)
always @(posedge clk or negedge set or posedge reset)
```

Example: always Block Template with Asynch, Active-high reset, set

```
always @(posedge clk or posedge set or posedge reset)
begin
   if (reset) begin

      /* Set the outputs to zero */

   end else if (set) begin
```

```
      /* Set the outputs to one */

   end else begin

      /* Clocked logic */
   end
end
```

Example: flip-flop with Asynchronous, Active-high reset and set

```
module dff1 (q, qb, d, clk, set, reset);
input d, clk, set, reset;
output q, qb;
// Declare q and qb as reg because assigned inside always
reg q, qb;

always @(posedge clk or posedge set or posedge reset)
begin
   if (reset) begin
      q <= 0;
      qb <= 1;
   end else if (set) begin
      q <= 1;
      qb <= 0;
   end else begin
      q <= d;
      qb <= ~d;
   end
end
endmodule
```

For simple, single variable flip-flops, the following template can be used.

```
always @(posedge clk or posedge set or posedge reset)

q = reset ? 1'b0 : set ? 1'b1 : d;
```

Synchronous Sets and Resets

Synchronous sets and resets set flip-flop outputs to logic 1 or 0 (respectively) on an active clock edge.

Copyright © 2013 Synopsys, Inc.

Do not list the set and reset signal names in the event expression of an always block so they do not trigger the always block to execute upon changing. Instead, trigger the always block on the active clock edge, and check the reset and set inside the always block first.

RTL View Primitives

The Verilog compiler can detect and extract the following flip-flops with synchronous sets and resets and display them in the RTL schematic view:

- sdffr – flip-flop with synchronous reset
- sdffs – flip-flop with synchronous set
- sdffrs – flip-flop with both synchronous set and reset
- sdffpat – vectored flip-flop with synchronous set/reset pattern
- sdffre – enabled flip-flop with synchronous reset
- sdffse – enabled flip-flop with synchronous set
- sdffpate – enabled, vectored flip-flop with synchronous set/reset pattern

You can check the name (type) of any primitive by placing the mouse pointer over it in the RTL view: a tooltip displays the name. The following figure shows flip-flops with synchronous sets and resets.

Event Control Syntax

always @ (*edgeKeyword clockName* **)**

In the syntax line, *edgeKeyword* is posedge for a positive-edge triggered clock or negedge for a negative-edge triggered clock.

Example: Event Control

```
// Positive edge triggered
always @(posedge clk)

// Negative edge triggered
always @(negedge clk)
```

Example: always Block Template with Synchronous, Active-high reset, set

```
always @(posedge clk)
begin
   if (reset) begin
      /* Set the outputs to zero */
   end else if (set) begin
      /* Set the outputs to one */
   end else begin
      /* Clocked logic */
   end
end
```

Example: D Flip-flop with Synchronous, Active-high set, reset

```
module dff2 (q, qb, d, clk, set, reset);
input d, clk, set, reset;
output q, qb;
reg q, qb;

always @(posedge clk)
begin
   if (reset) begin
      q <= 0;
      qb <= 1;
   end else if (set) begin
      q <= 1;
      qb <= 0;
   end else begin
```


Copyright © 2013 Synopsys, Inc.

```
      q <= d;
      qb <= ~d;
   end
end
endmodule
```

SRL Inference

Sequential elements can be mapped into SRLs using an initialization assignment in the Verilog code. You can now infer SRLs with initialization values. Enable the System Verilog option on the Verilog tab of the Implementation Options dialog box before you run synthesis.

This is an example of a SRL with no resets. It has four 4-bit wide registers and a 4-bit wide read address. Registers shift when the write enable is 1.

```
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[3:0]='{4'h0,4'h1,4'h2,4'h3};
integer i;

always @(posedge clk) begin
   if (enable == 1) begin
      for (i=3; i>0; i=i-1) begin
         regBank[i] <= regBank[i-1];
      end
   regBank[0] <= dataIn;
   end
end

assign result = regBank[addr];
endmodule
```

Verilog State Machines

This section describes Verilog state machines: guidelines for using them, defining state values, and dealing with asynchrony. The topics include:

- State Machine Guidelines, on page 705
- State Values, on page 707
- Asynchronous State Machines, on page 708
- Limited RAM Resources, on page 710
- Additional Glue Logic, on page 710
- Synchronous READ RAMs, on page 711
- Verilog Guidelines for Byte-Enable Block RAMs, on page 741
- Verilog Guidelines for Byte-Wide Write Enable RAMs, on page 742
- Multi-Port RAM Extraction, on page 711
- Stratix III LUTRAM Verilog Examples, on page 717

State Machine Guidelines

A finite state machine (FSM) is hardware that advances from state to state at a clock edge.

The synthesis tool works best with synchronous state machines. You typically write a fully synchronous design and avoid asynchronous paths such as paths through the asynchronous reset of a register. See *Asynchronous State Machines*, on page 708, for information about asynchronous state machines.

- The state machine must have a synchronous or asynchronous reset, to be inferred. State machines must have an asynchronous or synchronous reset to set the hardware to a valid state after power-up, and to reset your hardware during operation (asynchronous resets are available freely in most FPGA architectures).
- You can define state machines using multiple event controls in an always block only if the event control expressions are identical (for example, @(posedge clk)). These state machines are known as implicit state machines. However it is better to use the explicit style described here and shown in *Example – FSM Coding Style*, on page 706.

Copyright © 2013 Synopsys, Inc.

- Separate the sequential from the combinational always block statements. Besides making it easier to read, it makes what is being registered very obvious. It also gives better control over the type of register element used.
- Represent states with defined labels or enumerated types.
- Use a case statement in an always block to check the current state at the clock edge, advance to the next state, then set the output values. You can use if statements in an always block, but stay with case statements, for consistency.
- Always use a default assignment as the last assignment in your case statement and set the state variable to 'bx. See *Example: default Assignment*, on page 706.
- Set encoding style with the syn_encoding directive. This attribute overrides the default encoding assigned during compilation. The default encoding is determined by the number of states, except in the case of Xilinx designs, where a non-default encoding is implemented if it produces better results. See *syn_encoding Values*, on page 1305 for a list of default and other encodings. When you specify a particular encoding style with syn_encoding, that value is used during the mapping stage to determine encoding style.

  object **/*synthesis syn_encoding="sequential"*/;**

  See *syn_encoding*, on page 1305, for details about the syntax and values.

  One-hot implementations are not always the best choice for state machines, even in FPGAs and CPLDs. For example, one-hot state machines might result in larger implementations, which can cause fitting problems. An example in an FPGA where one-hot implementation can be detrimental is a state machine that drives a large decoder, generating many output signals. In a 16-state state machine, for instance, the output decoder logic might reference sixteen signals in a one-hot implementation, but only four signals in a sequential representation.

Example – FSM Coding Style

Example: default Assignment

```
default: state = 'bx;
```

Assigning 'bx to the state variable (a “don't care” for synthesis) tells the tool that you have specified all the used states in your case statement. Any remaining states are not used, and the synthesis tool can remove unnecessary decoding and gates associated with the unused states. You do not have to add any special, non-Verilog directives.

If you set the state to a used state for the default case (for example, default state = state1), the tool generates the same logic as if you assign 'bx, but there will be pre- and post-synthesis simulation mismatches until you reset the state machine. These mismatches occur because all inputs are unknown at start up on the simulator. You therefore go immediately into the default case, which sets the state variable to state1. When you power up the hardware, it can be in a used state, such as state2, and then advance to a state other than state1. Post-synthesis simulation behaves more like hardware with respect to initialization.

State Values

In Verilog, you must give explicit state values for states. You do this using parameter or `define statements. It is recommended that you use parameter, for the following reasons:

- The `define is applied globally whereas parameter definitions are local. With global `define definitions, you cannot reuse common state names that you might want to use in multiple designs, like RESET, IDLE, READY, READ, WRITE, ERROR and DONE. Local definitions make it easier to reuse certain state names in multiple FSM designs. If you work around this restriction by using `undef and then redefining them with `define in the new FSM modules, it makes it difficult to probe the internal values of FSM state buses from a testbench and compare them to state names.
- The tool only displays state names in the FSM Viewer if they are defined using parameter.

Example 1: Using Parameters for State Values

```
parameter state1 = 2'h1, state2 = 2'h2;
...
current_state = state2; // Setting current state to 2'h2
```


Copyright © 2013 Synopsys, Inc.

Example 2: Using `define for State Values

```
`define state1     2'h1
`define state2     2'h2
...
current_state = `state2; // Setting current state to 2'h2
```

Asynchronous State Machines

Avoid defining asynchronous state machines in Verilog. An asynchronous state machine has states, but no clearly defined clock, and has combinational loops.

Do not use tools to design asynchronous state machines; the synthesis tool might remove your hazard-suppressing logic when it performs logic optimization, causing your asynchronous state machines to work incorrectly.

The synthesis tool displays a “Found combinational loop” warning message for an asynchronous state machine when it detects combinational loops in continuous assignment statements, always blocks, and built-in gate-primitive logic.

To create asynchronous state machines, do one of the following:

- To use Verilog, make a netlist of technology primitives from your target library. Any instantiated technology primitives are left in the netlist, and not removed during optimization.
- Use a schematic editor (and not Verilog) for the asynchronous state machine part of your design.

The following asynchronous state machine examples generate warning messages.

Example – Asynchronous FSM with Continuous Assignment

Example – Asynchronous FSM with an always Block

Verilog Guidelines for RAM Inference

This section provides Verilog guidelines and examples for coding RAMs for synthesis. It covers the following topics:

- RAM Inference from Verilog Code, on page 709
- Limited RAM Resources, on page 710
- Additional Glue Logic, on page 710
- Synchronous READ RAMs, on page 711
- Multi-Port RAM Extraction, on page 711

For additional information about coding Altera and Xilinx RAMs, see *Verilog Coding for Altera RAM Inference*, on page 717 and *Verilog Coding for Xilinx RAM Inference*, on page 738.

RAM Inference from Verilog Code

The synthesis tool can infer synchronous and synchronously resettable RAMs from your Verilog source code and, where appropriate, generate technology-specific single-port or dual-port RAMs. You do not need any special input in your source code like attributes or directives.

The RTL-level RAM inferred by the compiler always has an asynchronous READ. During synthesis, the mappers convert this to the appropriate technology-specific RAM primitives. Depending on the technology used, the synthesized design can include RAM primitives with either synchronous or asynchronous READs. See *Synchronous READ RAMs*, on page 711, for information on coding your Verilog description to ensure that technology-specific RAM primitives with synchronous READs are used.

The following table lists the supported technology-specific RAMs that the synthesis tool can generate.

Copyright © 2013 Synopsys, Inc.

Technology Vendor	Family	RAM Type
Microsemi	ProASIC, ProASIC^{PLUS}, ProASIC3/3E/3L	Single and dual port
Altera	Stratix, Stratix II, Stratix III, Stratix IV, Stratix V, Cyclone, Cyclone II, Cyclone III, Cyclone IV E, Cyclone IV GX	Single and dual port
Altera	Stratix, Stratix II, Stratix III, Stratix IV, Stratix V	Multi-port
Lattice	ORCA FPSC, ORCA Series 2, 3, MachXO, LatticeECP2M/ECP2, LatticeECP/EC, LatticeXP2/XP, LatticeSC/SCM	Single and dual port
Xilinx	Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, Virtex-6, Spartan-3, Spartan-6	Single and dual port
Xilinx	Spartan-II	Multi-port (true dual-port)

Example – RAM Inferencing

Limited RAM Resources

If your RAM resources are limited, you can designate additional instances of inferred RAMs as flip-flops and logic using the syn_ramstyle attribute. This attribute takes the string argument of registers, which you place on the RAM instance.

Additional Glue Logic

After inferring a RAM for some technologies, you might notice that the tool has generated a few additional components adjacent to the RAM. This glue logic ensures accuracy in your post place-and-route simulation results.

Copyright © 2013 Synopsys, Inc.

Synchronous READ RAMs

All RAM primitives inferred by the compiler (RTL view) have asynchronous READs, except for Altera Stratix RAMs, single-port Altera FLEX10K RAMs and single-port Xilinx Virtex block RAMs. For these exceptions, the compiler infers a synchronous RAM.

The following examples show how the READ address or the data output of the RAM can be registered to ensure generation of a synchronous READ RAM.

Example – READ Address Registered

Example – Data Output Registered

Multi-Port RAM Extraction

The compiler extracts multi-write port RAMs, creating an nram primitive which can have n write ports. The nrams are implemented differently depending on your technology. For information about the implementations, see the appropriate vendor chapter.

The following aspects of multi-port RAM extraction are described here:

- Prerequisites for Multi-Port RAM Extraction, on page 712
- Write Processes and Multi-Port RAM Inference, on page 712
- Multi-Port RAM Recommended Coding Style, on page 713
- Multi-Port RAM Examples, on page 714

Copyright © 2013 Synopsys, Inc.

Prerequisites for Multi-Port RAM Extraction

In order to infer the nram, you must do the following:

- The compiler will infer the nram even if you do not register the read address, as subsequent examples show. However, for many technology families, you must register the read address for the nram to be implemented as vendor RAMs.
- If the writes are in different processes, you no longer need to specify no_rw_check for the syn_ramstyle attribute in order to infer the RAM. Note that the simulator produces x's when the two writes occur at the same time while the synthesis tool assumes that one write gets priority over the other based on the code.

Write Processes and Multi-Port RAM Inference

The compiler infers the multi-port RAMs from the write processes as follows:

- When all the writes are made in one process, there are no address conflicts, and the compiler generates an nram. The following example results in an nram with two write ports, one with write address waddr0 and the other with write address waddr1:

```
always @(posedge clk)
begin
   if(we1) mem[waddr0] <= data1;
   if(we2) mem[waddr1] <= data2;
end
```

waddr0:	we1 && (!we2 || (we2 && waddr0 != waddr1))
waddr1	we2

- When writes are made in multiple processes, the software does not infer a multi-port RAM unless there is an explicit syn_ramstyle attribute. If there is no attribute, the software does not infer a RAM. If there is an attribute, the software infers a RAM with multiple write ports. You might get a warning about simulation mismatches when the two addresses are the same.

  In the following case, the compiler infers an nram with two write ports because of the syn_ramstyle attribute. The write enable associated with waddr0 is we1 and the write enable associated with waddr1 is we2.

```
reg [1:0] mem [7:0] /* synthesis syn_ramstyle="no_rw_check" */ ;

always @(posedge clk1)
begin
   if(we1) mem[waddr0] <= data1;
end

always @(posedge clk2)
begin
   if(we2) mem[waddr1] <= data2;

end
```

Multi-Port RAM Recommended Coding Style

Currently, you must use a registered read address when you code the RAM. If you do not, you could get error messages. The following is an example of how to code the RAM:

```
module ram(data0, data1, waddr0, waddr1, we0,we1,
   clk0, clk1, q0, q1);
parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;

input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;
reg [addr_width-1:0] reg_addr0, reg_addr1;
reg [d_width-1:0] mem [mem_depth-1:0] /* synthesis syn_ramstyle="no_rw_check" */;

assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];

always @(posedge clk0)
begin
   reg_addr0 <= waddr0;
   if (we0)
      mem[waddr0] <= data0;
end
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge clk1)
begin
   reg_addr1 <= waddr1;
   if (we1)
      mem[waddr1] <= data1;
end

endmodule
```

There are two situations which can result in the error message: "@E:MF216 : ram.v(29) | Found NRAM mem_1[7:0] with multiple processes":

- An nram with two clocks and two write addresses has syn_ramstyle set to a value of registers. The software cannot implement this, because there is a physical FPGA limitation that does not allow registers with multiple writes.
- You have a registered output for an nram with two clocks and two write addresses.

Multi-Port RAM Examples

This section consists of the following examples of multi-port RAM extraction:

- A RAM with Two Write Ports, on page 714
- Another RAM with Two Write Ports, on page 715
- A RAM with Three Write Ports, on page 715
- A Two-Write Port RAM, on page 716
- Two Write Processes with Two Different Clocks, on page 716
- Two Write Processes with One Clock, on page 716

A RAM with Two Write Ports

```
module ram0(input [1:0] data1, data2, input [2:0] waddr0,
   waddr1, addr, input we1, we2, clk, output reg [1:0] q);
reg [1:0] mem [7:0]/* synthesis syn_ramstyle="no_rw_check" */;

always @(posedge clk)
begin
   if(we1) mem[waddr0] <= data1;
   if(we2) mem[waddr1] <= data2;
      q = mem[addr];
end
endmodule
```

Another RAM with Two Write Ports

```
module ram0(input [1:0] data0, data1, data2, data3,
   input [1:0] waddr0, waddr1, waddr2, waddr3, addr,
   input sel, sel2, we1, we2, clk, output reg [1:0] q);
reg [1:0] mem [3:0]/* synthesis syn_ramstyle="no_rw_check" */;

always @(posedge clk)
begin
   if(sel)
      mem[waddr0] <= data0;
   else if(sel2)
   begin
      if(we1) mem[waddr1] <= data1;
      if(we2) mem[waddr2] <= data2;
   end
   else
      mem[waddr3] <= data3;
      q = mem[addr];
end
endmodule
```

A RAM with Three Write Ports

```
module ram0(input [1:0] data0, data1, data2, data3, data4,
   input [1:0] waddr0, waddr1, waddr2, waddr3, waddr4, addr,
   input sel, sel2, we1, we2, clk, output reg [1:0] q);
reg [1:0] mem [3:0]/* synthesis syn_ramstyle="no_rw_check" */;

always @(posedge clk)
begin
   if(sel)
      mem[waddr0] <= data0;
   else if(sel2)
   begin
      if(we1) mem[waddr1] <= data1;
      if(we2) mem[waddr2] <= data2;
   end
   else
      mem[waddr3] <= data3;
      mem[waddr4] <= data4;
      q = mem[addr];
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

A Two-Write Port RAM

```
module ram0(input [1:0] data0, data1, data2, data3,
   input [1:0] waddr0, waddr1, waddr2, waddr3, addr, input sel,
   sel2, clk, output [1:0] q);
reg [1:0] mem [3:0]; /* synthesis syn_ramstyle="no_rw_check" */
always @(posedge clk)
begin
   if(sel)
      mem[waddr0] <= data0;
   else
      mem[waddr1] <= data1;
   if(sel2)
      mem[waddr2] <= data2;
   else
      mem[waddr3] <= data3;
end
assign q = mem[addr];
endmodule
```

Two Write Processes with Two Different Clocks

This example has the writes in two processes, and generates a two-write port RAM with two different clocks.

```
module ram0(input [1:0] data1, data2, input [2:0] waddr1, addr,
input we1, we2, clk1, clk2, output reg [1:0] q);

reg [1:0] mem [7:0] /* synthesis syn_ramstyle="no_rw_check" */ ;

always @(posedge clk1)
begin
   if(we1)
      mem[addr] <= data1;
      q = mem[addr];
end

always @(posedge clk2)
begin
   if(we2) mem[waddr1] <= data2;
end

endmodule
```

Two Write Processes with One Clock

This example has the writes in two processes, and generates a two-write port RAM that uses only one clock.

```
module ram0(input [1:0] data1, data2, input [2:0] waddr0, waddr1,
   addr, input we1, we2, clk, output reg [1:0] q);
reg [1:0] mem [7:0] /* synthesis syn_ramstyle="no_rw_check" */ ;

always @(posedge clk)
begin
   if(we1) mem[waddr0] <= data1;
   q = mem[addr];
end

always @(posedge clk)
begin
   if(we2) mem[waddr1] <= data2;
end

endmodule
```

Verilog Coding for Altera RAM Inference

See the following for Verilog coding guidelines and examples for Altera RAMs:

- Stratix III LUTRAM Verilog Examples, on page 717
- Stratix, Arria and Cyclone Dual-Port Verilog RAM Examples, on page 720
- Verilog Coding for StratixIII and StratixIV RAMs with Control Signals, on page 724
- Verilog Coding for Cyclone III RAMs with Control Signals, on page 730

Stratix III LUTRAM Verilog Examples

The following are Verilog code examples that result in memories being extracted and mapped to LUTRAM. See *Inferring Altera Stratix III LUTRAMs*, on page 383 in the *User Guide* for a procedure.

- Synchronous Read Single-Port RAM, Unregistered Output, on page 718
- Asynchronous Read Single-Port RAM, Unregistered Output, on page 718
- Simple Synchronous Dual-Port RAM, Unregistered Output, on page 719

Copyright © 2013 Synopsys, Inc.

- Simple Asynchronous Dual-Port RAM, Unregistered Output, on page 720

Synchronous Read Single-Port RAM, Unregistered Output

The following code implements a single-port LUTRAM in Stratix III designs:

```
//SPRAM, with registered address and unregistered output,
//synchronous read

module test (clk,we,addr,datain,dataout);
parameter addr_width = 6;
parameter data_width = 10;
input clk,we;
input [addr_width - 1 : 0] addr;
input [data_width - 1 : 0] datain;
output [data_width - 1 : 0] dataout;

reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0]
   /* synthesis syn_ramstyle = "MLAB"*/;
reg [addr_width - 1 : 0]   addr_reg;

always @ (posedge clk)begin
   addr_reg <= addr;
   if(we) mem[addr] <= datain;
end

assign dataout = mem[addr_reg];
endmodule
```

Asynchronous Read Single-Port RAM, Unregistered Output

The following code implements a single-port LUTRAM in Stratix III designs:

```
//SPRAM, with registered output, asynchronous read

module test (clk,we,addr,datain,dataout);
parameter addr_width = 6;
parameter data_width = 10;
input clk,we;
input [addr_width - 1 : 0] addr;
input [data_width - 1 : 0] datain;
output [data_width - 1 : 0] dataout;

reg [data_width - 1 : 0] mem [(2**addr_width) - 1  : 0]
   /* synthesis syn_ramstyle = "MLAB"*/;
```

```
always @ (posedge clk)
begin
   if(we) mem[addr] <= datain;
end

assign dataout = mem[addr];

endmodule
```

Simple Synchronous Dual-Port RAM, Unregistered Output

The following code implements a simple dual-port LUTRAM in Stratix III designs:

```
//SDPRAM, with registered address and unregistered output
//synchronous read

module test(clk,we,wraddr,rdaddr,datain,dataout);
parameter addr_width = 6;
parameter data_width = 10;
input clk,we;
input [data_width-1:0] datain;
input [addr_width-1: 0] wraddr,rdaddr;
output [data_width-1:0] dataout;

reg [data_width-1:0] mem [(2**addr_width)-1:0]
   /* synthesis syn_ramstyle = "MLAB"*/;

reg [addr_width-1: 0] rdaddr_reg;
initial $readmemb("mem64x10.ini",mem);

always @ (posedge clk)
begin
   if(we) mem[wraddr] <= datain;
end

always @ (posedge clk)
begin
   rdaddr_reg <= rdaddr;
end

assign dataout = mem[rdaddr_reg];

endmodule
```


Copyright © 2013 Synopsys, Inc.

Simple Asynchronous Dual-Port RAM, Unregistered Output

The following code implements a simple dual-port LUTRAM in Stratix III designs:

```
//SDPRAM, with registered output, asynchronous read

module test(clk,we,wraddr,rdaddr,datain,dataout);
parameter addr_width = 6;
parameter data_width = 10;

input clk,we;
input [data_width-1:0] datain;
input [addr_width-1: 0] wraddr,rdaddr;
output [data_width-1:0] dataout;

reg [data_width-1:0] mem [(2**addr_width)-1:0]
   /* synthesis syn_ramstyle = "MLAB"*/;

always @ (posedge clk)
begin
   if(we) mem[wraddr] <= datain;
end

assign dataout = mem[rdaddr];

endmodule
```

Stratix, Arria and Cyclone Dual-Port Verilog RAM Examples

There are several ways to write RTL code to describe an Altera dual-port RAM. The software only maps an inferred RAM to Altera's dedicated synchronous RAM resources when it can construct hardware that exactly matches the behavior of the RTL.

The Altera dual-port RAM has two ports that can be used for read and write. The synthesis software maps to true dual-port (where both ports are used for reading and writing simultaneously) in the following cases:

- An inferred RAM with two writes and one read. The read shares an address with only one of the write ports
- Two inferred RAMs with exactly the same write addresses, clocks, and enables, but with different read address can be paired together and mapped

Copyright © 2013 Synopsys, Inc.

Altera dual-port RAM supports only two clocks, one for each port. If an inferred RAM with more than two write ports uses more than two clocks for read and write operations, the RAM is mapped to logic.

Some Verilog statements with blocking assignments may not be mapped to block RAM, so avoid blocking statements when modeling RAMs.

This section describes *Read-First Dual-Port RAM*, on page 721 and *Write-First Dual-Port RAM*, on page 723.

Read-First Dual-Port RAM

If your coding style permits mismatches between the RTL behavior and the hardware behavior when you simultaneously write to and read from the same address, the software does not map it to a RAM.

When a dual-port RAM is inferred in RTL using non-recommended coding styles, the software issues a warning message like this one:

```
"<memory> instance will be mapped into logic. Please refer to the
online help section of stratix_ram_block for the recommended
coding style to map to ALTSYNCRAM."
```

The following typical code for a Read First dual-port RAM illustrates this style that permits mismatches:

```
module Read_First_RAM   (read_clk, read_address, data_in,
   write_clk, write_enable, write_address, data_out);

parameter address_width = 10;
parameter data_width = 4;
parameter depth = 1024;

input read_clk, write_clk, write_enable;
input [address_width-1:0] read_address, write_address;
input [data_width-1:0] data_in;
output [data_width-1:0] data_out;

reg [data_width-1:0] mem [depth -1 : 0]
   /* synthesis syn_ramstyle="M4K" */;
reg [data_width-1:0] data_out;

always @(negedge write_clk)
   if(write_enable)
      mem[write_address] <= data_in;
```


Copyright © 2013 Synopsys, Inc.

```
always @(negedge read_clk)
   data_out <= mem[read_address];

endmodule
```

To map this RAM, you must modify the HDL code so that the RTL behavior matches the hardware behavior. Make the value of the read data a don't care for simultaneous read and write operations to the same address. This ensures that the synthesis software infers a RAM and saves the user from spending time debugging the differences between the RTL RAM and hardware RAM behaviors. The following example shows the recommended Verilog coding style.

Verilog Altera Read-First Dual-Port Example

```
module Read_First_RAM (
   read_clk,
   read_address,
   data_in,
   write_clk,
   write_enable,
   write_address,
   data_out );

parameter address_width = 10;
parameter data_width = 4;
parameter depth = 1024;

input read_clk, write_clk, write_enable;
input [address_width-1:0] read_address, write_address;
input [data_width-1:0] data_in;
output [data_width-1:0] data_out;

reg [data_width-1:0] mem [depth -1 : 0]/* synthesis
   syn_ramstyle="M4K" */;
reg [data_width-1:0] data_out;

reg [address_width-1:0] reg_write_address;
reg reg_write_enable;
```

```
always @(posedge write_clk)
begin
   reg_write_address <= write_address;
   reg_write_enable <= write_enable;
end
always @( posedge write_clk)
   if(write_enable)
      mem[write_address] <= data_in;

always @( posedge read_clk)
   data_out <= (reg_write_address == read_address) &&
      reg_write_enable ? 'bx : mem[read_address];

endmodule
```

Write-First Dual-Port RAM

For a Write First dual-port RAM coded as shown below, the software maps the inferred RAM to the Altera RAM block because the RTL and hardware behaviors are the same.

Verilog Altera Write-First Dual-Port Example

```
module Write_First_RAM (read_clk, read_address, data_in,
   write_clk, write_enable, write_address, data_out);

parameter address_width = 10;
parameter data_width = 4;
parameter depth = 1024;

input read_clk, write_clk, write_enable;
input [address_width-1:0] read_address, write_address;
input [data_width-1:0] data_in;
output [data_width-1:0] data_out;

reg [data_width-1:0] mem [depth -1 : 0]
   /* synthesis syn_ramstyle="M4K" */;
reg [address_width-1:0] read_address_reg;

always @(posedge write_clk)
   if(write_enable)
      mem[write_address] = data_in;

always @(posedge read_clk)
   read_address_reg = read_address;
   assign data_out = mem[read_address_reg];

endmodule
```


Copyright © 2013 Synopsys, Inc.

Verilog Coding for StratixIII and StratixIV RAMs with Control Signals

The enable pins on Altera StratixIII and StratixIV RAM blocks provide built-in functionality for control signals. The synthesis tool extracts control signals for single-port and simple dual-port RAMs with a single common clock, and maps the logic to StratixIII and StratixIV LutRAMs instead of registers. This avoids inferring extra logic, and improves resource utilization and QOR. The control signals can be read enables, write enables and clock enables.

To extract RAMs with control signals, the design must meet the following prerequisites:

- For clock enables with read enable and write enable, the design must only have a single clock.
- In write-first mode, you currently cannot have two clock enables with read enable and write enable.
- Currently , the design cannot have asynchronous clear, dual clocks.

See the following for additional information and code examples:

- Guidelines for Read-First Mode, on page 724
- Guidelines for Write-First Mode, on page 725
- Dual-Port RAM in Read-First Mode with Read Enable and Clock Enable, on page 725
- Single-Port RAM in Write-First Mode with Clock Enable Controlling Read, on page 726
- Dual-Port RAM with Read Enable, Write Enable and Common Clock Enable, on page 728

Guidelines for Read-First Mode

These guidelines describe the recommended coding style for read-first mode:

- To implement the clock enable and a read enable controlling the read operation, register the RAM output with two registers, as shown in *Dual-Port RAM in Read-First Mode with Read Enable and Clock Enable*, on page 725. A read enable controls the first register output, and a clock enable controls the second register stage.
- To implement the clock enable with read and write enables, the output register must be enabled by both clkenable and the read enable. The

write enable for the RAM should be fed by both the write enable pin and clkenable.

Guidelines for Write-First Mode

These guidelines describe the recommended coding style for read-first mode:

- To extract read enable in write-first mode, register the read enable signal should be registered. For read-first mode, the read enable is inferred normally.
- If you do not register the read enable signal in write-first mode, it is mapped to one of the clock enable pins "ena0/ena1/ena2/ena3", depending on the functionality, and the read enable is tied to 1.
- If you use a clock enable in write-first mode, the read enable register must use the clock enable as the enable for that register.
- Currently only common clock enables with read enable and write enable are supported for write-first mode.

Dual-Port RAM in Read-First Mode with Read Enable and Clock Enable

The following example shows code for a simple dual-port RAM in read-first mode with read enable and clock enable, and the altsyncram that is inferred.

```
module test (addra,addrb,clka,din1a,out1a,we,re,ena0);
input [8:0] addra, addrb;
input   clka;
input we,re,ena0;
input [17:0] din1a;
output reg [17:0] out1a;
reg [17:0] out1a_reg1;
reg [17:0] ram1 [511:0] ;

always @(posedge clka)
   if (re)
      out1a_reg1 <= ram1[addra[8:0]];

//Write code Port A
always@(posedge clka)
begin
   if (we)
      ram1[addrb[8:0]] <= din1a;
end
```


Copyright © 2013 Synopsys, Inc.

```
always@(posedge clka)
begin
   if (ena0)
      out1a <= out1a_reg1;
end

endmodule
```

Single-Port RAM in Write-First Mode with Clock Enable Controlling Read

The following example shows code for a single-port RAM in write-first Mode with a clock enable controlling the read operation. In this code, the clock enable signal is mapped to the ena0 pin, and read enable is tied to 1.

```
module test (addra,clka,din1a,out1a,we,ena);
input [12:0] addra;
input clka;
```

```
input we,ena;
input [17:0] din1a;
output reg [17:0] out1a;
reg [12:0] addra_reg;
reg re_reg;
reg [17:0] ram1 [8191:0];

always@(posedge clka )
begin
   if (ena)
      out1a <= ram1[addra_reg[12:0]];
end

//Write code Port A
always@(posedge clka)
begin
   if (we)
      ram1[addra[12:0]] <= din1a;
end

always@(posedge clka)
begin
   addra_reg <= addra;
end

endmodule
```


Copyright © 2013 Synopsys, Inc.

Dual-Port RAM with Read Enable, Write Enable and Common Clock Enable

The following example shows code for a simple dual-port RAM with read enable, write enable, and common clock enable:

```
module test     (addra,addrb,clka,din1a,out1a,we,re,ena0);
parameter addr_width = 12;
parameter data_width = 36;
parameter mem_depth = 4096;

input [addr_width - 1:0] addra, addrb;
input clka;
input we,re,ena0;
input [data_width -1 :0] din1a;
output reg [data_width -1 :0] out1a;
reg [addr_width -1 :0] addra_reg;
reg re_reg;
reg [data_width -1:0] ram1 [mem_depth - 1:0] ;
```

```
always@(posedge clka)
   if (re_reg & ena0)
      out1a <= ram1[addra_reg[addr_width - 1:0]];

//Write code Port A
always@(posedge clka)
begin
   if (we & ena0)
ram1[addrb[addr_width - 1:0]] <= din1a;
end

always@(posedge clka)
begin
   if (ena0)
      addra_reg <= addra;
end

always@(posedge clka)
begin
   if (ena0)
      re_reg <= re;
end

endmodule
```


Copyright © 2013 Synopsys, Inc.

Verilog Coding for Cyclone III RAMs with Control Signals

You can extract the control signals and take advantage of the built-in functionality for control signals provided with Cyclone III RAMs. The synthesis tool extracts control signals for single-port, simple dual-port and true dual-port RAMs and maps the logic to Cyclone III dedicated RAMs instead of registers.

The tool maps the following control signals: read enables, write enables, clock enables, two clocks and asynchronous clear.

See the following for details:

- Guidelines for Extracting RAMs with Control Signals, on page 731
- Example of Simple Dual-port RAM (Read-first) with Enables, on page 732

- Example of Simple Dual-port RAM with Two Clocks, on page 733
- Example of True Dual-port RAM with Two Clocks, on page 735

Guidelines for Extracting RAMs with Control Signals

The following are the tool guidelines for extracting Cyclone III RAMs with control signals:

- For true dual-port RAMs, the tool supports RAMs with one write, two read processes.
- The tool supports different clock modes as follows:

Single port, simple dual-port, and true dual port	Input Output clock mode
Simple dual-port	Read-write clock mode
True dual-port	Independent clock mode

- If you have a negative level-sensitive signal combined with an enable, the software might not extract a RAM.
- Write-first mode only supports common clock enables with read enable and write enable.
- In Write-first mode, the tool does not support two clock enables with read enable and write enable.
- Register the read enable signal if you want it to be extracted in Write-first mode. In Read-first mode, the tool infers the read enable normally.

  If you do not register the read enable signal in Write-first mode, the tool maps it to one of the clock enable pins (ena0/ena1/ena2/ena3) depending on the functionality, and ties the read enable to 1.
- To use a clock enable in Write-first mode, the read enable register must use the clock enable as the enable for that register.
- To implement the clock enable and read enable controlling read operation in Read-first mode, register the output of RAM with two stage registers. The read enable controls the first register output and the clock enable controls the second register stage. See *Single-Port RAM in Write-First Mode with Clock Enable Controlling Read*, on page 726.

Copyright © 2013 Synopsys, Inc.

- To implement the clock enable with read and write enables in Read-first mode, the output register must be enabled by both clock enable and read enable. In addition, the write enable for the RAM should be fed by both the write enable pin and the clock enable.

Example of Simple Dual-port RAM (Read-first) with Enables

The following code implements a simple dual-port Cyclone III RAM in Read-first mode with read and clock enables, as shown in the subsequent figure.

```
module test (addra,addrb,clka,din1a,out1a,we,re,ena0);

input [8:0] addra, addrb;
input clka;
input we,re,ena0;
input [17:0] din1a;
output reg [17:0] out1a;
reg [17:0] out1a_reg1;

reg [17:0] ram1 [511:0] ;

always @(posedge clka)
   if (re)
      out1a_reg1 <= ram1[addra[8:0]];

//Write code Port A
always @(posedge clka)
begin
   if (we)
ram1[addrb[8:0]] <= din1a;
end

always @(posedge clka)
begin
   if (ena0)
      out1a <= out1a_reg1;
end

endmodule
```

Example of Simple Dual-port RAM with Two Clocks

The following code implements a simple dual-port Cyclone III RAM in Read-write clock and asynchronous clear at the output. The subsequent figure shows the implementation.

```
module test (addra,addrb,clka,clkb,aclr,din1a,out1a,ena,we);

input [9:0] addra,addrb;
input clka,clkb,aclr;
input ena,we;
input [8:0] din1a;
output reg [8:0] out1a;
```


Copyright © 2013 Synopsys, Inc.

```
reg [9:0] addra_reg;

reg [8:0] ram1 [1023:0];

//ram code
//Read code Port A
always @(posedge clkb or negedge aclr)
begin
   if(!aclr)
      out1a <= 0;
   else if (ena)
   out1a <= ram1[addra_reg[9:0]];
end

//Write code Port B
always @(posedge clka)
begin
   if (ena & we)
ram1[addrb[9:0]] <= din1a;
end

always @(posedge clkb)
begin
   if(ena)
      addra_reg <= addra;
end

endmodule
```

Copyright © 2013 Synopsys, Inc.

Example of True Dual-port RAM with Two Clocks

The following is an example of a true dual-port RAM with two clocks in independent clock mode and with asynchronous clear at the outputs. The figure that follows shows the Cyclone III implementation of the code.

```
module test(clka,clkb,wea,rsta,rstb,addra,dataina,
   qa,ena,enb,ena2,enb2,addrb,qb);

parameter addr_width = 10;
parameter data_width = 9;
```


Copyright © 2013 Synopsys, Inc.

```
input clka,clkb,wea,rsta,rstb,ena,enb,ena2,enb2;
input [data_width - 1 : 0] dataina;
input [addr_width - 1 : 0] addra,addrb;
output reg [data_width - 1 : 0] qa,qb;
reg [data_width - 1 : 0] qa_reg1,qb_reg1;
reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0];

always @ (posedge clka)
begin
   if(wea & ena) mem[addra] <= dataina;
end

always @ (posedge clka)
begin
if(ena)
   qa_reg1 <= mem[addra];
end

always @ (posedge clkb)
begin
if(enb)
   qb_reg1 <= mem[addrb];
end

always @ (posedge clka or negedge rsta)
begin
   if(~rsta) qa <= 0;
   else if (ena2)
   begin
      qa <= qa_reg1;
   end
end

always @ (posedge clkb or negedge rstb)
begin
   if(~rstb) qb <= 0;
   else if (enb2)
   begin
      qb <= qb_reg1;
   end
end
endmodule
```

Copyright © 2013 Synopsys, Inc.

Verilog Coding for Xilinx RAM Inference

See the following for Verilog coding guidelines to implement Xilinx RAMs:

- Verilog Coding for Xilinx Single-port and Dual-port Block RAMs, on page 738
- Verilog Guidelines for Byte-Enable Block RAMs, on page 741
- Verilog Guidelines for Byte-Wide Write Enable RAMs, on page 742
- Virtex-5 RAM Verilog Examples, on page 743

Verilog Coding for Xilinx Single-port and Dual-port Block RAMs

Block RAMs support multiple clocks and use enables and resets to control RAM read and write operations. By default, the tool infers block SelectRAM, but you can force block RAM inference by specifying the syn_ramstyle attribute with a value of block_ram or no_rw_check. This attribute is specified for the register port that defines the RAM, or the label of the RAM component instance.

For syntax information, see *syn_ramstyle*, on page 1467; for usage, see *Specifying RAM Implementation Styles*, on page 375 of the *User Guide*. See *Verilog Guidelines for RAM Inference*, on page 709 for other details.

Single-Port Block RAMs

In single-port block RAMs, the enable signal has the highest priority. This table describes what happens with different enable and reset settings:

Enable	Reset	Results
Inactive	-	All read and write operations are disabled.
Active	Active	Write operations and memory contents are unaffected (data_out is 0).
Active	Inactive	If Write Enable is active, data_in goes to data_out and is updated in memory. If Write Enable is inactive, data_out reflects the memory contents

The following Verilog code defines the behavior of a single-port block RAM.

```
module RAMB4_S4 (data_out, ADDR, data_in, EN, CLK, WE, RST);
output[3:0] data_out;
input [7:0] ADDR;
input [3:0] data_in;
input EN, CLK, WE, RST;
reg   [3:0] mem [255:0];
reg   [3:0] data_out;

always@(posedge CLK)
   if(EN)
      if(RST == 1)
         data_out <= 0;
      else
      begin
      if(WE == 1)
         data_out <= data_in;
      else
         data_out <= mem[ADDR];
      end

always @(posedge CLK)
   if (EN && WE) mem[ADDR] = data_in;
endmodule
```

For a code segment to map to a single-port block RAM, there must be a one-to-one correspondence between the following signals:

- Read and write clocks
- Read and write addresses
- Enable signal
- Write Enable signal

There are three inferred write modes, and they determine what appears at data_out when Write Enable is active.

Writefirst	data_in goes to data_out
Readfirst	memory goes to data_out
Nochange	data_out remains unchanged

Copyright © 2013 Synopsys, Inc.

The following table shows device support for the inferred modes:

Writefirst	All Virtex families and all Spartan families
Readfirst	All Virtex families; all Spartan-3 and later families
Nochange	All Virtex families; all Spartan-3 and later families

This table shows that a code segment that can be mapped to a single-port block RAM in a more recent Xilinx device family where you may not be able to map it to a block RAM with an earlier Xilinx family. These code segments are mapped to distributed RAM.

For information about implementing single-port RAMs, see *Specifying RAM Implementation Styles*, on page 375 and *Implementing Xilinx RAMs Automatically*, on page 379 in the *User Guide*.

Dual-Port Block RAMs

Except for additional address collision recovery logic, a dual-port RAM is essentially two independent single-port RAMs that share a common memory. Dual-port RAMs have only one write port.

The following segment of Verilog code defines the behavior of a dual-port block RAM for all Virtex families, and all Spartan-3 and later families (address collision recovery code is not shown).

```
module DP_Block_Ram (ADDRA, ADDRB, data_in, CLK, WEA, data_out);
output[3:0] data_out;
input [9:0] ADDRA;
input [9:0] ADDRB;
input [3:0] data_in;
input CLK, WEA;
reg [3:0] mem [1023:0] /* synthesis syn_ramstyle="block_ram" */;
reg [9:0] ADDRB_reg;

always@(posedge CLK)
   if(WEA)
      mem[ADDRA] = data_in;

always @(posedge CLK)
ADDRB_reg <= ADDRB;
assign data_out = mem[ADDRB_reg];
endmodule
```

Copyright © 2013 Synopsys, Inc.

For information about implementing dual-port RAMs, see *Specifying RAM Implementation Styles*, on page 375 and *Implementing Xilinx RAMs Automatically*, on page 379 in the *User Guide*.

Verilog Guidelines for Byte-Enable Block RAMs

For Xilinx designs, the tool infers byte enable pins automatically and maps it to block RAM, as long as it is coded using an if-else-if (mutually exclusive) style.

```
module test (clock,wren,rden,address,byteena,data,q);
input clock;
input wren;
input rden;
input [8:0] address;
input [3:0] byteena;
input [31:0] data;
output reg [31:0] q;
reg [31:0] mem [511:0]/* synthesis syn_ramstyle="block_ram"*/;

always @(posedge clock)
begin
   if (wren)
   begin
      if (byteena[3])
         mem[address][31:24] <= data[31:24];
      else if (byteena[2])
         mem[address][23:16] <= data[23:16];
      else if (byteena[1])
         mem[address][15:8] <= data[15:8];
      else if (byteena[0])
         mem[address][7:0] <= data[7:0];
   end
end

always @(posedge clock)
begin
   if (rden)
      q <= mem[address];
end

endmodule
```


Copyright © 2013 Synopsys, Inc.

Verilog Guidelines for Byte-Wide Write Enable RAMs

For Xilinx designs, the synthesis tool infers byte-wide write enable for RAMs with mutually exclusive writes and maps it to block RAMs. You can specify a dual-port RAM with only one read and one write.

True dual-port RAMs with byte-wide enables cannot be implemented in block RAM.

Example 1

```
module test (clock,address,byteena,data,q,qb,addressb,
   byteenab,datab,clockb);
input clock;
input clockb;
input [8:0] address;
input [8:0] addressb;
input [3:0] byteena;
input [3:0] byteenab;
input [31:0] data;
input [31:0] datab;
output  reg [31:0] q;
output  reg [31:0] qb;
reg [31:0] mem [255:0] ;

always @(posedge clock)  //write process
   begin
      if (byteena[3])
         mem[address][7:0] <= data[7:0];
      if (byteena[2])
         mem[address][15:8] <= data[15:8];
      if (byteena[1])
         mem[address][23:16] <= data[23:16];
      if (byteena[0])
         mem[address][31:24] <= data[31:24];
   end

always @(posedge clock)  //read process
   begin
      q <= mem[address];
   end ///end of always

endmodule
```

Example 2

```
module test (clock,address,byteena,data,q);
input clock;
input [8:0] address;
input [3:0] byteena;
input [31:0] data;
output reg [31:0] q;
reg [31:0] mem [255:0] /* synthesis syn_ramstyle = "block_ram,
   no_rw_check" */;

always @(posedge clock)  //write process
begin
   if (byteena[3])
      mem[address][7:0] <= data[7:0];
   else if (byteena[2])
      mem[address][15:8] <= data[15:8];
   else if (byteena[1])
      mem[address][23:16] <= data[23:16];
   else if (byteena[0])
      mem[address][31:24] <= data[31:24];
end

always @(posedge clock)  //read process
begin
   q <= mem[address];
end //end of always

endmodule
```

Virtex-5 RAM Verilog Examples

Virtex-5 RAM supports different Block RAM primitives: RAMB18, RAMB18_SDP, RAMB36, and RAMB36_SDP. Depending on how you code it, your logic can be mapped to different RAM primitives.

The following is an example of a simple, dual-port RAM in write-first mode, which the tool maps to RAMB18s and RAMB36s:

```
module test (addr,addw,we,clk,data_out,data_in);

parameter ADDR_WIDTH = 9;
parameter ADDW_WIDTH = 9;
parameter DATA_WIDTH = 256;
parameter MEM_DEPTH = 512;
```


Copyright © 2013 Synopsys, Inc.

```
input [ADDR_WIDTH-1:0]addr;
input [ADDW_WIDTH-1:0]addw;
input clk, we;
input [DATA_WIDTH-1 : 0]data_in;
output [DATA_WIDTH-1 : 0]data_out;
reg [ADDR_WIDTH-1:0]addr_reg;
reg [DATA_WIDTH-1 : 0]mem[MEM_DEPTH-1 :0];
assign  data_out = mem[addr_reg];

always @(posedge clk) begin
   addr_reg <= addr;
end

always @(posedge clk)
begin
   if (we)
      mem[addw] <= data_in;
end

endmodule
```

However, if you want to infer wider RAM primitives, you can write the code so that the RAM is written in read_first mode. The following example results in the RAM being implemented in RAMB36_SDP primitives.

```
module test (dout,waddr,raddr,din,we,clk);

parameter data_width = 256;
parameter address_width = 9;
parameter ram_size = 512;

output [data_width-1:0] dout;
input [data_width-1:0] din;
input [address_width-1:0] waddr, raddr;
input we, clk;
reg [data_width-1:0] dout;
reg [data_width-1:0] mem [ram_size-1:0]/* synthesis
   syn_ramstyle="block_ram" */;

always @(posedge clk)
begin
   dout <= mem[raddr];
end
```

```
always @(posedge clk)
begin
   if (we)
      mem[waddr] <= din;
end

endmodule
```

RAM Instantiation with SYNCORE

The SYNCORE Memory Compiler in the IP Wizard helps you generate HDL code for your specific RAM implementation requirements. For information on using the SYNCORE Memory Compiler, see *Specifying RAMs with SYNCore*, on page 939 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

ROM Inference

As part of BEST (Behavioral Extraction Synthesis Technology) feature, the synthesis tool infers ROMs (read-only memories) from your HDL source code, and generates block components for them in the RTL view.

The data contents of the ROMs are stored in a text file named rom.info. To quickly view rom.info in read-only mode, synthesize your HDL source code, open an RTL view, then push down into the ROM component.

Generally, the Synopsys FPGA synthesis tool infers ROMs from HDL source code that uses case statements, or equivalent if statements, to make 16 or more signal assignments using constant values (words). The constants must all be the same width.

Example

```
module rom(z,a);
output [3:0] z;
input [4:0] a;
reg [3:0] z;

always @(a) begin
   case (a)
      5'b00000 : z = 4'b0001;
      5'b00001 : z = 4'b0010;
      5'b00010 : z = 4'b0110;
      5'b00011 : z = 4'b1010;
      5'b00100 : z = 4'b1000;
      5'b00101 : z = 4'b1001;
      5'b00110 : z = 4'b0000;
      5'b00111 : z = 4'b1110;
      5'b01000 : z = 4'b1111;
      5'b01001 : z = 4'b1110;
      5'b01010 : z = 4'b0001;
      5'b01011 : z = 4'b1000;
      5'b01100 : z = 4'b1110;
      5'b01101 : z = 4'b0011;
      5'b01110 : z = 4'b1111;
      5'b01111 : z = 4'b1100;
      5'b10000 : z = 4'b1000;
      5'b10001 : z = 4'b0000;
```

Copyright © 2013 Synopsys, Inc.

```
      5'b10010 : z = 4'b0011;
      default : z = 4'b0111;
   endcase
end
endmodule
```

ROM Table Data (rom.info File)

```
Note: This data is for viewing only

ROM work.rom4(behave)-z_1[3:0]
address width: 5
data width: 4
inputs:
0: a[0]
1: a[1]
2: a[2]
3: a[3]
4: a[4]
outputs:
0: z_1[0]
1: z_1[1]
2: z_1[2]
3: z_1[3]

data:
00000 -> 0001
00001 -> 0010
00010 -> 0110
00011 -> 1010
00100 -> 1000
00101 -> 1001
00110 -> 0000
00111 -> 1110
01000 -> 1111
01001 -> 1110
01010 -> 0001
01011 -> 1000
01100 -> 1110
01101 -> 0011
01110 -> 0010
01111 -> 0010
10000 -> 0010
10001 -> 0010
10010 -> 0010
default -> 0111
```


Copyright © 2013 Synopsys, Inc.

ROM Initialization with Generate Block

The software supports conditional ROM initialization with the generate block, as shown in the following example:

```
generate
   if (INIT) begin
      initial
      begin
         $readmemb("init.hex",mem);
      end
   end
endgenerate
```

Instantiating Black Boxes in Verilog

Black boxes are modules with just the interface specified; internal information is ignored by the software. Black boxes can be used to directly instantiate:

- Technology-vendor primitives and macros (including I/Os).
- User-designed macros whose functionality was defined in a schematic editor, or another input source. (When the place-and-route tool can merge design netlists from different sources.)

Black boxes are specified with the syn_black_box directive. If the macro is an I/O, use black_box_pad_pin=1 on the external pad pin. The input, output, and delay through a black box are specified with special black box timing directives (see *syn_black_box*, on page 1235).

For most of the technology-vendor architectures, macro libraries are provided (in *installDirectory*/lib/*technology*/*family*.v) that predefine the black boxes for their primitives and macros (including I/Os).

Verilog simulators require a functional description of the internals of a black box. To ensure that the functional description is ignored and treated as a black box, use the translate_off and translate_on directives. See translate_off/translate_on, on page 1593 for information on the translate_off and translate_on directives.

If the black box has tristate outputs, you must define these outputs with a black_box_tri_pins directive (see black_box_tri_pins, on page 1197).

For information on how to instantiate black boxes and technology-vendor I/Os, see *Defining Black Boxes for Synthesis*, on page 330 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

PREP Verilog Benchmarks

PREP (Programmable Electronics Performance) Corporation distributes benchmark results that show how FPGA vendors compare with each other in terms of device performance and area. The following PREP benchmarks are included in the *installDirectory*/examples/verilog/common_rtl/prep:

- PREP Benchmark 1, Data Path (prep1.v)
- PREP Benchmark 2, Timer/Counter (prep2.v)
- PREP Benchmark 3, Small State Machine (prep3.v)
- PREP Benchmark 4, Large State Machine (prep4.v)
- PREP Benchmark 5, Arithmetic Circuit (prep5.v)
- PREP Benchmark 6, 16-Bit Accumulator (prep6.v)
- PREP Benchmark 7, 16-Bit Counter (prep7.v)
- PREP Benchmark 8, 16-Bit Pre-scaled Counter (prep8.v)
- PREP Benchmark 9, Memory Map (prep9.v)

The source code for the benchmarks can be used for design examples for synthesis or for doing your own FPGA vendor comparisons.

Copyright © 2013 Synopsys, Inc.

Hierarchical or Structural Verilog Designs

This section describes the creation and use of hierarchical Verilog designs:

- Using Hierarchical Verilog Designs, on page 751
- Creating a Hierarchical Verilog Design, on page 751
- synthesis Macro, on page 753
- text Macro, on page 754

Using Hierarchical Verilog Designs

The software accepts and processes hierarchical Verilog designs. You create hierarchy by instantiating a module or a built-in gate primitive within another module.

The signals connect across the hierarchical boundaries through the port list, and can either be listed by position (the same order that you declare them in the lower-level module), or by name (where you specify the name of the lower-level signals to connect to).

Connecting by name minimizes errors, and can be especially advantageous when the instantiated module has many ports.

Creating a Hierarchical Verilog Design

To create a hierarchical design:

1. Create modules.
2. Instantiate the modules within other modules. (When you instantiate modules inside of others, the ones that you have instantiated are sometimes called “lower-level modules” to distinguish them from the “top-level” module that is not inside of another module.)
3. Connect signals in the port list together across the hierarchy either “by position” or “by name” (see the examples, below).

Copyright © 2013 Synopsys, Inc.

Example: Creating Modules (Interfaces Shown)

```
module mux(out, a, b, sel); // mux
output [7:0] out;
input [7:0] a, b;
input sel;

// mux functionality

endmodule

module reg8(q, data, clk, rst); // Eight-bit register
output [7:0] q;
input [7:0] data;
input clk, rst;
// Eight-bit register functionality
endmodule

module rotate(q, data, clk, r_l, rst); // Rotates bits or loads
output [7:0] q;
input [7:0] data;
input clk, r_l, rst;
// When r_l is high, it rotates; if low, it loads data
// Rotate functionality
endmodule
```

Example: Top-level Module with Ports Connected by Position

```
module top1(q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;

// The order of the listed signals here will match
// the order of the signals in the mux module declaration.
mux mux_1 (mux_out, a, b, sel);
reg8 reg8_1 (reg_out, mux_out, clk, rst);
rotate rotate_1 (q, reg_out, clk, r_l, rst);

endmodule
```

Copyright © 2013 Synopsys, Inc.

Example: Top-level Module with Ports Connected by Name

```
module top2(q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;

/* The syntax to connect a signal "by name" is:
.<lower_level_signal_name>(<local_signal_name>)
*/
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));

/* Ports connected "by name" can be in any order */
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
rotate rotate_1 (.q(q), .data(reg_out), .clk(clk),
   .r_l(r_l), .rst(rst) );
endmodule
```

synthesis Macro

Use this text macro along with the Verilog `ifdef compiler directive to conditionally exclude part of your Verilog code from being synthesized. The most common use of the synthesis macro is to avoid synthesizing stimulus that only has meaning for logic simulation.

The synthesis macro is defined so that the statement `ifdef synthesis is true. The statements in the `ifdef branch are compiled; the stimulus statements in the `else branch are ignored.

Note: Because Verilog simulators do *not* recognize a synthesis macro, the compiler for your simulator will use the stimulus in the `else branch.

In the following example, an AND gate is used for synthesis because the tool recognizes the synthesis macro to be defined (as true); the assign c = a & b branch is taken. During simulation, an OR gate is used instead, because the simulator does not recognize the synthesis macro to be defined; the assign c = a | b branch is taken.

Copyright © 2013 Synopsys, Inc.

Note: A macro in Verilog has a nonzero value only if it is defined.

```
module top (a,b,c);
   input a,b;
   output c;
`ifdef synthesis
   assign c = a & b;
`else
   assign c = a | b;
`endif
endmodule
```

text Macro

The directive define creates a macro for text substitution. The compiler substitutes the text of the macro for the string *macroName*. A text macro is defined using arguments that can be customized for each individual use.

The syntax for a text macro definition is as follows.

textMacroDefinition **::= define** *textMacroName macroText*

textMacroName **::=** *textMacroIdentifier*[**(***formalArgumentList***)**]

formalArgumentList **::=** formalArgumentIdentifier **{,** formalArgumentIdentifier**}**

When formal arguments are used to define a text macro, the scope of the formal argument is extended to the end of the macro text. You can use a formal argument in the same manner as an identifier.

A text macro with one or more arguments is expanded by replacing each formal argument with the actual argument expression.

Example 1

```
`define MIN(p1, p2) (p1)<(p2)?(p1):(p2)

module example1(i1, i2, o);
input i1, i2;
output o;
reg o;
```

```
always @(i1, i2) begin
o = `MIN(i1, i2);
end
endmodule
```

Example 2

```
`define SQR_OF_MAX(a1, a2) (`MAX(a1, a2))*(`MAX(a1, a2))
`define MAX(p1, p2) (p1)<(p2)?(p1):(p2)

module example2(i1, i2, o);
input i1, i2;
output o;
reg o;

always @(i1, i2) begin
o = `SQR_OF_MAX(i1, i2);
end
endmodule
```

Example 3

Include File ppm_top_ports_def.inc

```
//ppm_top_ports_def.inc

// Single source definition for module ports and signals
// of PPM TOP.
// Input
`DEF_DOT `DEF_IN([7:0]) in_test1 `DEF_PORT(in_test1) `DEF_END
`DEF_DOT `DEF_IN([7:0]) in_test2 `DEF_PORT(in_test2) `DEF_END

// In/Out
// `DEF_DOT `DEF_INOUT([7:0]) io_bus1 `DEF_PORT(io_bus1) `DEF_END

// Output
`DEF_DOT `DEF_OUT([7:0]) out_test2 `DEF_PORT(out_test2)
// No DEF_END here...

`undef DEF_IN
`undef DEF_INOUT
`undef DEF_OUT
`undef DEF_END
`undef DEF_DOT
`undef DEF_PORT
```


Copyright © 2013 Synopsys, Inc.

Verilog File top.v

```
// top.v

`define INC_TYPE 1
module ppm_top(
 `ifdef INC_TYPE
// Inc file Port def...
   `define DEF_IN(arg1) /* arg1 */
   `define DEF_INOUT(arg1) /* arg1 */
   `define DEF_OUT(arg1) /* arg1 */
   `define DEF_END ,
   `define DEF_DOT /* nothing */
   `define DEF_PORT(arg1) /* arg1 */

`include "ppm_top_ports_def.inc"
   `else
   // Non-Inc file Port def, above defines should expand to
   // what is below...
      /* nothing */ /* [7:0] */ in_test1 /* in_test1 */ ,
      /* nothing */ /* [7:0] */ in_test2 /* in_test2 */ ,

   // In/Out
   //`DEF_DOT `DEF_INOUT([7:0]) io_bus1 `DEF_PORT(io_bus1)
`DEF_END

   // Output
      /* nothing */ /* [7:0] */ out_test2 /* out_test2 */
// No DEF_END here...
 `endif
);

   `ifdef INC_TYPE
   // Inc file Signal type def...
   `define DEF_IN(arg1) input arg1
   `define DEF_INOUT(arg1) inout arg1
   `define DEF_OUT(arg1) output arg1
   `define DEF_END ;
   `define DEF_DOT /* nothing */
   `define DEF_PORT(arg1) /* arg1 */
```

```
`include "ppm_top_ports_def.inc"
   `else
   // Non-Inc file Signal type def, defines should expand to
   // what is below...
      /* nothing */ input [7:0] in_test1 /* in_test1 */ ;
      /* nothing */ input [7:0] in_test2 /* in_test2 */ ;

// In/Out
   //`DEF_DOT `DEF_INOUT([7:0]) io_bus1 `DEF_PORT(io_bus1)`DEF_END

// Output
   /* nothing */ output [7:0] out_test2 /* out_test2) */
// No DEF_END here...
   `endif

 ; /* Because of the 'No DEF_END here...' in line of the include
file. */

   assign out_test2 = (in_test1 & in_test2);

endmodule
```


Copyright © 2013 Synopsys, Inc.

Verilog Attribute and Directive Syntax

Verilog attributes and directives allow you to associate information with your design to control the way it is analyzed, compiled, and mapped.

- *Attributes* direct the way your design is optimized and mapped during synthesis.
- *Directives* control the way your design is analyzed prior to mapping. They must therefore be included directly in your source code; they cannot be specified in a constraint file like attributes.

Verilog does not have predefined attributes or directives for synthesis. To define directives or attributes in Verilog, attach them to the appropriate objects in the source code as comments. You can use either of the following comment styles:

- Regular line comments
- Block or C-style comments

Each specification begins with the keyword synthesis. The directive or attribute value is either a string, placed within double quotes, or a Boolean integer (0 or 1). Directives, attributes, and their values are-case sensitive and are usually in lower case.

Attribute Syntax and Examples using Verilog Line Comments

Here is the syntax using a regular Verilog comment:

// **synthesis** *directive* | *attribute* [**= "***value***"**]

This example shows how to use the syn_hier attribute:

```
// synthesis syn_hier = "firm"
```

This example shows the parallel_case directive:

```
// synthesis parallel_case
```

This directive forces a multiplexed structure in Verilog designs. It is implicitly true whenever you use it, which is why there is no associated value.

Attribute Syntax and Examples Using Verilog C-Style Comments

Here is the syntax for specifying attributes and directives with the C-style block comment:

*/** **synthesis** *directive* | *attribute* [**= "***value***"**] **/*

This example shows the syn_hier attribute specified with a C-style comment:

```
/* synthesis syn_hier = "firm" */
```

The following are some other rules for using C-style comments to define attributes:

- If you use C-style comments, you must place the comment *after* the *object* declaration and *before* the semicolon of the statement. For example:

  ```
  module bl_box(out, in) /* synthesis syn_black_box */ ;
  ```

- To specify more than one directive or attribute for a given design object, place them within the same comment, separated by a space. Do *not* use commas as separators. Here is an example where the syn_preserve and syn_state_machine directives are specified in a single comment:

  ```
  module radhard_dffrs(q,d,c,s,r)
     /* synthesis syn_preserve=1 syn_state_machine=0 */;
  ```

- To make source code more readable, you can split long block comment lines by inserting a backslash character (\) followed immediately by a newline character (carriage return). A line split this way is still read as a single line; the backslash causes the newline following it to be ignored. You can split a comment line this way any number of times. However, note these exceptions:

  – The first split cannot occur before the first attribute or directive specification.

  – A given attribute or directive specification cannot be split before its equal sign (=).

  Take this block comment specification for example:

```
/* synthesis syn_probe=1 xc_loc="P20,P21,P22,P23,P24,P25,P26,P27" */;
```

  You cannot split the line before you specify the first attribute, syn_probe. You cannot split the line before either of the equal signs (syn_probe= or

Copyright © 2013 Synopsys, Inc.

`xc_loc=`). You can split it anywhere within the string value `"P20,P21,P22,P23,P24,P25,P26,P27"`.

Attribute Examples Using Verilog 2001 Parenthetical Comments

Here is the syntax for specifying attributes and directives as Verilog 2001 parenthetical comments:

(* *directive*|*attribute* [**= "***value***"**] ***)**

Verilog 2001 parenthetical comments can be applied to:

- individual objects
- multiple objects
- individual objects within a module definition

The following example shows two syn_keep attributes specified as parenthetical comments:

```
module example1(out1, out2, clk, in1, in2);
output out1, out2;
input clk;
input in1, in2;
wire and_out;
(* syn_keep=1 *) wire keep1;
(* syn_keep=1 *) wire keep2;
reg out1, out2;
assign and_out=in1&in2;
assign keep1=and_out;
assign keep2=and_out;

always @(posedge clk)begin;
   out1<=keep1;
   out2<=keep2;
end
endmodule
```

For the above example, a single parenthetical comment could be added directly to the reg statement to apply the syn_keep attribute to both out1 and out2:

```
(* syn_keep=1 *) reg out1, out2;
```

The following rules apply when using parenthetical comments to define attributes:

- Always place the comment *before* the design object (and terminating semicolon). For example:

```
(* syn_black_box *) module bl_box(out, in);
```

- To specify more than one directive or attribute for a given object, place the attributes within the same parenthetical comment, separated by a space (do *not* use commas as separators). The following example shows the syn_preserve and syn_state_machine directives applied in a single parenthetical comment:

```
(* syn_preserve=1 syn_state_machine=0 *)
   module radhard_dffrs(q,d,c,s,r);
```

- Parenthetical comments can be applied to individual objects within a module definition. For example,

```
module example2 (out1, (*syn_preserve=1*) out2, clk, in1, in2);
```

applies a syn_preserve attribute to out2, and

```
module example2 ( (*syn_preserve=1*) out1,
   (*syn_preserve=1*) out2, clk, in1, in2);
```

applies a syn_preserve attribute to both out1 and out2

Copyright © 2013 Synopsys, Inc.

CHAPTER 10

SystemVerilog Language Support

This chapter describes support for the SystemVerilog standard in the Synopsys FPGA synthesis tools. For information on the Verilog standard, see Chapter 9, *Verilog Language Support*. The following describe SystemVerilog-specific support:

- Feature Summary, on page 764
- Unsized Literals, on page 769
- Data Types, on page 769
- Arrays, on page 778
- Data Declarations, on page 781
- Operators and Expressions, on page 786
- Procedural Statements and Control Flow, on page 797
- Processes, on page 800
- Tasks and Functions, on page 805
- Hierarchy, on page 809
- Interface, on page 816
- System Tasks and System Functions, on page 825
- Generate Statement, on page 827
- Assertions, on page 832
- Keyword Support, on page 836

Copyright © 2013 Synopsys, Inc.

Feature Summary

SystemVerilog is a IEEE (P1800) standard with extensions to the IEEE Std.1364-2001 Verilog standard. The extensions integrate features from C, C++, VHDL, OVA, and PSL. The following table summarizes the SystemVerilog features currently supported in the Synopsys FPGA Verilog compilers. See *SystemVerilog Limitations*, on page 767 for a list of limitations.

Feature	Brief Description
Unsized Literals	Specification of unsized literals as single-bit values without a base specifier.
Data Types • Typedefs • Enumerated Types • Struct Construct • Union Construct • Static Casting	Data types that are a hybrid of both Verilog and C including: • User-defined types that allow you to create new type definitions from existing types • Variables and nets defined with a specific set of named values • Structure data type to represent collections of variables referenced as a single name • Data type collections sharing the same memory location • Conversion of one data type to another data type.
Arrays • Arrays • Arrays of Structures	Packed, unpacked, and multi-dimensional arrays of structures.
Data Declarations • Constants • Variables • Nets • Data Types in Parameters • Type Parameters	Data declarations including constant, variable, net, and parameter data types.

Copyright © 2013 Synopsys, Inc.

Feature	Brief Description
Operators and Expressions • Operators • Aggregate Expressions • Streaming Operator • Set Membership Operator • Set Membership Case Inside Operator • Type Operator	C assignment operators and special bit-wise assignment operators.
Procedural Statements and Control Flow • Do-While Loops • For Loops • Unnamed Blocks • Block Name on end Keyword • Unique and Priority Modifiers	Procedural statements including variable declarations and block functions.
Processes • always_comb • always_latch • always_ff	Specialized procedural blocks that reduce ambiguity and indicate the intent.
Tasks and Functions • Implicit Statement Group • Formal Arguments • endtask/endfunction Names	Information on implicit grouping for multiple statements, passing formal arguments, and naming end statements for functions and tasks.
Hierarchy • Compilation Units • Packages • Port Connection Constructs • Extern Module	Permits sharing of language-defined data types, user-defined types, parameters, constants, function definitions, and task definitions among one or more compilation units, modules, or interfaces (pkgs)
Interface • Interface Construct • Modports	Interface data type to represent port lists and port connection lists as single name.
System Tasks and System Functions • $bits System Function • Array Querying Functions	Queries to returns number of bits required to hold an expression as a bit stream or array.

Copyright © 2013 Synopsys, Inc.

Feature	Brief Description
Generate Statement: Conditional Generate Constructs	Generate-loop, generate-conditional, or generate-case statements with defparams, parameters, and function and task declarations. Conditional if-generate and case-generate constructs
Assertions: SVA System Functions	SystemVerilog assertion support.
Keyword Support	Supported and unsupported keywords.

Copyright © 2013 Synopsys, Inc.

SystemVerilog Limitations

The following SystemVerilog limitations are present in the current release.

Interface

- An array of interfaces cannot be used as a module port.
- An interface cannot have a multi-dimensional port.
- Access of array type elements outside of the interface are not supported. For example:

```
interface ff_if (input logic din, input [7:0] DHAin1,
   input [7:0] DHAin2, output logic dout);
logic [1:0] [1:0] [1:0] DHAout_intf;

always_comb
DHAout_intf = DHAin1 + DHAin2;

modport write (input din, output dout);
endinterface: ff_if

ff_if ff_if_top(.*);
DHAout = ff_if_top.DHAout_intf;
```

- Modport definitions within a Generate block are not supported. For example:

```
interface myintf_if (input logic [7:0] a , input logic [7:0]  b,
   output logic [7:0] out1, output logic [7:0] out2);
generate
   begin: x
   genvar i;
      for (i = 0;i <= 7;i=i+1)
      begin : u
         modport myinst(input .ma(a[i]), input .mb(b[i]),
            output .mout1(out1[i]) , output .mout2(out2[i]));
      end
   end
endgenerate
endinterface
```


Copyright © 2013 Synopsys, Inc.

Compilation Unit and Package

- Write access to the variable defined in package/compilation unit is not supported. For example:

```
package MyPack;
typedef struct packed {
   int r;
   longint g;
   byte b;
}  MyStruct ;

MyStruct StructMyStruct;
endpackage: MyPack

import MyPack::*;
module top ( ...
...

always@(posedge clk)
StructMyStruct <= '{default:254};
```

Unsized Literals

SystemVerilog allows you to specify unsized literals without a base specifier (auto-fill literals) as single-bit values with a preceding apostrophe ('). All bits of the unsized value are set to the value of the specified bit.

```
'0, '1, 'X, 'x, 'Z, 'z // sets all bits to this value
```

In other words, this feature allows you to fill a register, wire, or any other data types with 0, 1, X, or Z in simple format.

Verilog Example	SystemVerilog equivalent
a = 4'b1111;	a = '1;

Data Types

SystemVerilog makes a clear distinction between an *object* and its *data type*. A data type is a set of values, or a set of operations that can be performed on those values. Data types can be used to declare data objects.

SystemVerilog offers the following data types, which represent a hybrid of both Verilog and C:

Data Type	Description
shortint	2-state, SystemVerilog data type, 16-bit signed integer
int	2-state, SystemVerilog data type, 32-bit signed integer
longint	2-state, SystemVerilog data type, 64-bit signed integer
byte	2-state, SystemVerilog data type, 8-bit signed integer or ASCII character
bit	2-state, SystemVerilog data type, user-defined vector size
logic	4-state, SystemVerilog data type, user-defined vector size

Copyright © 2013 Synopsys, Inc.

Data types are characterized as either of the following:

- 4-state (4-valued) data types that can hold 1, 0, X, and Z values
- 2-state (2-valued) data types that can hold 1 and 0 values

The following apply when using data types:

- The data types byte, shortint, int, integer and longint default to signed; data types bit, reg, and logic default to unsigned, as do arrays of these types.
- The signed keyword is part of Verilog. The unsigned keyword can be used to change the default behavior of signed data types.
- The Verilog compiler does not generate an error even if a 2-state data type is assigned X or Z. It treats it as a “don't care” and issues a warning.
- Do not use the syn_keep directive on nets with SystemVerilog data types. When you use data types such as bit, logic, longint, or shortint, the synthesis software might not be aware of the bit sizes on the LHS and RHS for the net. For example:

  ```
  bit x;
     shortint y;
     assign y =x;
  ```

  In this case, bit defaults to a 1-bit width and includes a shortint of 16-bit width. If syn_keep is applied on y, the software does not use the other 15 bits.

Typedefs

You can create your own names for type definitions that you use frequently in your code. SystemVerilog adds the ability to define new net and variable user-defined names for existing types using the typedef keyword.

Example – Simple typedef Variable Assignment

Example – Using Multiple typedef Assignments

Enumerated Types

The synthesis tools support SystemVerilog enumerated types in accordance with SV LRM section: 4.10; enumerated methods are not supported.

The enumerated types feature allows variables and nets to be defined with a specific set of named values. This capability is particularly useful in state-machine implementation where the states of the state machine can be verbally represented

Data Types

Enumerated types have a base data type which, by default, is int (a 2-state, 32-bit value). By default, the first label in the enumerated list has a logic value of 0, and each subsequent label is incremented by one.

For example, a variable that has three legal states:

```
enum {WAITE, LOAD, READY} State ;
```

The first label in the enumerated list has a logic value of 0 and each subsequent label is incremented by one. In the example above, State is an int type and WAITE, LOAD And READY have 32-bit int values. WAITE is 0, LOAD is 1, and READY is 2.

You can specify an explicit base type to allow enumerated types to more specifically model hardware. For example, two enumerated variables with one-hot values:

```
enum logic [2:0] {WAITE=3'b001, LOAD=3'b010,READY=3'b100} State;
```

Specifying Ranges

SystemVerilog enumerated types also allow you to specify ranges that are automatically elaborated. Types can be specified as outlined in the following table.

Copyright © 2013 Synopsys, Inc.

Syntax	Description
name	Associates the next consecutive number with the specified name.
name = C	Associates the constant C to the specified name.
name[*N*]	Generates N named constants in this sequence: *name0, name1,..., nameN-1*. N must be a positive integral number.
name[*N*] = C	Optionally assigns a constant to the generated named constants to associate that constant with the first generated named constant. Subsequent generated named constants are associated with consecutive values. N must be a positive integral number.
name[*N:M*]	Creates a sequence of named constants, starting with *nameN* and incrementing or decrementing until it reaches named constant *nameM*. N and M are non-negative integral numbers.
name[*N:M*] = C	Optionally assigns a constant to the generated named constants to associate that constant with the first generated named constants. Subsequent generated named constants are associated consecutive values. N and M must be positive integral numbers.

The following example declares enumerated variable vr, which creates the enumerated named constants register0 and register1, which are assigned the values 1 and 2, respectively. Next, it creates the enumerated named constants register2, register3, and register4 and assigns them the values 10, 11, and 12.

```
enum { register[2] = 1, register[2:4] = 10 } vr;
```

State-Machine Example

The following is an example state-machine design in SystemVerilog.

Example – State-machine Design

Type Casting Using Enumerated Types

By using enumerated types, you can define a type. For example:

```
typedef enum { red,green,blue,yellow,white,black } Colors;
```

The above definition assigns a unique number to each of the color identifiers and creates the new data type Colors. This new type can then be used to create variables of that type.

Valid assignment would be:

```
Colors c;

C = green;
```

Enumerated Types in Expressions

Elements of enumerated types can be used in numerical expressions. The value used in the expression is the value specified with the numerical value. For example:

```
typedef enum { red,green,blue,yellow,white,black } Colors;
integer a,b;
a = blue *3 // 6 is assigned to a
b = yellow + green; // 4 is assigned to b
```

Struct Construct

SystemVerilog adds several enhancements to Verilog for representing large amounts of data. In SystemVerilog, the Verilog array constructs are extended both in how data can be represented and for operations on arrays. A structure data type has been defined as a means to represent collections of data types. These data types can be either standard data types (such as int, logic, or bit) or, they can be user-defined types (using SystemVerilog typedef). Structures allow multiple signals, of various data types, to be bundled together and referenced by a single name.

Structures are defined under section 4.11 of IEEE Std 1800-2005 (IEEE Standard for SystemVerilog).

In the example structure floating_pt_num below, both characteristic and mantissa are 32-bit values of type bit.

```
struct {
   bit [31:0] characteristic;
   bit [31:0] mantissa;
}  floating_pt_num;
```

Alternately, the structure could be written as:

Copyright © 2013 Synopsys, Inc.

```
typedef struct {
   bit [31:0] characteristic;
   bit [31:0] mantissa;
}  flpt;
flpt floating_pt_num;
```

In the above sequence, a type flpt is defined using typedef which is then used to declare the variable floating_pt_num.

Assigning a value to one or more fields of a structure is straight-forward.

```
floating_pt_num.characteristic = 32'h1234_5678;

floating_pt_num.mantissa       = 32'h0000_0010;
```

As mentioned, a structure can be defined with fields that are themselves other structures.

```
typedef struct {
   flpt x;
   flpt y;
}  coordinate;
```

Packed Struct

Various other unique features of SystemVerilog data types can also be applied to structures. By default, the members of a structure are *unpacked*, which allows the Synopsys FPGA tools to store structure members as independent objects. It is also possible to *pack* a structure in memory without gaps between its bit fields. This capability can be useful for fast access of data during simulation and possibly result in a smaller footprint of your simulation binary.

To pack a structure in memory, use the packed keyword in the definition of the structure:

```
typedef struct packed {
   bit [31:0] characteristic;
   bit [31:0] mantissa;
} flpt;
```

An advantage of using packed structures is that one or more bits from such a structure can be selected as if the structure was a packed array. For instance, flpt[47:32] in the above declaration is the same as characteristic[15:0].

Copyright © 2013 Synopsys, Inc.

Struct members are selected using the .name syntax as shown in the following two code segments.

```
// segment 1
typedef struct {
   bit [7:0] opcode;
   bit [23:0] addr;
} instruction; // named structure type
instruction IR; // define variable
IR.opcode = 1; //set field in IR.
```

```
// segment 2
struct {
   int x,y;
} p;
p.x = 1;
```

Union Construct

A union is a collection of different data types similar to structure with the exception that members of the union share the same memory location. At any given time, you can write to any one member of the union which can then be read by the same member or a different member of that union.

Union is broadly classified as:

- Packed Union
- Unpacked Union

Currently, only packed unions are supported.

Packed Union

A packed union can only have members that are of the packed type (packed structure, packed array of logic, bit, int, etc.). All members of a packed union must be of equal size.

Syntax

Copyright © 2013 Synopsys, Inc.

```
Union packed
{
    member1;
    member2;
} unionName;
```

Unpacked Union

The members of an unpacked union can include both packed and unpacked types (packed/unpacked structures, arrays of packed/unpacked logic, bit, int, etc.) with no restrictions as to the size of the union members.

Syntax

```
Union
{
    member1;
    member2;
} unionName;
```

Example 1 – Basic Packed Union (logical operation)

Example 2 – Basic Packed Union (arithmetic operation)

Example 3 – Nested Packed Union

Example 4 – Array of packed Union

Limitations

The SystemVerilog compiler does not support the following union constructs:

- unpacked union
- tagged packed union
- tagged unpacked union

Currently, support is limited to packed unions, arrays of packed unions, and nested packed unions.

Copyright © 2013 Synopsys, Inc.

Static Casting

Static casting allows one data type to be converted to another data type. The static casting operator is used to change the data type, the size, or the sign:

- Type casting – a predefined data type is used as a *castingType* to change the data type.
- Size casting – a positive decimal number is used as a *castingType* to change the number of data bits.
- Sign casting – signed/unsigned are used to change the sign of data type.
- Bit-stream casting – type casting that is applied to unpacked arrays and structs. During bit-stream casting, both the left and right sides of the equation must be the same size. Arithmetic operations cannot be combined with static casting operations as is in the case of singular data types.

Syntax

castingType **'(***castingExpression***)**

Example – Type Casting of Singular Data Types

Example – Type Casting of Aggregate Data Types

Example – Bit-stream Casting

Example – Size Casting

Example – Sign Casting

Copyright © 2013 Synopsys, Inc.

Arrays

Topics in this section include:

- Arrays, on page 778
- Arrays of Structures, on page 780

Arrays

SystemVerilog uses the term *packed array* to refer to the dimensions declared before the object name (same as Verilog *vector width*). The term *unpacked array* refers to the dimensions declared after the object name (same as Verilog *array dimensions*). For example:

```
reg [7:0] foo1; //packed array
reg foo2 [7:0]; //unpacked array
```

A packed array is guaranteed to be represented as a contiguous set of bits and, therefore, can be conveniently accessed as array elements. While unpacked is not guaranteed to work so, but in terms of hardware, both would be treated or bit-blasted into a single dimension.

```
module test1 (input [3:0] data, output [3:0] dout);
   //example on packed array four-bit wide.

assign dout = data;
endmodule

module test2 (input data [3:0], output dout [3:0]);
//unpacked array of 1 bit by 4 depth;

assign dout = data;
endmodule
```

Multi-dimensional packed arrays unify and extend Verilog's notion of *registers* and *memories*:

```
reg [1:0][2:0] my_var[32];
```

Classical Verilog permitted only one dimension to be declared to the left of the variable name. SystemVerilog permits any number of such *packed* dimensions. A variable of packed array type maps 1:1 onto an integer arithmetic quantity. In the example above, each element of my_var can be used in expres-

sions as a six-bit integer. The dimensions to the right of the name (32 in this case) are referred to as *unpacked* dimensions. As in Verilog-2001, any number of unpacked dimensions is permitted.

The general rule for multi-dimensional packed array is as follows:

```
reg/wire [matrix_n:0] … [matrix_1:0][depth:0][width:0] temp;
```

The general rule for multi-dimensional unpacked array is as follows:

```
reg/wire temp1 [matrix_n:0]… [matrix_1:0][depth:0]; //single bit wide
reg/wire [width_m:0] temp2 [matrix_n:0]… [matrix_1:0][depth:0];
   // width_m bit wide
```

The general rule for multi-dimensional array, mix of packed/unpacked, is as follows:

```
reg/wire [width_m:0] temp3 [matrix:0]… [depth:0];

reg/wire [depth:0][width:0] temp4 [matrix_m:0]… [matrix_1:0]
```

For example, in a multi-dimensional declaration, the dimensions declared following the type and before the name vary more rapidly than the dimensions following the name.

Multi-dimensional arrays can be used as ports of the module.

The following items are now supported for multi-dimensional arrays:

- Assignment of a whole multi-dimensional array to another.
- Access (reading) of an entire multi-dimensional array.
- Assignment of an index (representing a complete dimension) of a multi-dimensional array to another.
- Access (reading) of an index of a multi-dimensional array.
- Assignment of a slice of a multi-dimensional array.
- Access of a slice of a multi-dimensional array.
- Access of a variable part-select of a multi-dimensional array.

In addition, wire declarations are supported for any packed or unpacked data type. This support includes multi-dimensional enum and struct data types in input port declarations (see *Nets*, on page 783 for more information).

Copyright © 2013 Synopsys, Inc.

Packed arrays are supported with the access/store mechanisms listed above. Packed arrays can also be used as ports and arguments to functions and tasks. The standard multi-dimensional access of packed arrays is supported.

Unpacked array support is the same as packed array supported stated in items one through seven above.

Example – Multi-dimensional Packed Array with Whole Assignment

Example – Multi-dimensional Packed Array with Partial Assignment

Example – Multi-dimensional Packed Array with Arithmetic Ops

Example – Packed/Unpacked Array with Partial Assignment

Arrays of Structures

SystemVerilog supports multi-dimensional arrays of structures which can be used in many applications to manipulate complex data structures. A multi-dimensional array of structure is a structured array of more than one dimension. The structure can be either packed or unpacked and the array of this structure can be either packed or unpacked or a combination of packed and unpacked. As a result, there are many combinations that define a multi-dimensional array of structure.

A multi-dimensional array of structure can be declared as either anonymous type (inline) or by using a typedef (user-defined data type).

Some applications where multi-dimensional arrays of structures can be used are where multi-channeled interfaces are required such as packet processing, dot-product of floating point numbers, or image processing.

Example – Multi-dimensional Array of Packed and Unpacked Structures Using typedef

Example – Multi-dimensional Array of UnPacked Structures Using typedef

Example – Multi-dimensional Array of Packed Structures Using Anonymous Type

Example – Multi-dimensional Array of Packed Structures Using typedef

Array Querying Functions

SystemVerilog provides system functions that return information about a particular dimension of an array. For information on this function, see *Array Querying Functions*, on page 826.

Data Declarations

There are several data declarations in SystemVerilog: *literals*, *parameters*, *constants*, *variables*, *nets*, and *attributes*. The following are described here:

- Constants, on page 782
- Variables, on page 782
- Nets, on page 783
- Data Types in Parameters, on page 784
- Type Parameters, on page 784

Copyright © 2013 Synopsys, Inc.

Constants

Constants are named data variables, which never change. A typical example for declaring a constant is as follows:

```
const a = 10;

const logic  [3:0] load = 4'b1111;

const reg  [7:0] load1 = 8'h0f, dataone = '1;
```

The Verilog compiler generates an error if constant is assigned a value.

```
const shortint a = 10;
assign a = '1;     // This is illegal
```

Variables

Variables can be declared two ways:

Method 1	Method 2

`shortint a, b;`
`logic [1:0] c, d;`
	`var logic [15:0] a;`
	var a,b; // equivalent var logic a, b
	var [1:0] c, d; // equivalent var logic [1:0] c, d
	input var shortint datain1,datain2;
	output var logic [15:0] dataout1,dataout2;

Method 2 uses the keyword var to preface the variable. In this type of declaration, a data type is optional. If the data type is not specified, logic is inferred.

Typical module declaration:

```
module test01 (input var shortint datain1,datain2,
   output var logic [15:0] dataout1,dataout2 );
```

A variable can be initialized as follows:

```
var  a = 1'b1;
```

Nets

Nets are typically declared using the wire keyword. Any 4-state data type can be used to declare a net. When using wire with struct and union constructs, each member of the construct must be a 4-state data type.

Syntax

wire *4stateDataType identifierName* **;**

Example – Logic Type Defined as a Wire Type

```
module top (
   input wire logic [1:0] din1,din2 , // logic defined as wire
   output logic [1:0] dout  );
   assign dout = din1 + din2;
endmodule
```

Example – struct Defined as a Wire Type

```
typedef struct { logic [4:1] a;
}  MyStruct;

module top (
   input wire MyStruct [1:0] din [1:0] [1:0], // structure
      // defined as wire
   output wire MyStruct [1:0] dout [1:0] [1:0] ); // structure
      // defined as wire
assign dout = din;
endmodule
```

Restrictions

Using wire with a 2-state data type (for example, int or bit) results in the following error message:

CG1205 | Net data types must be 4-state values

A lexical restriction also applies to a net or port declaration in that the net type keyword wire cannot be followed by reg.

Copyright © 2013 Synopsys, Inc.

Data Types in Parameters

In SystemVerilog with different data types being introduced, the *parameter* can be of any data type (i.e., language-defined data type, user-defined data type, and packed/unpacked arrays and structures). By default, parameter is the int data type.

Syntax

parameter *dataType varaibleName* **=** *value*

In the above syntax, *dataType* is a language-defined data type, user-defined data type, or a packed/unpacked structure or array.

Example – Parameter is of Type longint

Example – Parameter is of Type enum

Example – Parameter is of Type structure

Example – Parameter is of Type longint Unpacked Array

Type Parameters

SystemVerilog includes the ability for a parameter to also specify a data type. This capability allows modules or instances to have data whose type is set for each instance – these *type* parameters can have different values for each of their instances.

Note: Overriding a type parameter with a defparam statement is illegal.

Syntax

parameter type *typeIdentifierName* **=** *dataType*;

localparam type *typeIdentifierName* **=** *dataType*;

In the above syntax, *dataType* is either a language-defined data type or a user-defined data type.

Example – Type Parameter of Language-Defined Data Type

Example – Type Parameter of User-Defined Data Type

Example – Type Local Parameter

Copyright © 2013 Synopsys, Inc.

Operators and Expressions

Topics in this section include:

- Operators, on page 786
- Aggregate Expressions, on page 787
- Streaming Operator, on page 789
- Set Membership Operator, on page 790
- Set Membership Case Inside Operator, on page 791
- Type Operator, on page 794

Operators

SystemVerilog includes the C assignment operators and special bit-wise assignment operators:

+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, <<<=, >>>=

An assignment operator is semantically equivalent to a blocking assignment with the exception that the expression is only evaluated once.

Operator Example	Same as
`A += 2;`	`A = A + 2;`
`B -= B;`	`B = B - A;`
`C *= B;`	`C = C * B;`
`D /= C;`	`D = D / C;`
`E %= D;`	`E = E % D;`
`F &= E;`	`F = F & E;`
`G |= F;`	`G = G | F;`
`H ^= G;`	`H = H ^ G;`
`I <<= H;`	`I = I << H;`

Copyright © 2013 Synopsys, Inc.

Operator Example	Same as
`J >>= I;`	`J = J  >> I;`
`K <<<=J;`	`K = K  <<< J;`
`L >>>=K;`	`L = L  >>> K;`

In addition, SystemVerilog also has the increment/decrement operators *i*++, *i*--, ++*i*, and --*i*.

Operator Example	Same as
`A++;`	A = A + 1;
`A--;`	A = A - 1;
`++A;`	Increment first and then use A
`--A;`	Decrement first and then use A

In the following code segment, out1 gets r1 and out2 gets the twice-decremented value of out1:

```
always @(*)
   begin
      out1 = r1--;
      out2 = --r1;
   end
```

Aggregate Expressions

Aggregate expressions (aggregate pattern assignments) are primarily used to initialize and assign default values to unpacked arrays and structures.

Syntax

SystemVerilog aggregate expressions are constructed from braces; an apostrophe prefixes the opening (left) brace.

'{ *listofValues* **}**

Copyright © 2013 Synopsys, Inc.

In the syntax, *listofValues* is a comma-separated list. SystemVerilog also provides a mechanism to initialize all of the elements of an unpacked array by specifying a default value within the braces using the following syntax:

'{ default: *value* **}**
'{ data type:*value* **}**
'{ index:*value* **}**

The aggregate (pattern) assignment can be used to initialize any of the following.

- a 2-dimensional unpacked array under a reset condition (see Initializing Unpacked Array Under Reset Condition example).
- all the elements of a 2-dimensional unpacked array to a default value using the default keyword under a reset condition (see Initializing Unpacked Array to Default Value example).
- a specific data type using the keyword for *type* instead of default (see Initializing Specific Data Type example).

Aggregate (pattern) assignment can also be specified in a package (see Aggregate Assignment in Package example) and in a compilation unit (see Aggregate Assignment in Compilation Unit example).

Example – Initializing Unpacked Array Under Reset Condition

Example – Initializing Unpacked Array to Default Value

Example – Initializing Specific Data Type

Example – Aggregate Assignment in Package

Example – Aggregate Assignment in Compilation Unit

Copyright © 2013 Synopsys, Inc.

Streaming Operator

The streaming operator (>> or <<) packs the bit-stream type to a particular sequence of bits in a user-specified order. Bit-stream types can be any integral, packed or unpacked type or structure. The streaming operator can be used on either the left or right side of the expression.

The streaming operator determines the order of bits in the output data stream:

- The left-to-right operator (>>) arranges the output data bits in the same order as the input bit stream
- The right-to-left operator (<<) arranges the output data bits in reverse order from the input bit stream

Syntax

streamingExpression **::= {** *streamOperator* [*sliceSize*] *streamConcatenation* **}**

streamOperator **::= >> | <<**

sliceSize **::=** *dataType* | *constantExpression*

streamConcatenation **::= {***streamExpression* {**,** *streamExpression*} **}**

streamExpression **::=** *arrayRangeExpression*

When an optional *sliceSize* value is included, the stream is broken up into the slice-size segments prior to performing the specified streaming operation. By default, the *sliceSize* value is 1.

Usage

The streaming operator is used to:

- Reverse the entire data stream
- Bit-stream from one data type to other

When the slice size is larger than the data stream, the stream is left-justified and zero-filled on the right. If the data stream is larger than the left side variable, an error is reported.

Example – Packed type inputs/outputs with RHS operator

Copyright © 2013 Synopsys, Inc.

Example – Unpacked type inputs/outputs with RHS operator

Example – Packed type inputs/outputs with LHS operator

Example – Slice-size streaming with RHS operator

Example – Slice-size streaming with LHS slice operation

Set Membership Operator

The set membership operator, also referred to as the *inside* operator, returns the value TRUE when the expression value (i.e., the LHS of the operator) is present in the value list of the RHS operator. If the expression value is not present in the RHS operator, returns FALSE.

Syntax

(*expressionValue***) inside {***listofValues***}**

expressionValue **::=** *singularExpression*

listofValues **::=** *rangeofValues*, *expressions*, *arrayofAggregateTypes*

Example – Inside operator with dynamically changing input at LHS operator

Example – Inside operator with expression at LHS operator

Example – Inside operator with dynamically changing input at LHS and RHS operators

Example – Inside operator with array of parameter at LHS operator

Copyright © 2013 Synopsys, Inc.

Set Membership Case Inside Operator

With the case inside operator, a case expression is compared to each case item. Also, when using this operator, the case items can include an open range. The comparison returns TRUE when the case expression matches a case item, otherwise it returns FALSE.

Syntax

[unique|priority] case (*caseExpression*) inside
(*caseItem*) : *statement* ;
(*caseItem*) : *statement* ;
.
.
.
[default : *statement* ;]
endcase

In the above syntax, *caseItem* can be:

- a list of constants
- an open range
- a combination of a list of constants and an open range

The case inside operator supports the following optional modifiers:

- unique – each *caseItem* is unique and there are no overlapping *caseItems*. If there is an overlapping *caseItem*, a warning is issued.
- priority – the case statement is prioritized and all possible legal cases are covered by the case statement. If the *caseExpression* fails to match any of the *caseItems*, a warning is issued.

Example – Case Inside

```
module top# (
   parameter byte p1[2:1][4:1] = '{'{0,2,4,6},'{1,3,5,7}} )
//Input
(  input logic[4:1]sel,a,b,
//Output
   output logic[3:1] q );
```


Copyright © 2013 Synopsys, Inc.

```
always_comb begin
   case (sel) inside
      8,p1[1],10,12,14:q <= a;
      p1[2],9,11,13,15:q <= b;
   endcase
end
endmodule
```

Example – Unique Case Inside

```
module top# (
   parameter byte p1[2:1][4:1] = '{'{15,14,13,12},'{0,1,2,3}} )
//Input
(  input logic[4:1]sel1,sel2,
   input byte a,b,
//Output
   output byte q );

generate begin
   always@(*) begin
      unique case (sel1^sel2) inside
         p1 : q = a+b;
         [4:7],13,14,15 : q = a ^ b;
         [9:12],8 : q = a*b;
      endcase
   end
end
endgenerate
endmodule
```

Example – Priority Case Inside

```
typedef enum logic[4:1] {s[0:15]} EnumDt;

module top (
   input logic reset,
   input logic clock,
   input logic x,
   input logic[2:1] y,
   output logic[3:1] op );
EnumDt state;
```

```
always@(posedge reset or posedge clock)
begin
   if (reset == 1'b1)
   begin
      op <= 3'b000;
      state <= s0;
   end
   else
   begin
      priority case (state) inside
         [s0:s2],s12 : begin
            if (x == 1'b0  && y == 1'b0)
            begin
               state <= s3;
               op <= 3'b001;
            end
            else
            begin
               state <= s2;
               op <= 3'b000;
            end
         end
         [s3:s5] : begin
            if(x == 1'b1  && y== 1'b0)
            begin
               state <= s7;
               op <= 3'b010;
            end
            else
            begin
               state <= s9;
               op <= 3'b110;
            end
         end
         [s6:s8],s13 : begin
            if(x == 1'b0  &&  y== 1'b1)
            begin
               state <= s11;
               op <= 3'b011;
            end
            else if (x == 1'b0 && y == 1'b1)
            begin
               state <= s4;
               op <= 3'b010;
            end
         end
```


Copyright © 2013 Synopsys, Inc.

```
            [s9:s11] : begin
               if(x == 1'b1  && y== 1'b1 )
               begin
                  state <= s5;
                  op <= 3'b100;
               end
               else if (x == 1'b0 && y == 1'b1)
               begin
                  state <= s0;
                  op <= 3'b111;
               end
            end
            default : begin
               state <= s1;
               op <= 3'b111;
            end
         endcase
      end
   end
   endmodule
```

Type Operator

SystemVerilog provides a type operator as a way of referencing the data type of a variable or an expression.

Syntax

type(*dataType* | *expression***)**

dataType – a user-defined data type or language-defined data type

expression – any expression, variable, or port

An *expression* inside the type operator results in a self-determined type of expression; the expression is not evaluated. Also the *expression* cannot contain any hierarchical references.

Data Declaration

The type operator can be used while declaring signals, variables, or ports of a module/interface or a member of that interface.

Example – Using Type Operator to Declare Input/Output Ports

```
typedef logic signed[4:1]logicdt;
// Module top
module top(
   input type(logicdt) d1,
   output type(logicdt) dout1 );
type(logicdt) sig;
var type(logicdt) sig1;
assign sig  = d1;
assign sig1= d1+1'b1;
assign dout1= sig + sig1;
endmodule
```

Data Type Declaration

Defining of the user-defined data type can have the type operator, wherein a variable or another user-defined data type can be directly referenced while defining a data type using the type operator. The data type can be defined in the compilation unit, package, or inside the module or interface.

Example – Using Type Operator to Declare Unpacked Data Type

```
typedef logic[4:1] logicdt;
typedef type(logicdt)Unpackdt[2:1];

module top(
   input Unpackdt d1,
   output Unpackdt dout1 );
assign dout1[2] = d1[2];
assign dout1[1] = d1[1];
endmodule
```

Type Casting

The type operator can be used to directly reference the data type of a variable or port, or can be user-defined and used in type casting to convert either signed to unsigned or unsigned to signed.

Example – Using Type Operator to Reference Data Type

Copyright © 2013 Synopsys, Inc.

```
typedef logic [20:0]dt;
//Module top
module top (
   input byte d1,d2,
   output int unsigned dout1 );
assign dout1 = type(dt)'(d1 * d2);
endmodule
```

Defining Type Parameter/Local Parameter

The type operator can be used when defining a Type parameter to define the data type. The definition can be overridden based on user requirements.

Example – Using Type Operator to Declare Parameter Type Value

```
// Module top
module top(
   input byte a1,
   input byte a2,
   output shortint dout1 );
parameter type dtype = type(a1);
dtype sig1;
assign sig1 = a1;
assign dout1 = ~sig1;
endmodule
```

Comparison and Case Comparison

The type operator can be used to compare two types when evaluating a condition or a case statement.

Example – Using Type Operator in a Comparison

```
// Module top
module top (
   input byte d1,
   input shortint d2,
   output shortint dout1 );
```

```
always_comb begin
   if(type(d1) == type(d2))
      dout1 = d1;
   else
      dout1 = d2;
end
endmodule
```

Limitations

The type operator is not supported on complex expressions (for example type(d1*d2)).

Procedural Statements and Control Flow

Topics in this section include

- Do-While Loops
- For Loops, on page 798
- Unnamed Blocks, on page 798
- Block Name on end Keyword, on page 799
- Unique and Priority Modifiers, on page 799

Do-While Loops

The while statement executes a loop for as long as the loop-control test is true. The control value is tested at the *beginning* of each pass through the loop. However, a while loop does not execute at all if the test on the control value is false the first time the loop is encountered. This top-testing behavior can require extra coding prior to beginning the while loop, to ensure that any output variables of the loop are consistent.

SystemVerilog enhances the for loop and adds a do-while loop, the same as in C. The control on the do-while loop is tested at the *end* of each pass through the loop (instead of at the beginning). This implies that each time the loop is encountered in the execution flow, the loop statements are executed at least once.

Copyright © 2013 Synopsys, Inc.

Because the statements within a do-while loop are going to execute at least once, all the logic for setting the outputs of the loop can be placed inside the loop. This bottom-testing behavior can simplify the coding of while loops, making the code more concise and more intuitive.

Example – Simple Do-while Loop

Example – Do-while with If Else Statement

Example – Do-while with Case Statement

For Loops

SystemVerilog simplifies declaring local variables for use in for loops. The declaration of the for loop variable can be made within the for loop. This eliminates the need to define several variables at the module level, or to define local variables within named begin…end blocks as shown in the following example.

Example – Simple for Loop

A variable defined as in the example above, is local to the loop. References to the variable name within the loop see the local variable, however, reference to the same variable outside the loop encounters an error. This type of variable is created and initialized when the for loop is invoked, and destroyed when the loop exits.

SystemVerilog also enhances for loops by allowing more than one initial assignment statement. Multiple initial or step assignments are separated by commas as shown in the following example.

Example – For Loop with Two Variables

Unnamed Blocks

SystemVerilog allows local variables to be declared in unnamed blocks.

Copyright © 2013 Synopsys, Inc.

Example – Local Variable in Unnamed Block

Block Name on end Keyword

SystemVerilog allows a block name to be defined after the end keyword when the name matches the one defined on the corresponding begin keyword. This means, you can name the start and end of a begin statement for a block. The additional name does not affect the block semantics, but does serve to enhance code readability by documenting the statement group that is being completed.

Example – Including Block Name with end Keyword

Unique and Priority Modifiers

SystemVerilog adds unique and priority modifiers to use in case statements. The Verilog full_case and parallel_case statements are located inside of comments and are ignored by the Verilog simulator. For synthesis, full_case and parallel_case directives instruct the tool to take certain actions or perform certain optimizations that are unknown to the simulator.

To prevent discrepancies when using full_case and parallel_case directives and to ensure that the simulator has the same understanding of them as the synthesis tool, use the priority or unique modifier in the case statement. The priority and unique keywords are recognized by all tools, including the Verilog simulators, allowing all tools to have the same information about the design.

The following table shows how to substitute the SystemVerilog unique and priority modifiers for Verilog full_case and parallel_case directives for synthesis.

Copyright © 2013 Synopsys, Inc.

Verilog using full_case, parallel_case	SystemVerilog using unique/priority case modifiers

`case (...)`
`...`
`endcase`
	`case (...)`
	`...`
	`endcase`

`case (...) //full_case`
`...`
`endcase`
	`priority case (...)`
	`...`
	`endcase`

`case (...) //parallel_case`
`...`
`endcase`
	`unique case (...)`
	`...`
	`default : ...`
	`endcase`

`case (...) //full_case parallel_case`
`...`
`endcase`
	`unique case (...)`
	`...`
	`endcase`

Example – Unique Case

Example – Priority Case

Processes

In Verilog, an "if" statement with a missing "else" condition infers an unintentional latch element, for which the Synopsys FPGA compiler currently generates a warning. Many commercially available compilers do not generate any warning, causing a serious mismatch between intention and inference. SystemVerilog adds three specialized procedural blocks that reduce ambiguity and clearly indicate the intent:

- always_comb, on page 801
- always_latch, on page 803

- always_ff, on page 804

Use them instead of the Verilog general purpose always procedural block to indicate design intent and aid in the inference of identical logic across synthesis, simulation, and formal verification tools.

always_comb

The SystemVerilog always_comb process block models combinational logic, and the logic inferred from the always_comb process must be combinational logic. The Synopsys FPGA compiler warns you if the behavior does not represent combinational logic.

The semantics of an always_comb block are different from a normal always block in these ways:

- It is illegal to declare a sensitivity list in tandem with an always_comb block.
- An always_comb statement cannot contain any block, timing, or event controls and fork, join, or wait statements.

Note the following about the always_comb block:

- There is an inferred sensitivity list that includes all the variables from the RHS of all assignments within the always_comb block and variables used to control or select assignments See *Examples of Sensitivity to LHS and RHS of Assignments*, on page 802.
- The variables on the LHS of the expression should not be written by any other processes.
- The always_comb block is guaranteed to be triggered once at time zero after the initial block is executed.
- always_comb is sensitive to changes within the contents of a function and not just the function arguments, unlike the always@(*) construct of Verilog 2001.

Example – always_comb Block

Invalid Use of always_comb Block

The following code segments show use of the construct that are *NOT VALID*.

Copyright © 2013 Synopsys, Inc.

```
always_comb @(a or b) //Wrong. Sensitivity list is inferred not
   //declared
begin
   foo;
end

always_comb
begin
   @clk out <=in; //Wrong to use trigger within this always block
end

always_comb
begin
   fork //Wrong to use fork-join within this always block
   out <=in;
   join
end

always_comb
begin
   if(en)mem[waddr]<=data; //Wrong to use trigger conditions
      //within this block
end
```

Examples of Sensitivity to LHS and RHS of Assignments

In the following code segment, sensitivity only to the LHS of assignments causes problems.

```
always @(y)
   if (sel)
      y= a1;
   else
      y= a0;
```

In the following code segment, sensitivity only to the RHS of assignments causes problems.

```
always @(a0, a1)
   if (sel)
      y= a1;
   else
      y= a0;
```

In the following code segment, sensitivity to the RHS of assignments and variables used in control logic for assignments produces correct results.

```
always @(a0, a1, sel)
   if (sel)
      y= a1;
   else
      y= a0;
```

always_latch

The SystemVerilog always_latch process models latched logic, and the logic inferred from the always_latch process must only be latches (of any kind). The Synopsys FPGA compiler warns you if the behavior does not follow the intent.

Note the following:

- It is illegal for always_latch statements to contain a sensitivity list, any block, timing, or event controls, and fork, join, or wait statements.
- The sensitivity list of an always_latch process is automatically inferred by the compiler and the inferring rules are similar to the always_comb process (see *always_comb*, on page 801).

Example – always_latch Block

Invalid Use of always_latch Block

The following code segments show use of the construct that are *NOT VALID*.

```
always_latch
begin
   if(en)
      treg<=1;
   else
      treg<=0; //Wrong to use fully specified if statement
end

always_latch
begin
   @(clk)out <=in; //Wrong to use trigger events within this
      //always block
end
```


Copyright © 2013 Synopsys, Inc.

always_ff

The SystemVerilog always_ff process block models sequential logic that is triggered by clocks. The compiler warns you if the behavior does not represent the intent. The always_ff process has the following restrictions:

- An always_ff block must contain only one event control and no blocking timing controls.
- Variables on the left side of assignments within an always_ff block must not be written to by any other process.

Example – always_ff Block

Invalid Use of always_ff Block

The following code segments show use of the construct that are *NOT VALID*.

```
always_ff @(posedge clk or negedge rst)
begin
   if(rst)
      treg<=in; //Illegal; wrong polarity for rst in the
         //sensitivity list and the if statement
end

always_ff
begin
   @(posedgerst)treg<=0;
   @(posedgeclk)treg<=in; //Illegal; two event controls
end

always_ff @(posedge clk or posedge rst)
begin
   treg<=0; //Illegal; not clear which trigger is to be
               // considered clk or rst
end
```

Copyright © 2013 Synopsys, Inc.

Tasks and Functions

Support for task and function calls includes the following:

- Implicit Statement Group
- Formal Arguments, on page 805
- endtask/endfunction Names, on page 808

Implicit Statement Group

Multiple statements in the task or function definition do not need to be placed within a begin…end block. Multiple statements are implicitly grouped, executed sequentially as if they are enclosed in a begin…end block.

```
/* Statement grouping */
function int incr2(int a);
   incr2 = a + 1;
   incr2 = incr2 + 1;
endfunction
```

Formal Arguments

This section includes information on passing formal arguments when calling functions or tasks. Topics include:

- Passing Arguments by Name
- Default Direction and Type
- Default Values

Passing Arguments by Name

When a task or function is called, SystemVerilog allows for argument values to be passed to the task/function using formal argument names; order of the formal arguments is not important. As in instantiations in Verilog, named argument values can be passed in any order, and are explicitly passed through to the specified formal argument. The syntax for the named argument passing is the same as Verilog's syntax for named port connections to a module instance. For example:

Copyright © 2013 Synopsys, Inc.

```
/* General functions */
function [1:0] inc(input [1:0] a);
   inc = a + 1;
endfunction
function [1:0] sel(input [1:0] a, b, input s);
   sel = s ? a : b;
endfunction

/* Tests named connections on function calls */
assign z0 = inc(.a(a));
assign z2 = sel(.b(b), .s(s), .a(a));
```

Default Direction and Type

In SystemVerilog, input is the default direction for the task/function declaration. Until a formal argument direction is declared, all arguments are assumed to be inputs. Once a direction is declared, subsequent arguments will be the declared direction, the same as in Verilog.

The default data type for task/function arguments is logic, unless explicitly declared as another variable type. (In Verilog, each formal argument of a task/function is assumed to be reg). For example:

```
/* Tests default direction of argument */
function int incr1(int a);
   incr1 = a + 1;
endfunction
```

In this case, the direction for a is input even though this is not explicitly defined.

Default Values

SystemVerilog allows an optional default value to be defined for each formal argument of a task or function. The default value is specified using a syntax similar to setting the initial value of a variable. For example:

```
function int testa(int a = 0, int b, int c = 1);
   testa = a + b + c;
endfunction

task testb(int a = 0, int b, int c = 1, output int d);
   d = a + b + c;
endtask
```

When a task/function is called, it is not necessary to pass a value to the arguments that have default argument values. If nothing is passed to the task/function for that argument position, the default value is used. Specifying default argument values allows a task/function definition to be used in multiple ways. Verilog requires that a task/function call have the exact same number of argument expressions as the number of formal arguments. SystemVerilog allows the task/function call to have fewer argument expressions than the number of formal arguments. A task/function call must pass a value to an argument, if the formal definition of the argument does not have a default value. Consider the following examples:

```
/* functions With positional associations and missing arguments */
assign a = testa(,5); /* Same as testa(0,5,1) */
assign b = testa(2,5); /* Same as testa(2,5,1) */
assign c = testa(,5,); /* Same as testa(0,5,1) */
assign d = testa(,5,7); /* Same as testa(0,5,7) */
assign e = testa(1,5,2); /* Same as testa(1,5,2) */

/* functions With named associations and missing arguments */
assign k = testa(.b(5)); /* Same as testa(0,5,1) */
assign l = testa(.a(2),.b(5)); /* Same as testa(2,5,1) */
assign m = testa(.b(5)); /* Same as testa(0,5,1) */
assign n = testa(.b(5),.c(7)); /* Same as testa(0,5,7) */
assign o = testa(.a(1),.b(5),.c(2)); /* Same as testa(1,5,2) */
```

In general, tasks are not supported outside the scope of a procedural block (even in previous versions). This is primarily due to the difference between tasks and function.

Here are some task examples using default values:

```
always @(*)
begin
/* tasks With named associations and missing arguments */
testb(.b(5),.d(f)); /* Same as testb(0,5,1) */
testb(.a(2),.b(5),.d(g)); /* Same as testb(2,5,1) */
testb(.b(5),.d(h)); /* Same as testb(0,5,1) */
testb(.b(5),.c(7),.d(i)); /* Same as testb(0,5,7) */
testb(.a(1),.b(5),.c(2),.d(j)); /* Same as testb(1,5,2) */

/* tasks With positional associations and missing arguments */
testb(,5,,p); /* Same as testb(0,5,1) */
testb(2,5,,q); /* Same as testb(2,5,1) */
testb(,5,,r); /* Same as testb(0,5,1) */
testb(,5,7,s); /* Same as testb(0,5,7) */
testb(1,5,2,t); /* Same as testb(1,5,2) */
```


Copyright © 2013 Synopsys, Inc.

endtask/endfunction Names

SystemVerilog allows a name to be specified with the endtask or endfunction keyword. The syntax is:

endtask : *taskName*

endfunction : *functionName*

The space before and after the colon is optional. The name specified must be the same as the name of the corresponding task or function as shown in the following example.

```
/* Function w/ statement grouping, also has an endfunction label */

function int incr3(int a);
   incr3 = a + 1;
   incr3 = incr3 + 1;
   incr3 = incr3 + 1;
endfunction : incr3

/* Test with a task - also has an endtask label */
task task1;
input [1:0] in1,in2,in3,in4;
output [1:0] out1,out2;
   out1 = in1 | in2;
   out2 = in3 & in4;
endtask : task1

/* Test with a task - some default values */
task task2(
input [1:0] in1=2'b01,in2= 2'b10,in3 = 2'b11,in4 = 2'b11,
output [1:0] out1 = 2'b10,out2);

   out2 = in3 & in4;
endtask : task2

/* Tests default values for arguments */
function int dflt0(input int a = 0, b = 1);
   dflt0 = a + b;
endfunction

/* Call to function with default direction */
assign z1 = incr1(3);
assign z3 = incr2(3);
assign z4 = incr3(3);
assign z9 = dflt0();
assign z10 = dflt0(.a(7), .b());
```

Copyright © 2013 Synopsys, Inc.

```
always @(*)
begin
   task1(.in1(in1), .out2(z6), .in2(in2), .out1(z5),
      .in3(in3), .in4(in4));
   task1(in5, in6, in7, in8, z7, z8);
   task2(in5, in6, in7, in8, z11, z12);
   task2(in5, in6, , , z13, z14);
   task2(.out1(z15), .in1(in5), .in2(in6), .out2(z16),
      .in3(in7), .in4(in8));
   task2(.out2(z18), .in2(in6), .in1(in5), .in3(),
      .out1(z17), .in4());
end
```

Hierarchy

Topics in this section include:

- Compilation Units, below
- Packages, on page 811
- Port Connection Constructs, on page 813
- Extern Module, on page 815

Compilation Units

Compilation units allow declarations to be made outside of a package, module, or interface boundary. These units are visible to all modules that are compiled at the same time.

A compilation unit's scope exists only for the source files that are compiled at the same time; each time a source file is compiled, a compilation unit scope is created that is unique to only that compilation.

Syntax

//$unit definitions

declarations;

Copyright © 2013 Synopsys, Inc.

```
//End of $unit

module ();
. . .
. . .
. . .
endmodule
```

In the above syntax, declarations can be variables, nets, constants, user-defined data types, tasks, or functions

Usage

Compilation units can be used to declare variables and nets, constants, user-defined data types, tasks, and functions as noted in the following examples.

A variable can be defined within a module as well as within a compilation unit. To reference the variable from the compilation unit, use the **$unit::***variableName* syntax. To resolve the scope of a declaration, local declarations must be searched first followed by the declarations in the compilation unit scope.

Example – Compilation Unit Variable Declaration

Example – Compilation Unit Net Declaration

Example – Compilation Unit Constant Declaration

Example – Compilation Unit User-defined Datatype Declaration

Example – Compilation Unit Task Declaration

Example – Compilation Unit Function Declaration

Example – Compilation Unit Access

Example – Compilation Unit Scope Resolution

Copyright © 2013 Synopsys, Inc.

To use the compilation unit for modules defined in multiple files, enable the Multiple File Compilation Unit check box on the Verilog tab of the Implementation Options dialog box as shown below.

You can also enable this compiler directive by including the following Tcl command in your project (`prj`) file:

```
set_option -multi_file_compilation_unit 1
```

Limitations

Compilation unit elements can only be accessed or read, and cannot appear between module and endmodule statements.

Packages

Packages permit the sharing of language-defined data types, typedef user-defined types, parameters, constants, function definitions, and task definitions among one or more compilation units, modules, or interfaces. The concept of packages is leveraged from the VHDL language.

Syntax

SystemVerilog packages are defined between the keywords package and endpackage.

package packageIdentifier;

Copyright © 2013 Synopsys, Inc.

packageItems

endpackage : packageIdentifier

PackageItems includes user-defined data types, parameter declarations, constant declarations, task declarations, function declarations, and import statements from other packages. To resolve the scope of any declaration, the local declarations are always searched before declarations in packages.

Referencing Package Items

As noted in the following examples, package items can be referenced by:

- Direct reference using a scope resolution operator (::). The scope resolution operator allows referencing a package by the package name and then selecting a specific package item.
- Importing specific package items using an import statement to import specific package items into a module.
- Importing package items using a wildcard (*) instead of naming a specific package item.

Example – Direct Reference Using Scope Resolution Operator (::)

Example – Importing Specific Package Items

Example – Wildcard (*) Import Package Items

Example – User-defined Data Types (typedef)

Example – Parameter Declarations

Example – Constant Declarations

Example – Task Declarations

Example – Function Declarations

Copyright © 2013 Synopsys, Inc.

Example – import Statements from Other Packages

Example – Scope Resolution

Limitations

The variables declared in packages can only be accessed or read; package variables cannot be written between a module statement and its end module statement.

Port Connection Constructs

Instantiating modules with a large number of ports is unnecessarily verbose and error-prone in Verilog. The SystemVerilog *.name* and ".*" constructs extend the 1364 Verilog feature of allowing named port connections on instantiations, to implicitly instantiate ports.

.name Connection

The SystemVerilog *.name* connection is semantically equivalent to a Verilog named port connection of type *.port_identifier(name)*. Use the *.name* construct when the name and size of an instance port are the same as those on the module. This construct eliminates the requirement to list a port name twice when both the port name and signal name are the same and their sizes are the same as shown below:

```
module myand(input [2:0] in1, in2, output [2:0] out);
...
endmodule

module foo (….ports….)
wire [2:0] in1, out;
wire [7:0] tmp;
wire [7:0] in2 = tmp;
myand mand1(.in1, .out, .in2(tmp[2:0]));  // valid
```

Note: SystemVerilog *.name* connection is currently not supported for mixed-language designs.

Copyright © 2013 Synopsys, Inc.

Restrictions to the *.name* feature are the same as the restrictions for named associations in Verilog. In addition, the following restrictions apply:

- Named associations and positional associations cannot be mixed:

```
myand mand2(.in1, out, tmp[2:0]);
```

- Sizes must match in mixed named and positional associations. The example below is not valid because of the size mismatch on in2.

```
myand mand3(.in1, .out, .in2);
```

- The identifier referred by the *.name* must not create an implicit declaration, regardless of the compiler directive '*default_nettype*.
- You cannot use the *.name* connection to create an implicit cast.
- Currently, the *.name* port connection is not supported for mixed HDL source code.

.* Connection

The SystemVerilog ".*" connection is semantically identical to the default *.name* connection for every port in the instantiated module. Use this connection to implicitly instantiate ports when the instance port names and sizes match the connecting module's variable port names and sizes. The implicit .* port connection syntax connects all other ports on the instantiated module.Using the .* connection facilitates the easy instantiation of modules with a large number of ports and wrappers around IP blocks.

The ".*" connection can be freely mixed with *.name* and *.port_identifier*(*name*) type connections. However, it is illegal to have more than one ".*" expression per instantiation.

The use of ".*" facilitates easy instantiation of modules with a large number of ports and wrappers around IP blocks as shown in the code segment below:

```
module myand(input [2:0] in1, in2, output [2:0] out);
...
endmodule

module foo (….ports….)
wire [2:0] in1, in2, out;
wire [7:0] tmp;
```

```
myand and1(.*); // Correct usage, connect in1, in2, out
myand and2(.in1, .*) // Correct usage, connect in2 and out
myand and3(.in1(tmp[2:0]), .*); // Correct Usage, connect
   // in2 and out
myand and5(.in1, .in2, .out, .*); //Correct Usage, ignore the .*
```

Note: SystemVerilog ".*" connection is currently not supported for mixed-language designs.

Restrictions to the .* feature are the same as the restrictions for the *.name* feature. See *.name Connection*, on page 813. In addition, the following restrictions apply:

- Named associations and positional associations cannot be mixed. For example

  ```
  myand and4(in1, .*);
  ```

  is illegal (named and positional connections cannot be mixed)
- Named associations where there is a mismatch of variable sizes or names generate an error.
- You can only use the .* once per instantiation, although you can mix the .* connection with *.name* and *.port_identifier(name)* type connections.
- If you use a .* construction but all remaining ports are explicitly connected, the compiler ignores the .* construct.
- Currently, the .* port connection is not supported for mixed HDL source code.

Extern Module

SystemVerilog simplifies the compilation process by allowing you to specify a prototype of the module being instantiated. The prototype is defined using the extern keyword, followed by the declaration of the module and its ports. Either the Verilog-1995 or the Verilog-2001 style of module declaration can be used for the prototype.

The extern module declaration can be made in any module, at any level of the design hierarchy. The declaration is only visible within the scope in which it is defined. Support is limited to declaring extern module outside the module.

Copyright © 2013 Synopsys, Inc.

Syntax

extern module *moduleName* **(***direction port1***,** *direction portVector port2***,** *direction port3***);**

Example 1 – Extern Module Instantiation

Example 2 – Extern Module Reference

Limitations

An extern module declaration is not supported within a module.

Interface

Topics in this section include:

- Interface Construct
- Modports, on page 823
- Limitations and Non-Supported Features, on page 823

Interface Construct

SystemVerilog includes enhancements to Verilog for representing port lists and port connection lists characterized by name repetition with a single name to reduce code size and simplify maintenance. The interface and modport structures in SystemVerilog perform this function. The interface construct includes all of the characteristics of a module with the exception of module instantiation; support for interface definitions is the same as the current support for module definitions. Interfaces can be instantiated and connected to client modules using generates.

Copyright © 2013 Synopsys, Inc.

Interface Definition: Internal Logic and Hierarchical Structure

Per the SystemVerilog standard, an interface definition can contain any logic that a module can contain with the exception that interfaces cannot contain module instantiations. An interface definition can contain instantiations of other interfaces. Like modules, interface port declaration lists can include interface-type ports. Synthesis support for interface logic is the same as the current support for modules.

Port Declarations and Port Connections for Interfaces

Per the SystemVerilog standard, interface port declaration and port connection syntax/semantics are identical to those of modules.

Interface Member Types

The following interface member types are visible to interface clients:

- 4-State var types: reg, logic, integer
- 2-State var types: bit, byte, shortint, int, longint
- Net types: wire, wire-OR, and wire-AND
- Scalars and 1-dimensional packed arrays of above types
- Multi-dimensional packed and unpacked arrays of above types
- SystemVerilog struct types

Interface Member Access

The members of an interface instance can be accessed using the syntax:

interfaceRef.*interfaceMemberName*

In the above syntax, *interfaceRef* is either:

- the name of an interface-type port of the module/interface containing the member access
- the name of an interface instance that is instantiated directly within the module/interface containing the member access.

Note that reference to interface members using full hierarchical naming is not supported and that only the limited form described above for instances at the current level of hierarchy is supported.

Copyright © 2013 Synopsys, Inc.

Access to an interface instance by clients at lower levels of the design hierarchy is achieved by connecting the interface instance to a compatible interface-type port of a client instance and connecting this port to other compatible interface-type ports down the hierarchy as required. This chaining of interface ports can be done to an arbitrary depth. Note that interface instances can be accessed only by clients residing at the same or lower levels of the design hierarchy.

Interface-Type Ports

Interface-type ports are supported as described in the SystemVerilog standard, and generic interface ports are supported. A modport qualifier can appear in either a port declaration or a port connection as described in the SystemVerilog standard. Interface-type ports:

- can appear in either module or interface port declarations
- can be used to access individual interface items using "." syntax:

  interfacePortname.interfaceMemberName

- can be connected directly to compatible interface ports of module/interface instances

Interface/Module Hierarchy

Interfaces can be instantiated within either module or interface definitions. See *Interface Member Access*, on page 817 for additional details on hierarchical interface port connections.

Interface Functions and Tasks

Import-only functions and tasks (using import keyword in modport) are supported.

Element access outside the interface

Interface can have a collection of variables or nets, and this collection can be of a language-defined data type, user-defined data type, or array of language and user-defined data type. All of these variables can be accessed outside the interface.

The following example illustrates accessing a 2-dimensional structure type defined within the interface that is being accessed from another module.

Copyright © 2013 Synopsys, Inc.

Example – Accessing a 2-dimensional structure

```
typedef struct
{
   byte st1;
}Struct1D_Dt[1:0][1:0];

//Interface Definition
interface intf(
   input bit clk,
   input bit rst
);

   Struct1D_Dt i1; //2D - Structure type
   modport MP( input i1,input clk,input rst); //Modport Definition
endinterface

//Sub1 Module definition
module sub1(
   intf INTF1, //Interface
   input int d1
);

   assign INTF1.i1[1][1].st1 = d1[7:0];
   assign INTF1.i1[1][0].st1 = d1[15:8];
   assign INTF1.i1[0][1].st1 = d1[23:16];
   assign INTF1.i1[0][0].st1 = d1[31:24];
endmodule

//Sub2 Module definition
module sub2(
   intf.MP IntfMp, //Modport Interface
   output byte dout[3:0]
);

always_ff@(posedge IntfMp.clk)
begin
   if(IntfMp.rst)
   begin
      dout <= '{default:'1};
   end
   else begin
      dout[3] <= IntfMp.i1[1][1].st1;
      dout[2] <= IntfMp.i1[1][0].st1;
```


Copyright © 2013 Synopsys, Inc.

```
      dout[1] <= IntfMp.i1[0][1].st1;
      dout[0] <= IntfMp.i1[0][0].st1;
   end
end
endmodule

//Top Module definition
module top(
   input bit clk,
   input bit rst,
   input int d1,
   output byte dout[3:0]
);
intf intu1(.clk(clk),.rst(rst));
sub1 sub1u1(.INTF1(intu1),.d1(d1));
sub2 sub2u1(.IntfMp(intu1.MP),.dout(dout));
endmodule
```

Nested Interface

With the nested interface feature, nesting of interface is possible by either instantiating one interface in another or by using one interface as a port in another interface. Generic interface is not supported for nested interface; array of interface when using interface as a port also is not supported.

The following example illustrates the use of a nested interface. In the example, one interface is instantiated within another interface and this top-level interface is used in the modules.

Example – Nested Interface

```
//intf1 Interface definition
interface intf1;
   byte i11;
   byte i12;
endinterface

//IntfTop Top Interface definition
interface IntfTop;
   intf1 intf1_u1(); //Interface instantiated
   shortint i21;
endinterface
```

```
//Sub1 Module definition
module sub1(
   input byte d1,
   input byte d2,
   IntfTop intfN1
);
assign intfN1.intf1_u1.i11 = d1; //Nested interface being accessed
assign intfN1.intf1_u1.i12 = d2; //Nested interface being accessed
endmodule

//Sub2 Module definition
module sub2(
   IntfTop intfN2
);
assign intfN2.i21 = intfN2.intf1_u1.i11 + intfN2.intf1_u1.i12;
//Nested
   //interface being accessed
endmodule

//Sub3 Module definition
module sub3(
   IntfTop intfN3,
   output shortint dout
);
assign dout = intfN3.i21;
endmodule

//Top Module definition
module top(
   input byte d1,
   input byte d2,
   output shortint dout
);
IntfTop IntfTopU1();
   sub1 sub1U1(.d1(d1),.d2(d2),.intfN1(IntfTopU1));
   sub2 sub2U1(.intfN2(IntfTopU1));
   sub3 sub3U1(.intfN3(IntfTopU1), .dout(dout));
endmodule
```

Arrays of Interface Instances

In Verilog, multiple instances of the same module can be created using the array of instances concept. This same concept is extended for the interface construct in SystemVerilog to allow multiple instances of the same interface

Copyright © 2013 Synopsys, Inc.

to be created during component instantiation or during port declaration. These arrays of interface instances and slices of interface instance arrays can be passed as connections to arrays of module instances across modules.

The following example illustrates the use of array of interface instance both during component instantiation and during port declaration.

Example – Array of interface during port declaration

```
//intf Interface Definition
interface intf;
   byte i1;
endinterface

//Sub1 Module definition
module sub1(
   intf IntfArr1 [3:0], //Array of interface during port
declaration
   input byte d1[3:0]
);
assign IntfArr1[0].i1 = d1[0];
assign IntfArr1[1].i1 = d1[1];
assign IntfArr1[2].i1 = d1[2];
assign IntfArr1[3].i1 = d1[3];
endmodule

//Sub2 Module definition
module sub2(
   intf IntfArr2[3:0], //Array of interface during port
declaration
   output byte dout[3:0]
);
assign dout[0] = IntfArr2[0].i1;
assign dout[1] = IntfArr2[1].i1;
assign dout[2] = IntfArr2[2].i1;
assign dout[3] = IntfArr2[3].i1;
endmodule
```

Copyright © 2013 Synopsys, Inc.

```
//Top module definition
module top(
   input byte d1[3:0],
   output byte dout[3:0]
);
intf intfu1[3:0]() ; //Array of interface instances
   sub1 sub1u1(intfu1,d1);
   sub2 sub2u1(intfu1,dout);
endmodule
```

Modports

Modport expressions are supported, and modport selection can be done in either the port declaration of a client module or in the port connection of a client module instance.

If a modport is associated with an interface port or instance through a client module, the module can only access the interface members enumerated in the modport. However, per the SystemVerilog standard, a client module is not constrained to use a modport, in which case it can access any interface members.

Modport Keywords

The input, output, inout, and import access modes are parsed without errors. The signal direction for input, output, and inout is ignored during synthesis, and the correct signal polarity is inferred from how the interface signal is used within the client module. The signal polarity keywords are ignored because the precise semantics are currently not well-defined in the SystemVerilog standard, and simulator support has yet to be standardized.

Example – Instantiating an interface Construct

Limitations and Non-Supported Features

The following restrictions apply when using interface/modport structures:

- Declaring interface within another interface is not supported
- Direction information in modports has no effect on synthesis.

Copyright © 2013 Synopsys, Inc.

- Exported (export keyword) interface functions and tasks are not supported.
- Virtual interfaces are not supported.
- Full hierarchical naming of interface members is not supported.
- Modports defined within generate statements are not supported.

System Tasks and System Functions

Topics in this section include:

- $bits System Function, on page 825
- Array Querying Functions, on page 826

$bits System Function

SystemVerilog supports a $bits system function which returns the number of bits required to hold an expression as a bit stream. The syntax is:

$bits(*datatype***)**

$bits(*expression***)**

In the above syntax, *datatype* can be any language-defined data type (reg, wire, integer, logic, bit, int, longint, or shortint) or user-defined datatype (typedef, struct, or enum) and *expression* can be any value including packed and unpacked arrays.

The $bits system function is synthesizable and can be used with any of the following applications:

- Port Declaration
- Variable Declaration
- Constant Definition
- Function Definition

System tasks and system functions are described in Section 22 of IEEE Std 1800-2005 (IEEE Standard for SystemVerilog); $bits is described in Section 22.3.

Example – $bits System Function

Example – $bits System Function within a Function

Copyright © 2013 Synopsys, Inc.

Array Querying Functions

SystemVerilog provides system functions that return information about a particular dimension of an array.

Syntax

arrayQuery **(***arrayIdentifier***[,***dimensionExpression***]);**
arrayQuery **(***dataTypeName***[,***dimensionExpression***]);**
$dimensions | $unpacked_dimensions (*arrayIdentifier* | *dataTypeName***)**

In the above syntax, *arrayQuery* is one of the following array querying functions:

- **$left** – returns the left bound (MSB) of the dimension.
- **$right** – returns the right bound (LSB) of the dimension.
- **$low** – returns the lowest value of the left and right bound dimension.
- **$high** – returns the highest value of the left and right bound dimension.
- **$size** – returns the number of elements in a given dimension.
- **$increment** – returns a value "1" when the left bound is greater than or equal to the right bound, else it returns a value "-1".

In the third syntax example, $dimensions returns the total number of packed and unpacked dimensions in a given array, and $unpacked_dimensions returns the total number of unpacked dimensions in a given array. The variable *dimensionExpression*, by default, is "1". The order of dimension expression increases from left to right for both unpacked and packed dimensions, starting with the unpacked dimension for a given array.

Example 1 – Array Querying Function $left and $right Used on Packed 2D-data Type

Example 2 – Array Querying Function $low and $high Used on Unpacked 3D-data Type

Example 3 – Array Querying Function $size and $increment Used on a Mixed Array

Copyright © 2013 Synopsys, Inc.

Example 4 – Array Querying Function $dimensions and $unpacked_dimensions Used on a Mixed Array

Example 5 – Array Querying Function with Data Type as Input

Generate Statement

The synthesis tools support the Verilog 2005 generate statement, which conforms to the Verilog 2005 LRM. The tools also support defparam, parameter, and function and task declarations within generate statements. The naming scheme for registers and instances is also enhanced to include closer correlation to specified generate symbolic hierarchies. Generated data types have unique identifier names and can be referenced hierarchically. Generate statements are created using one of the following three methods: generate-loop, generate-conditional, or generate-case.

Note: The generate statement is a Verilog 2005 feature; to use this statement with the FPGA synthesis tools, you must enable SystemVerilog for your project.

Example 1 – Shift Register Using generate-for

Example 2 – Accessing Variables Declared in a generate-if

Example 3 – Accessing Variables Declared in a generate-case

Limitations

The following generate statement functions are not currently supported:

- Defparam support for generate instances
- Hierarchical access for interface
- Hierarchical access of function/task defined within a generate block

Copyright © 2013 Synopsys, Inc.

Note: Whenever the generate statement contains symbolic hierarchies separated by a hierarchy separator (.), the instance name includes the (\) character before this hierarchy separator (.).

Conditional Generate Constructs

The if-generate and case-generate conditional generate constructs allow the selection of, at most, one generate block from a set of alternative generate blocks based on constant expressions evaluated during elaboration. The generate and endgenerate keywords are optional.

Generate blocks in conditional generate constructs can be either named or unnamed and can consist of only a single item. It is not necessary to enclose the blocks with begin and end keywords; the block is still a generate block and, like all generate blocks, comprises a separate scope and a new level of hierarchy when it is instantiated. The if-generate and case-generate constructs can be combined to form a complex generate scheme.

Note: Conditional generate constructs are a Verilog 2005 feature; to use these constructs with the FPGA synthesis tools, you must enable SystemVerilog for your project.

Example 1 – Conditional Generate: if-generate

```
// test.v
module test
#  (parameter width = 8,
    sel = 2 )

   (input clk,
    input [width-1:0] din,
    output [width-1:0] dout1,
    output [width-1:0] dout2 );

if(sel == 1)
   begin:sh
   reg [width-1:0] sh_r;

   always_ff @ (posedge clk)
      sh_r <= din;
   end
else
   begin:sh
      reg [width-1:0] sh_r1;
      reg [width-1:0] sh_r2;
```


Copyright © 2013 Synopsys, Inc.

```
   always_ff @ (posedge clk)
   begin
      sh_r1 <= din;
      sh_r2 <= sh_r1;
   end
end

assign dout1 = sh.sh_r1;
assign dout2 = sh.sh_r2;

endmodule
```

Example 2 – Conditional Generate: case-generate

```
// top.v
module top
#  (parameter mod_sel = 3,
    mod_sel2 = 3,
    width1 = 8,
    width2 = 16 )

   (input [width1-1:0] a1,
    input [width1-1:0] b1,
    output [width1-1:0] c1,
    input [width2-1:0] a2,
    input [width2-1:0] b2,
    output [width2-1:0] c2 );

case(mod_sel)
   0:
      begin:u1
         my_or u1(.a(a1),.b(b1),.c(c1) );
      end
   1:
      begin:u1
         my_and u2(.a(a2),.b(b2),.c(c2) );
      end
   default:
      begin:u1
         my_or u1(.a(a1),.b(b1),.c(c1) );
      end
endcase
```

```
case(mod_sel2)
   0:
      begin:u3
         my_or u3(.a(a1),.b(b1),.c(c1) );
      end
   1:
      begin:u4
         my_and u4(.a(a2),.b(b2),.c(c2) );
      end
   default:
      begin:def
         my_and u2(.a(a2),.b(b2),.c(c2) );
      end
endcase
endmodule

// my_and.v
module my_and
# (parameter width2 = 16 )

   (input [width2-1:0] a,
    input  [width2-1:0] b,
    output [width2-1:0] c
   );

assign c = a & b;
endmodule

// my_or.v
module  my_or
#  (parameter width = 8)

   (input [width-1:0] a,
    input [width-1:0] b,
    output [width-1:0] c );

assign c = a | b;
endmodule
```


Copyright © 2013 Synopsys, Inc.

Assertions

The parsing of SystemVerilog Assertions (SVA) is supported as outlined in the following table.

Assertion Construct	Support Level	Comment
Immediate assertions	Supported	
Concurrent assertions	Partially Supported, Ignored	Multiclock properties are not supported
Boolean expressions	Partially Supported, Ignored	In the boolean expressions, $rose function having a clocking event is not supported.
Sequence	Supported, ignored	
Declaring sequences	Partially Supported, Ignored	Sequence with ports declared in global space is not supported
Sequence operations	Partially Supported, Ignored	All variations of first_match, within and intersect in a sequence is not supported.
Manipulating data in a sequence	Partially Supported, Ignored	More than one assignment in the parenthesis is not supported.
Calling subroutines on sequence match	Partially Supported, Ignored	Calling of more than one tasks is not supported
System functions	Partially Supported	System functions $onehot, $onehot1, and $countones supported; $isunknown not supported
Declaring properties	Partially Supported, Ignored	Declaring of properties in a package and properties with ports declared in global space are not supported
Multiclock support	Not Supported	
Clock resolutions	Partially Supported, Ignored	Default clocking is not supported

Copyright © 2013 Synopsys, Inc.

Assertion Construct	Support Level	Comment
Binding properties to scopes or instances	Not Supported	
Expect statement	Not Supported	
Clocking blocks and concurrent assertions	Not Supported	

SVA System Functions

SystemVerilog assertion support includes the $onehot, $onehot0, and $countones system functions. These functions check for specific characteristics on a particular signal and return a single-bit value.

- $onehot returns true when only one bit of the expression is true.
- $onehot0 returns true when no more than one bit of the expression is high (either one bit high or no bits are high).
- $countones returns true when the number of ones in a given expression matches a predefined value.

Syntax

$onehot (*expression*)

$onehot0 (*expression*)

$countones (*expression*)

Example 1 – System Function within if Statement

The following example shows a $onehot/$onehot0 function used inside an if statement and ternary operator.

```
module top
   (
   //Input
   input byte d1,
   input byte d2,
   input shortint d3,
```


Copyright © 2013 Synopsys, Inc.

```
   //Output
   output byte dout1,
   output byte dout2
   );
byte sig1;
assign sig1 = d1 + d2;

//Use of $onehot
always_comb begin
   if($onehot(sig1))
      dout1 = d3[7:0];
   else
      dout1 = d3[15:8];
end

byte sig2;
assign sig2 = d1 ^ d2;
//Use of $onehot0
assign dout2 = $onehot0(sig2)? d3[7:0] : d3[15:8];

endmodule
```

Example 2 – System Function with Expression

The following example includes an expression, which is evaluated to a single-bit value, as an argument to a system function.

```
module top
   (
   //Input
   input byte d1,
   input byte d2,
   input shortint d3,
   //Output
   output byte dout1,
   output byte dout2
   );

//Use of $onehot with Expression inside onehot function
always@*
begin
   if($onehot((d1 == d2) ? d1[3:0] : d1[7:4]))
      dout1 = d3[7:0];
   else
      dout1 = d3[15:8];
end
```

```
//Use of $onehot0 with AND operation inside onehot function
assign dout2 = $onehot0(d1 & d2 )? d3[7:0] : d3[15:8];

endmodule
```

Example 3 – Ones Count

In the following example, a 4-bit count is checked for two and only two bits set to 1 which, when present, returns true.

```
module top(
   input clk,
   input rst,
   input byte d1,
   output byte dout
);
logic[3:0] count;

always_ff@(posedge clk)begin
   if(rst)
      count <= '0;
   else
      count <= count + 1'b1;
end

assign dout = $countones(count) == 3'd2 ? d1 : ~d1;
endmodule
```


Copyright © 2013 Synopsys, Inc.

Keyword Support

This table lists supported SystemVerilog keywords in the Synopsys FPGA synthesis tools:

always_comb	always_ff	always_latch	assert*
assume*	automatic	bind*	bit
break	byte	checker*	clocking*
const	continue	cover*	do
endchecker*	endclocking*	endinterface	endproperty*
endsequence*	enum	expect*	extern
final*	function	global*	import
inside	int	interface	intersect*
let*	logic	longint	modport
packed	package	parameter	priority
property*	restrict*	return	sequence*
shortint	struct	task	throughout*
timeprecision*	timeunit*	typedef	union
unique	void	within*	

* Reserved keywords for SystemVerilog assertion parsing; cannot be used as identifiers or object names

Copyright © 2013 Synopsys, Inc.

CHAPTER 11

VHDL Language Support

This chapter discusses how you can use the VHDL language to create HDL sour ce code for the synthesis tool:

- Language Constructs, on page 838
- VHDL Language Constructs, on page 840
- VHDL Implicit Data-type Defaults, on page 874
- VHDL Synthesis Guidelines, on page 879
- Sets and Resets, on page 893
- VHDL State Machines, on page 897
- Hierarchical Design Creation in VHDL, on page 906
- Configuration Specification and Declaration, on page 910
- Scalable Designs, on page 934
- VHDL Guidelines for RAM Inference, on page 940
- VHDL Coding for Altera RAM Inference, on page 948
- VHDL Coding for Xilinx RAM Inference, on page 952
- Instantiating RAMs with SYNCORE, on page 957
- ROM Inference, on page 958
- Instantiating Black Boxes in VHDL, on page 961
- VHDL Attribute and Directive Syntax, on page 963
- VHDL Synthesis Examples, on page 965
- PREP VHDL Benchmarks, on page 967

Copyright © 2013 Synopsys, Inc.

Language Constructs

This section generally describes how the synthesis tool relates to different VHDL language constructs. The topics include:

- Supported VHDL Language Constructs, on page 838
- Unsupported VHDL Language Constructs, on page 839
- Partially-supported VHDL Language Constructs, on page 840
- Ignored VHDL Language Constructs, on page 840

Supported VHDL Language Constructs

The following is a compact list of language constructs that are supported.

- Entity, architecture, and package design units
- Function and procedure subprograms
- All IEEE library packages, including:
  - std_logic_1164
  - std_logic_unsigned
  - std_logic_signed
  - std_logic_arith
  - numeric_std and numeric_bit
  - standard library package (std)
- In, out, inout, buffer, linkage ports
- Signals, constants, and variables
- Aliases
- Integer, physical, and enumeration data types; subtypes of these
- Arrays of scalars and records
- Record data types
- File types
- All operators (-, -, *, /, **, mod, rem, abs, not, =, /=, <, <=, >, >=, and, or, nand, nor, xor, xnor, sll, srl, sla, sra, rol, ror, &)

Copyright © 2013 Synopsys, Inc.

Note: With the ** operator, arguments are compiler constants. When the left operand is 2, the right operand can be a variable.

- Sequential statements: signal and variable assignment, wait, if, case, loop, for, while, return, null, function, and procedure call
- Concurrent statements: signal assignment, process, block, generate (for and if), component instantiation, function, and procedure call
- Component declarations and four methods of component instantiations
- Configuration specification and declaration
- Generics; attributes; overloading
- Next and exit looping control constructs
- Predefined attributes: t'base, t'left, t'right, t'high, t'low, t'succ, t'pred, t'val, t'pos, t'leftof, t'rightof, integer'image, a'left, a'right, a'high, a'low, a'range, a'reverse_range, a'length, a'ascending, s'stable, s'event
- Unconstrained ports in entities
- Global signals declared in packages

Unsupported VHDL Language Constructs

If any of these constructs are found, an error message is reported and the synthesis run is cancelled.

- Register and bus kind signals
- Guarded blocks
- Expanded (hierarchical) names
- User-defined resolution functions. The synthesis tool only supports the resolution functions for std_logic and std_logic_vector.
- Slices with range indices that do not evaluate to constants

Copyright © 2013 Synopsys, Inc.

Partially-supported VHDL Language Constructs

When one of the following constructs in encountered, compilation continues, but will subsequently error out if logic must be generated for the construct.

- real data types (real data expressions are supported in VHDL-2008 IEEE float_pkg.vhd) – real data types are supported as constant declarations or as constants used in expressions as long as no floating point logic must be generated
- access types
- null arrays – null arrays are allowed as operands in concatenation expressions

Ignored VHDL Language Constructs

The synthesis tool ignores the following constructs in your design. If found, the tool parses and ignores the construct (provided that no logic is required to be synthesized) and continues with the synthesis run.

- disconnect
- assert and report
- initial values on inout ports

VHDL Language Constructs

This section describes the synthesis language support that the synthesis tool provides for each VHDL construct. The language information is taken from the most recent VHDL Language Reference Manual (Revision ANSI/IEEE Std 1076-1993). The section names match those from the LRM, for easy reference.

- Data Types
- Declaring and Assigning Objects in VHDL
- VHDL Dynamic Range Assignments
- Signals and Ports

Copyright © 2013 Synopsys, Inc.

- Variables
- VHDL Constants
- Libraries and Packages
- Operators
- VHDL Process
- Common Sequential Statements
- Concurrent Signal Assignments
- Resource Sharing
- Combinational Logic
- Sequential Logic
- Component Instantiation in VHDL
- VHDL Selected Name Support
- User-defined Function Support
- Demand Loading

Data Types

Predefined Enumeration Types

Enumeration types have a fixed set of unique values. The two predefined data types – bit and Boolean, as well as the std_logic type defined in the std_logic_1164 package are the types that represent hardware values. You can declare signals and variables (and constants) that are vectors (arrays) of these types by using the types bit_vector, and std_logic_vector. You typically use std_logic and std_logic_vector, because they are highly flexible for synthesis and simulation.

std_logic Values	Treated by the synthesis tool as...
'U' (uninitialized)	don't care
'X' (forcing unknown)	don't care
'0' (forcing logic 0)	logic 0

Copyright © 2013 Synopsys, Inc.

std_logic Values	Treated by the synthesis tool as...
'1' (forcing logic 1)	logic 1
'Z' (high impedance)	high impedance
'W' (weak unknown)	don't care
'L' (weak logic 0)	logic 0
'H' (weak logic 1)	logic 1
'-' (don't care)	don't care

bit Values	Treated by the synthesis tool a...
'0'	logic 0
'1'	logic 1

boolean Values	Treated by the synthesis tool as...
false	logic 0
true	logic 1

User-defined Enumeration Types

You can create your own enumerated types. This is common for state machines because it allows you to work with named values rather than individual bits or bit vectors.

Syntax

type *type_name* **is (***value_list***) ;**

Examples

```
type states is ( state0, state1, state2, state3);
type traffic_light_state is ( red, yellow, green);
```

Integers

An integer is a predefined VHDL type that has 32 bits. When you declare an object as an integer, restrict the range of values to those you are using. This results in a minimum number of bits for implementation and on ports.

Data Types for Signed and Unsigned Arithmetic

For signed arithmetic, you have the following choices:

- Use the std_logic_vector data type defined in the std_logic_1164 package, and the package std_logic_signed.
- Use the signed data type, and signed arithmetic defined in the package std_logic_arith.
- Use an integer subrange (for example: signal mysig: integer range -8 to 7). If the range includes negative numbers, the synthesis tool uses a two's-complement bit vector of minimum width to represent it (four bits in this example). Using integers limits you to a 32-bit range of values, and is typically only used to represent small buses. Integers are most commonly used for indexes.
- Use the signed data type from the numeric_std or numeric_bit packages.

For unsigned arithmetic, you have the following choices:

- Use the std_logic_vector data type defined in the std_logic_1164 package and the package std_logic_unsigned.
- Use the unsigned data type and unsigned arithmetic defined in the package std_logic_arith.
- Use an integer subrange (for example: signal mysig: integer range 0 to 15). If the integers are restricted to positive values, the synthesis tool uses an unsigned bit vector of minimum width to represent it (four bits in this example). Using integers limits you to a 32-bit range of values, and is typically only used to represent small buses (integers are most commonly used for indexes).
- Use the unsigned data type from the numeric_std or numeric_bit packages.

Copyright © 2013 Synopsys, Inc.

Declaring and Assigning Objects in VHDL

VHDL objects (object classes) include signals (and ports), variables, and constants. The synthesis tool does not support the file object class.

Naming Objects

VHDL is case insensitive. A VHDL name (identifier) must start with a letter and can be followed by any number of letters, numbers, or underscores ('_'). Underscores cannot be the first or last character in a name and cannot be used twice in a row. No special characters such as '$', '?', '*', '-', or '!', can be used as part of a VHDL identifier.

Syntax

object_class object_name : *data_type* [:= *initial_value*] ;

- In the above syntax:
- object_class is a signal, variable, or constant.
- object_name is the name (the identifier) of the object.
- data_type can be any predefined data type (such as bit or std_logic_vector) or user-defined data type.

Assignment Operators

<= Signal assignment operator.

:= Variable assignment and initial value operator.

VHDL Dynamic Range Assignments

The tools support VHDL assignments with dynamic ranges, which are defined as follows:

```
A(b downto c) <= D(e downto f);
```

A and D are constrained variables or signals, and b, c, e, and f are constants (generics) or variables. Dynamic range assignments can be used for RHS, LHS, or both.

Copyright © 2013 Synopsys, Inc.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity test is
   port (data_out: out  std_logic_vector(63 downto 0);
         data_in: in std_logic_vector(63 downto 0);
         selector: in NATURAL range 0 to 7 );
end test;

architecture rtl of test is
begin
   data_out( (selector*8)+7 downto (selector*8))
      <= data_in((selector*8)+7 downto (selector*8));
end rtl;
```

Currently, the following limitations apply to dynamic range assignments:

- There is no support for procedures.
- There is no support for selected signal assignment; i.e., with *expression* Select.
- There is no support for use with concatenation operators.

Null Ranges

A null range is a range that specifies an empty subset of values. A range specified as *m* to *n* is a null range when *m* is greater than *n*, and a range specified as *n* downto *m* is a null range when *n* is less than *m*.

Support for null ranges allows ports with negative ranges to be compiled successfully. During compilation, any port declared with a null range and its related logic are removed by the compiler.

In the following example, port a_in1 (-1 to 0) is a null range and is subsequently removed by the compiler.

```
-- top.vhd
library ieee;
use ieee.std_logic_1164.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity top is
generic (width : integer := 0);
   port (a_in1 : in std_logic_vector(width -1 downto 0);
         b_in1 : in std_logic_vector(3 downto 0);
         c_out1 : out std_logic_vector(3 downto 0) );
end top;

architecture struct of top is
component sub is
   port (a_in1 : in std_logic_vector(width -1 downto 0);
         b_in1 : in std_logic_vector(3 downto 0);
         c_out1 : out std_logic_vector(3 downto 0) );
end component;

begin
   UUT : sub port map ( a_in1 => a_in1, b_in1 => b_in1,
      c_out1 => c_out1);
end struct;

-- sub.vhd
library ieee;
use ieee.std_logic_1164.all;

entity sub is
generic (width : integer := 0);
   port (a_in1 : in std_logic_vector(width -1 downto 0);
         b_in1 : in std_logic_vector(3 downto 0);
         c_out1 : out std_logic_vector(3 downto 0) );
end sub;

architecture rtl of sub is
begin
   c_out1 <= not (b_in1 & a_in1);
end rtl;
```

Signals and Ports

In VHDL, the port list of the entity lists the I/O signals for the design. Ports of mode in can be read from, but not assigned (written) to. Ports of mode out can be assigned to, but not read from. Ports of mode inout are bidirectional and can be read from and assigned to. Ports of mode buffer are like inout ports but can have only one internal driver on them.

Internal signals are declared in the architecture declarative area and can be read from or assigned to anywhere within the architecture.

Examples

```
signal my_sig1 : std_logic; -- Holds a single std_logic bit
begin -- An architecture statement area
my_sig1 <= '1'; -- Assign a constant value '1'

-- My_sig2 is a 4-bit integer
   signal my_sig2 : integer range 0 to 15;
begin -- An architecture statement area
my_sig2 <= 12;

-- My_sig_vec1 holds 8 bits of std_logic, indexed from 0 to 7
   signal my_sig_vec1 : std_logic_vector (0 to 7) ;
begin -- An architecture statement area

-- Simple signal assignment with a literal value
my_sig_vec1 <= "01001000";

-- 16 bits of std_logic, indexed from 15 down to 0
   signal my_sig_vec2 : std_logic_vector (15 downto 0) ;
begin -- An architecture statement area

-- Simple signal assignment with a vector value
my_sig_vec2 <= "0111110010000101";

-- Assigning with a hex value FFFF
my_sig_vec2 <= X"FFFF";

-- Use package Std_Logic_Signed
   signal my_sig_vec3 : signed (3 downto 0);
begin -- An architecture statement area

-- Assigning a signed value, negative 7
my_sig_vec3 <= "1111";

-- Use package Std_Logic_Unsigned
   signal my_sig_vec4 : unsigned (3 downto 0);
begin -- An architecture statement area

-- Assigning an unsigned value of 15
my_sig_vec4 <= "1111";

-- Declare an enumerated type, a signal of that type, and
-- then make an valid assignment to the signal
   type states is ( state0, state1, state2, state3);
   signal current_state : states;
begin -- An architecture statement area
current_state <= state2;
```


Copyright © 2013 Synopsys, Inc.

```
-- Declare an array type, a signal of that type, and
-- then make a valid assignment to the signal

   type array_type is array (1 downto 0) of
      std_logic_vector (7 downto 0);
   signal my_sig: array_type;
begin -- An architecture statement area
my_sig <= ("10101010","01010101");
```

Variables

VHDL variables are declared within a process or subprogram and then used internally. Generally, variables are not visible outside the process or subprogram where they are declared unless passed as a parameter to another subprogram.

Example

```
process (clk) -- What follows is the process declaration area
   variable my_var1 : std_logic := '0'; -- Initial value '0'

begin -- What follows is the process statement area
   my_var1 := '1';
end process;
```

Example

```
process (clk, reset)
-- Set the initial value of the variable to hex FF
   variable my_var2 : std_logic_vector (1 to 8) := X"FF";

begin
-- my_var2 is assigned the octal value 44
   my_var2 := O"44";
end process;
```

VHDL Constants

VHDL constants are declared in any declarative region and can be used within that region. The value of a constant cannot be changed.

Copyright © 2013 Synopsys, Inc.

Example

```
package my_constants is
   constant num_bits : integer := 8;

-- Other package declarations

end my_constants;
```

Libraries and Packages

When you want to synthesize a design in VHDL, include the HDL files in the source files list of your synthesis tool project. Often your VHDL design will have more than one source file. List all the source files in the order you want them compiled; the files at the top of the list are compiled first.

Compiling Design Units into Libraries

All design units in VHDL, including your entities and packages are compiled into libraries. A library is a special directory of entities, architectures and/or packages. You compile source files into libraries by adding them to the source file list. In VHDL, the library you are compiling has the default name work. All entities and packages in your source files are automatically compiled into work. You can keep source files anywhere on your disk, even though you add them to libraries. You can have as many libraries as are needed.

1. To add a file to a library, select the file in the Project view.

   The library structure is maintained in the Project view. The name of the library where a file belongs appears on the same line as the filename, and directly in front of it.

2. Choose Project -> Set Library from the menu bar, then type the library name. You can add any number of files to a library.

3. If you want to use a design unit that you compiled into a library (one that is no longer in the default work library), you must use a library clause in the VHDL source code to reference it.

   For example, if you add a source file for the entity ram16x8 to library my_rams, and instantiate the 16x8 RAM in the design called top_level, you must add library my_rams; just before defining top_level.

Copyright © 2013 Synopsys, Inc.

Predefined Packages

The synthesis tool supports the two standard libraries, std and ieee, that contain packages containing commonly used definitions of data types, functions, and procedures. These libraries and their packages are built in to the synthesis tool, so you do not compile them. The predefined packages are described in the following table.

Library	Package	Description
std	standard	Defines the basic VHDL types including bit and bit_vector
ieee	std_logic_1164	Defines the 9-value std_logic and std_logic_vector types
ieee	numeric_bit	Defines numeric types and arithmetic functions. The base type is BIT.
ieee	numeric_std	Defines arithmetic operations on types defined in std_logic_1164
ieee	std_logic_arith	Defines the signed and unsigned types, and arithmetic operations for the signed and unsigned types
ieee	std_logic_signed	Defines signed arithmetic for std_logic and std_logic_vector
ieee	std_logic_unsigned	Defines unsigned arithmetic for std_logic and std_logic_vector

The synthesis tools also have vendor-specific built-in macro libraries for components like gates, counters, flip-flops, and I/Os. The libraries are located in *installDirectory*/lib/*vendorName*. Use the built-in macro libraries to instantiate vendor macros directly into the VHDL designs and forward-annotate them to the output netlist. Refer to the appropriate vendor support chapter for more information.

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

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

If you want the addition operator (+) to take two std_ulogic or std_ulogic_vector as inputs , you need the function defined in the std_logic_arith package (the cdn_arith.vhd file in *installDirectory*/lib/vhd/). Add this file to the project. To successfully compile, the VHDL file that uses the addition operator on these input types must have include the following statement:

```
use work.std_logic_arith.all;
```

Accessing Packages

To gain access to a package include a library clause in your VHDL source code to specify the library the package is contained in, and a use clause to specify the name of the package. The library and use clauses must be included immediately before the design unit (entity or architecture) that uses the package definitions.

Syntax

library *library_name* **;**
use *library_name*.*package_name*.**all ;**

To access the data types, functions and procedures declared in std_logic_1164, std_logic_arith, std_logic_signed, or std_logic_unsigned, you need a library ieee clause and a use clause for each of the packages you want to use.

Example

```
library ieee;
use ieee.std_logic_1164.all ;
use ieee.std_logic_signed.all ;

-- Other code
```

Library and Package Rules

To access the standard package, no library or use clause is required. The standard package is predefined (built-in) in VHDL, and contains the basic data types of bit, bit_vector, Boolean, integer, real, character, string, and others along with the operators and functions that work on them.

Copyright © 2013 Synopsys, Inc.

If you create your own package and compile it into the work library to access its definitions, you still need a use clause before the entity using them, but not a library clause (because work is the default library.)

To access packages other than those in work and std, you must provide a library and use clause for each package as shown in the following example of creating a resource library.

```
-- Compile this in library mylib
library ieee;
use ieee.std_logic_1164.all;

package my_constants is
constant max: std_logic_vector(3 downto 0):="1111";
   .
   .
   .
end package;

-- Compile this in library work
library ieee, mylib;
use ieee.std_logic_1164.all;
use mylib.my_constants.all;

entity compare is
   port (a: in std_logic_vector(3 downto 0);
         z: out std_logic );
end compare;

architecture rtl of compare is
begin
   z <= '1' when (a = max) else '0';
end rtl;
```

The rising_edge and falling_edge functions are defined in the std_logic_1164 package. If you use these functions, your clock signal must be declared as type std_logic.

Instantiating Components in a Design

No library or use clause is required to instantiate components (entities and their associated architectures) compiled in the default work library. The files containing the components must be listed in the source files list before any files that instantiate them.

To instantiate components from the built-in technology-vendor macro libraries, you must include the appropriate use and library clauses in your source code. Refer to the section for your vendor for more information.

To create a separate resource library to hold your components, put all the entities and architectures in one source file, and assign that source file the library components in the synthesis tool Project view. To access the components from your source code, put the clause library components; before the designs that instantiate them. There is no need for a use clause. The project file (`prj`) must include both the files that create the package components and the source file that accesses them.

Operators

The synthesis tool supports creating expressions using all predefined VHDL operators:

Arithmetic Operator	Description
-	addition
-	subtraction
*	multiplication
/	division
**	exponentiation (supported for compile-time constants and when left operand is 2; right operand can be a variable)
mod	modulus
rem	remainder

Copyright © 2013 Synopsys, Inc.

Relational Operator	Description
=	equal (if either operand has a bit with an 'X' or 'Z' value, the result is 'X')
/=	not equal (if either operand has a bit with an 'X' or 'Z' value, the result is 'X')
<	less than (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value 'X')
<=	less than or equal to (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value 'X')
>	greater than (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value 'X')
>=	greater than or equal to (if, because of unknown bits in the operands, the relation is ambiguous, then the result is the unknown value 'X')

Logical Operator	Description
and	and
or	or
nand	nand
nor	nor
xor	xor
xnor	xnor
not	not (takes only one operand)

Copyright © 2013 Synopsys, Inc.

Shift Operator	Description
sll	shift left logical – logically shifted left by R index positions
srl	shift right logical – logically shifted right by R index positions
sla	shift left arithmetic – arithmetically shifted left by R index positions
sra	shift right arithmetic – arithmetically shifted right by R index positions
rol	rotate left logical – rotated left by R index positions
ror	rotate right logical – rotated right by R index positions

Misc. Operator	Description
-	identity
-	negation
&	concatenation

Note: Initially, X's are treated as “don’t-cares”, but they are eventually converted to 0's or 1's in a way that minimizes hardware.

Large Time Resolution

The support of predefined physical time types includes the expanded range from –2147483647 to +2147483647 with units ranging from femtoseconds, and secondary units ranging up to an hour. Predefined physical time types allow selection of a wide number range representative of time type.

Copyright © 2013 Synopsys, Inc.

Example 1 – Using Large Time Values in Comparisons

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity test is
   generic (INTERVAL1 : time := 1000 fs ;
            INTERVAL2 : time := 1 ps;
            INTERVAL3 : time := 1000 ps;
            INTERVAL4 : time := 1 ns
   );

   port (a : in std_logic_vector(3 downto 0);
         b : in std_logic_vector(3 downto 0);
         c : out std_logic_vector(3 downto 0);
         d : out std_logic_vector(3 downto 0)
   );
end test;

architecture RTL of test is
begin
   c <= (a and b) when (INTERVAL1 = INTERVAL2) else
      (a or b);
   d <= (a xor b) when (INTERVAL3 /= INTERVAL4) else
      (a nand b);
end RTL;
```

Example 2 – Using Large Time Values in Constant Calculations

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
   generic (Interval : time := 20 ns;
            CLK_PERIOD : time := 8 ns );
   port (en : in std_logic;
         a : in std_logic_vector(10 downto 0);
         b : in std_logic_vector(10 downto 0);
         a_in : in std_logic_vector(7 downto 0);
         b_in : in std_logic_vector(7 downto 0);
         dummyOut : out std_logic_vector(7 downto 0);
         out1 : out std_logic_vector(10 downto 0) );
end entity;
```

Copyright © 2013 Synopsys, Inc.

```
architecture behv of test is
   constant my_time : positive := (Interval / 2 ns);
   constant CLK_PERIOD_PS : real := real(CLK_PERIOD / 1 ns);
   constant RESULT : positive := integer(CLK_PERIOD_PS);
   signal dummy : std_logic_vector (RESULT-1 downto 0);
   signal temp : std_logic_vector (my_time downto 0);
begin
   process (a, b)
   begin
      temp <= a and b;
      out1 <= temp;
   end process;
dummy <= (others => '0') when en = '1' else
   (a_in or b_in);
dummyOut <= dummy;
end behv;
```

Example 3 – Using Large Time Values in Generic Calculations

```
library IEEE;
use IEEE.std_logic_1164.all;
library unisim;
use unisim.vcomponents.all;

entity test is
   generic (clk_period : time := 6 ns);
   port (rst_in : in std_logic;
         in1 : in std_logic;
         CLK_PAD : in std_logic;
         RST_DLL : in std_logic;
         dout : out std_logic;
         CLK_out : out std_logic;
         LOCKED : out std_logic );
end test;

architecture STRUCT of test is
   signal CLK, CLK_int, CLK_dcm : std_logic;
   signal clk_dv : std_logic;
   constant clk_prd : real := real(clk_period / 2.0 ns);
begin
U1 : IBUFG port map (I => CLK_PAD, O => CLK_int);
U2 : DCM generic map
(CLKDV_DIVIDE  => clk_prd)
   port map (CLKFB => CLK,
            CLKIN => CLK_int,
            CLKDV => clk_dv,
            DSSEN => '0',
            PSCLK => '0',
```


Copyright © 2013 Synopsys, Inc.

```
            PSEN => '0',
            PSINCDEC => '0',
            RST => RST_DLL,
            CLK0 => CLK_dcm,
            LOCKED => LOCKED );
U3 : BUFG port map (I => CLK_dcm, O => CLK);
CLK_out <= CLK;
   process (clk_dv , rst_in, in1)
   begin
      if (rst_in = '1') then
         dout <= '0';
      elsif (clk_dv'event and clk_dv = '1') then
         dout <= in1;
      end if;
   end process;
end architecture STRUCT;
```

VHDL Process

The VHDL keyword process introduces a block of logic that is triggered to execute when one or more signals change value. Use processes to model combinational and sequential logic.

process Template to Model Combinational Logic

```
<optional_label> : process (<sensitivity_list>)

-- Declare local variables, data types,
-- and other local declarations here

begin
   -- Sequential statements go here, including:
   --    signal and variable assignments
   --    if and case statements
   --    while and for loops
   --    function and procedure calls
end process  <optional_label>;
```

Sensitivity List

The sensitivity list specifies the signal transitions that trigger the process to execute. This is analogous to specifying the inputs to logic on a schematic by drawing wires to gate inputs. If there is more than one signal, separate the names with commas.

A warning is issued when a signal is not in the sensitivity list but is used in the process, or when the signal is in the sensitivity list but not used by the process.

Syntax

process (*signal1***,** *signal2***,** ...**) ;**

A process can have only one sensitivity list, located immediately after the keyword process, or one or more wait statements. If there are one or more wait statements, one of these wait statements must be either the first or last statement in the process.

List all signals feeding into the combinational logic (all signals that affect signals assigned inside the process) in the sensitivity list. If you forget to list all signals, the synthesis tool generates the desired hardware, and reports a warning message that you are not triggering the process every time the hardware is changing its outputs, and therefore your pre- and post-synthesis simulation results might not match.

Any signals assigned in the process must either be outputs specified in the port list of the entity or declared as signals in the architecture declarative area.

Any variables assigned in the process are local and must be declared in the process declarative area.

Copyright © 2013 Synopsys, Inc.

Note: Make sure all signals assigned in a combinational process are explicitly assigned values each time the process executes. Otherwise, the synthesis tool must insert level-sensitive latches in your design, in order to hold the last value for the paths that don't assign values (if, for example, you have combinational loops in your design). This usually represents coding error, so the synthesis tool issues a warning message that level-sensitive latches are being inserted into the design because of combinational loops. You will get an error message if you have combinational loops in your design that are not recognized as level-sensitive latches.

Common Sequential Statements

This section describes the if-then-else and case statements.

if-then-else Statement

Syntax

if *condition1* **then**
  sequential_statement(s)
[**elsif** *condition2* **then**
  sequential_statement(s)]
[**else**
  sequential_statement(s)]
end if ;

The else and elsif clauses are optional.

Example

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
   port (output_signal : out std_logic;
         a, b, sel : in std_logic );
end mux;
```

```
architecture if_mux of mux is
begin
   process (sel, a, b)
   begin
      if sel = '1' then
         output_signal <= a;
      elsif sel = '0' then
         output_signal <= b;
      else
         output_signal <= 'X';
      end if;
   end process ;
end if_mux;
```

case Statement

Syntax

case *expression* **is**
  when *choice1* **=>** *sequential_statement(s)*
  when *choice2* **=>** *sequential _statement(s)*

-- *Other case choices*

  when *choiceN* **=>** *sequential_statement(s)*
end case ;

Note: VHDL requires that the expression match one of the given choices. To create a default, have the final choice be when others => sequential_statement(s) or null. (Null means not to do anything.)

Example

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
   port (output_signal : out std_logic;
         a, b, sel : in std_logic );
end mux;
```


Copyright © 2013 Synopsys, Inc.

```
architecture case_mux of mux is
begin
   process (sel, a, b)
   begin
      case sel is
         when '1' =>
            output_signal <= a;
         when '0' =>
            output_signal <= b;
         when others =>
            output_signal <= 'X';
      end case;
   end process;
end case_mux;
```

Note: To test the condition of matching a bit vector, such as `"0-11"`, that contains one or more don't-care bits, do *not* use the equality relational operator (=). Instead, use the std_match function (in the ieee.numeric_std package), which succeeds (evaluates to true) whenever all of the explicit constant bits (0 or 1) of the vector are matched, regardless of the values of the bits in the don't-care (-) positions. For example, use the condition `std_match(a, "0-11")` to test for a vector with the first bit unset (0) and the third and fourth bits set (1).

Concurrent Signal Assignments

There are three types of concurrent signal assignments in VHDL.

- Simple
- Selected (with-select-when)
- Conditional (when-else)

Use the concurrent signal assignment to model combinational logic. Put the concurrent signal assignment in the architecture body. You can any number of statements to describe your hardware implementation. Because all statements are concurrently active, the order you place them in the architecture body is not significant.

Copyright © 2013 Synopsys, Inc.

Re-evaluation of Signal Assignments

Every time any signal on the right side of the assignment operator (**<=**) changes value (including signals used in the expressions, values, choices, or conditions), the assignment statement is re-evaluated, and the result is assigned to the signal on the left side of the assignment operator. You can use any of the predefined operators to create the assigned value.

Simple Signal Assignments

Syntax

signal **<=** *expression* **;**

Example

```
architecture simple_example of simple is
begin
   c <= a nand b ;
end simple_example;
```

Selected Signal Assignments

Syntax

with *expression* **select**
signal **<=** *value1* **when** *choice1* **,**
value2 **when** *choice2* **,**
.
.
.
valueN **when** *choiceN* **;**

Example

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
   port (output_signal : out std_logic;
         a, b, sel : in std_logic );
end mux;
```


Copyright © 2013 Synopsys, Inc.

```
architecture with_select_when of mux is
begin
   with_sel_select
   output_signal <= a when '1',
      b when '0',
      'X' when others;
end with_select_when;
```

Conditional Signal Assignments

Syntax

signal **<=** *value1* **when** *condition1* **else**
value2 **when** *condition2* **else**
valueN-1 **when** *conditionN-1* **else**
valueN **;**

Example

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
   port (output_signal: out std_logic;
         a, b, sel: in std_logic );
end mux;

architecture when_else_mux of mux is
begin
output_signal <= a when sel = '1' else
   b when sel = '0' else
   'X';
end when_else_mux;
```

Note: To test the condition of matching a bit vector, such as `"0-11"`, that contains one or more don't-care bits, do *not* use the equality relational operator (=). Instead, use the std_match function (in the ieee.numeric_std package), which succeeds (evaluates to true) whenever all of the explicit constant bits (0 or 1) of the vector are matched, regardless of the values of the bits in the don't-care (-) positions. For example, use the condition `std_match(a, "0-11")` to test for a vector with the first bit unset (0) and the third and fourth bits set (1).

Resource Sharing

When you have mutually exclusive operators in a case statement, the synthesis tool shares resources for the operators in the case statements. For example, automatic sharing of operator resources includes adders, subtractors, incrementors, decrementors, and multipliers.

Combinational Logic

Combinational logic is hardware with output values based on some function of the current input values. There is no clock and no saved states. Most hardware is a mixture of combinational and sequential logic.

Create combinational logic with concurrent signal assignments and/or processes.

Sequential Logic

Sequential logic is hardware that has an internal state or memory. The state elements are either flip-flops that update on the active edge of a clock signal, or level-sensitive latches, that update during the active level of a clock signal.

Because of the internal state, the output values might depend not only on the current input values, but also on input values at previous times. State machines are made of sequential logic where the updated state values

Copyright © 2013 Synopsys, Inc.

depend on the previous state values. There are standard ways of modeling state machines in VHDL. Most hardware is a mixture of combinational and sequential logic.

Create sequential logic with processes and/or concurrent signal assignments.

Component Instantiation in VHDL

A structural description of a design is made up of component instantiations that describe the subsystems of the design and their signal interconnects. The synthesis tool supports four major methods of component instantiation:

- Simple component instantiation (described below)
- Selected component instantiation
- Direct entity instantiation
- Configurations described in *Configuration Specification*, on page 910

Simple Component Instantiation

In this method, a component is first declared either in the declaration region of the architecture, or in a package of (typically) component declarations, and then instantiated in the statement region of the architecture. By default, the synthesis process binds a named entity (and architecture) in the working library to all component instances that specify a component declaration with the same name.

Syntax

label **:** [**component**] *declaration_name*
[**generic map (***actual_generic1***,** *actual_generic2***,** ... **)**]
[**port map (** *port1***,** *port2***,** ... **)**] **;**

The use of the reserved word component is optional in component instantiations.

Example: VHDL 1987

```
architecture struct of hier_add is
component add
   generic (size : natural);
   port (a : in bit_vector(3 downto 0);
         b : in bit_vector(3 downto 0);
         result : out bit_vector(3 downto 0) );
end component;

begin
-- Simple component instantiation
add1: add
   generic map(size => 4)
   port map(a => ain,
      b => bin,
      result => q);

-- Other code
```

Example: VHDL 1993

```
architecture struct of hier_add is
component add
   generic (size : natural);
   port (a : in bit_vector(3 downto 0);
         b : in bit_vector(3 downto 0);
         result : out bit_vector(3 downto 0) );
end component;

begin
-- Simple component instantiation
add1: component add -- Component keyword new in 1993
   generic map(size => 4)
   port map(a => ain,
      b => bin,
      result => q);

-- Other code
```

Note: If no entity is found in the working library named the same as the declared component, all instances of the declared component are mapped to a black box and the error message “Unbound component mapped to black box” is issued.

Copyright © 2013 Synopsys, Inc.

VHDL Selected Name Support

Selected Name Support (SNS) is provided in VHDL for constants, operators, and functions in library packages. SNS eliminates ambiguity in a design referencing elements with the same names, but that have unique functionality when the design uses the elements with the same name defined in multiple packages. By specifying the library, package, and specific element (constant, operator, or function), SNS designates the specific constant, operator, or function used. This section discusses all facets of SNS. Complete VHDL examples are included to assist you in understanding how to use SNS effectively.

Constants

SNS lets you designate the constant to use from multiple library packages. To incorporate a constant into a design, specify the library, package, and constant. Using this feature eliminates ambiguity when multiple library packages have identical names for constants and are used in an entity-architecture pair.

The following example has two library packages available to the design constants. Each library package has a constant defined by the name C1 and each has a different value. SNS is used to specify the constant (see work.PACKAGE.C1 in the constants example below).

```
-- CONSTANTS PACKAGE1
library IEEE;
use IEEE.std_logic_1164.all;
package PACKAGE1 is
   constant Cl: std_logic_vector := "10001010";
end PACKAGE1;

-- CONSTANTS PACKAGE2
library IEEE;
use IEEE.std_logic_1164.all;
package PACKAGE2 is
   constant C1: std_logic_vector := "10110110";
end PACKAGE2;

-- CONSTANTS EXAMPLE
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
entity CONSTANTS is
   generic (num_bits : integer := 8) ;
      port (a,b: in std_logic_vector (num_bits -1 downto 0);
            out1, out2: out std_logic_vector (num_bits -1 downto 0)
            );
end CONSTANTS;

architecture RTL of CONSTANTS is
begin
   out1 <= a - work.PACKAGE1.Cl; -Example of specifying SNS
   out2 <= b - work.PACKAGE2.C1; -Example of specifying SNS
end RTL;
```

In the above design, outputs out1 and out2 use two C1 constants from two different packages. Although each output uses a constant named C1, the constants are not equivalent. For out1, the constant C1 is from PACKAGE1. For out2, the constant C1 is from PACKAGE2. For example:

`out1 <= a - work.PACKAGE1.Cl;` is equivalent to `out1 <= a - "10001010";`

whereas:

`out2 <= b - work.PACKAGE2.Cl;` is equivalent to `out2 <= b - "10110110";`

The constants have different values in different packages. SNS specifies the package and eliminates ambiguity within the design.

Functions and Operators

Functions and operators in VHDL library packages customarily have overlapping naming conventions. For example, the add operator in the IEEE standard library exists in both the std_logic_signed and std_logic_unsigned packages. Depending upon the add operator used, different values result. Defining only one of the IEEE library packages is a straightforward solution to eliminate ambiguity, but applying this solution is not always possible. A design requiring both std_logic_signed and std_logic_unsigned package elements must use SNS to eliminate ambiguity.

Functions

In the following example, multiple IEEE packages are declared in a 256x8 RAM design. Both std_logic_signed and std_logic_unsigned packages are included. In the RAM definition, the signal address_in is converted from type std_logic_vector to type integer using the CONV_INTEGER function, but which

Copyright © 2013 Synopsys, Inc.

CONV_INTEGER function will be called? SNS determines the function to use. The RAM definition clearly declares the std_logic_unsigned package as the source for the CONV_INTEGER function.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

entity FUNCTIONS is
   port (address : in std_logic_vector(7 downto 0);
         data_in : in std_logic_vector(7 downto 0);
         data_out : out std_logic_vector(7 downto 0);
         we : in std_logic;
         clk : in std_logic );

end FUNCTIONS;

architecture RTL of FUNCTIONS is
type mem_type is array (255 downto 0) of
   std_logic_vector (7 downto 0);
signal mem: mem_type;
signal address_in: std_logic_vector(7 downto 0);
begin
data_out <= mem(IEEE.std_logic_unsigned.CONV_INTEGER(address_in));
   process (clk)
   begin
      if rising_edge(clk) then
         if (we = '1') then
            mem(IEEE.std_logic_unsigned.CONV_INTEGER(address_in))
               <= data_in;
         end if;
      address_in <= address;
      end if;
   end process;
end RTL;
```

Operators

In this example, comparator functions from the IEEE std_logic_signed and std_logic_unsigned library packages are used. Depending upon the comparator called, a signed or an unsigned comparison results. In the assigned outputs below, the op1 and op2 functions show the valid SNS syntax for operator implementation.

```
library IEEE;
use IEEE.std_logic_1164.std_logic_vector;
use IEEE.std_logic_signed.">";
use IEEE.std_logic_unsigned.">";

entity OPERATORS is
   port (in1 :std_logic_vector(1 to 4);
         in2 :std_logic_vector(1 to 4);
         in3 :std_logic_vector(1 to 4);
         in4 :std_logic_vector(1 to 4);
         op1,op2 :out boolean );
end OPERATORS;

architecture RTL of OPERATORS is
begin
   process(in1,in2,in3,in4)
   begin

      --Example of specifying SNS
      op1 <= IEEE.std_logic_signed.">"(in1,in2);

      --Example of specifying SNS
      op2 <= IEEE.std_logic_unsigned.">"(in3,in4);
   end process;
end RTL;
```

User-defined Function Support

SNS is not limited to predefined standard IEEE packages and packages supported by the synthesis tool; SNS also supports user-defined packages. You can create library packages that access constants, operators, and functions in the same manner as the packages supported by IEEE or the synthesis tool.

The following example incorporates two user-defined packages in the design. Each package includes a function named func. In PACKAGE1, func is an XOR gate, whereas in PACKAGE2, func is an AND gate. Depending on the package called, func results in either an XOR or an AND gate. The function call uses SNS to distinguish the function that is called.

Copyright © 2013 Synopsys, Inc.

```
-- USER DEFINED PACKAGE1
library IEEE;
use IEEE.std_logic_1164.all;
package PACKAGE1 is
   function func(a,b: in std_logic) return std_logic;
end PACKAGE1;

package body PACKAGE1 is
   function func(a,b: in std_logic) return std_logic is
begin
   return(a xor b);
end func;
end PACKAGE1;

-- USER DEFINED PACKAGE2
library IEEE;
use IEEE.std_logic_1164.all;

package PACKAGE2 is
   function func(a,b: in std_logic) return std_logic;
end PACKAGE2;

package body PACKAGE2 is
   function func(a,b: in std_logic) return std_logic is
begin
   return(a and b);
end func;
end PACKAGE2;

-- USER DEFINED FUNCTION EXAMPLE
library IEEE;
use IEEE.std_logic_1164.all;

entity USER_DEFINED_FUNCTION is
   port (in0: in std_logic;
         in1: in std_logic;
         out0: out std_logic;
         out1: out std_logic );
end USER_DEFINED_FUNCTION;

architecture RTL of USER_DEFINED_FUNCTION is
begin
   out0 <= work.PACKAGE1.func(in0, in1);
   out1 <= work.PACKAGE2.func(in0, in1);
end RTL;
```

Demand Loading

In the previous section, the user-defined function example successfully uses SNS to determine the func function to implement. However, neither PACKAGE1 nor PACKAGE2 was declared as a use package clause (for example, work.PACKAGE1.all;). How could func have been executed without a use package declaration? A feature of SNS is demand loading: this loads the necessary package without explicit use declarations. Demand loading lets you create designs using SNS without use package declarations, which supports all necessary constants, operators, and functions.

Copyright © 2013 Synopsys, Inc.

VHDL Implicit Data-type Defaults

Type default propagation avoids potential simulation mismatches that are the result of differences in behavior with how initial values for registers are treated in the synthesis tools and how they are treated in the simulation tools.

With implicit data-type defaults, when there is no explicit initial-value declaration for a signal being registered, the VHDL compiler passes an init value through a syn_init property to the mapper, and the mapper then propagates the value to the respective register. Compiler requirements are based on specific data types. These requirements can be broadly grouped based on the different data types available in the VHDL language.

Implicit data-type defaults are enabled on the VHDL panel of the Implementation Options dialog box or through a -supporttypedflt argument to a set_option command.

To illustrate the use of implicit data-type defaults, consider the following example.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
   port (clk:in std_logic;
         a : in integer range 1 to 8;
         b : out integer range 1 to 8;
         d : out positive range 1 to 7 );
end entity top;
```

```
architecture rtl of top is
signal a1,a2 : integer range 1 to 8 ;
signal a3,a4 : positive range 1 to 7;
begin
a1 <= a ;
a3 <= a ;
b <= a2 ;
d <= a4 ;
   process(clk)
   begin
      if (rising_edge(clk))then
         a2 <= a1 ;
         a4 <= a3 ;
      end if;
   end process;
end rtl;
```

In the above example, two signals (a2 and a4) with different type default values are registered. Without implicit data-type defaults, if the values of the signals being registered are not the same, the compiler merges the redundant logic into a single register as shown in the figure below.

Enabling implicit data-type defaults prevents certain compiler and mapper optimizations to preserve both registers as shown in the following figure.

Copyright © 2013 Synopsys, Inc.

Example – Impact on Integer Ranges

The default value for the integer type when a range is specified is the minumum value of the range specified, and size is the upper limit of that range. With implicit data-type defaults, the compiler is required to propagate the minimum value of the range as the init value to the mapper. Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
   port (clk,set:in std_logic;
         a : in integer range -6 to 8;
         b : out integer range -6 to 8 );
end entity top;

architecture rtl of top is
signal a1,a2: integer range -6 to 8;
begin
a1 <= a ;
   process(clk,set)
   begin
      if (rising_edge(clk))then
         if set = '1' then
            a2 <= a;
         else
            a2 <= a1 ;
```

```
            end if;
         end if;
      end process;
   b <= a2;
   end rtl;
```

In the example,

```
signal a1, a2: integer range -6 to 8;
```

the default value is -6 (FA in 2's complement) and the range is -6 to 8. With a total of 15 values, the size of the range can be represented in four bits.

Example – Impact on RAM Inferencing

When inferencing a RAM with implicit data-type defaults, the compiler propagates the type default values as init values for each RAM location. The mapper must check if the block RAMs of the selected technology support initial values and then determine if the compiler-propagated init values are to be considered. If the mapper chooses to ignore the init values, a warning is issued stating that the init values are being ignored. Consider the following VHDL design:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity top is
   port (clk : in std_logic;
         addr : in std_logic_vector (6 downto 0);
         din : in positive;
         wen : in std_logic;
         dout : out positive );
end top;

architecture behavioral of top is
-- RAM
type tram is array(0 to 127) of  positive ;
signal ram : tram ;
begin
-- Contents of RAM has initial value = 1
   process (clk)
   begin
      if clk'event and clk = '1' then
         if wen = '1' then
            ram(conv_integer(addr)) <= din_sig;
```


Copyright © 2013 Synopsys, Inc.

```
            end if;
         dout <= ram(conv_integer(addr));
         end if;
      end process;
   end behavioral;
```

In the above example:

- The type of signal a1 is bit_vector
- The default value for type integer is 1 when no range is specified

Accordingly, a value of x00000001 is propagated by the compiler to the mapper with a syn_init property.

VHDL Synthesis Guidelines

This section provides guidelines for synthesis using VHDL. The following topics are covered:

- General Synthesis Guidelines, on page 879
- VHDL Language Guidelines, on page 880
- Model Template, on page 881
- Constraint Files for VHDL Designs, on page 882
- Creating Flip-flops and Registers Using VHDL Processes, on page 883
- Clock Edges, on page 884
- Defining an Event Outside a Process, on page 885
- Using a WAIT Statement Inside a Process, on page 886
- Level-sensitive Latches Using Concurrent Signal Assignments, on page 887
- Level-sensitive Latches Using VHDL Processes, on page 888
- Signed mod Support for Constant Operands, on page 891

General Synthesis Guidelines

Some general guidelines are presented here to help you synthesize your VHDL design.

- Top-level entity and architecture. The synthesis tool chooses the top-level entity and architecture – the last architecture for the last entity in the last file compiled. Entity selection can be overridden from the VHDL panel of the Implementation Options dialog box. Files are compiled in the order they appear – from top to bottom in the Project view source files list.
- Simulate your design before synthesis because it exposes logic errors. Logic errors that are not caught are passed through the synthesis tool, and the synthesized results will contain the same logic errors.

Copyright © 2013 Synopsys, Inc.

- Simulate your design after placement and routing. Have the place-and-route tool generate a post placement and routing (timing-accurate) simulation netlist, and do a final simulation before programming your devices.
- Avoid asynchronous state machines. To use the synthesis tool for asynchronous state machines, make a netlist of technology primitives from your target library.
- For modeling level-sensitive latches, it is simplest to use concurrent signal assignments.

VHDL Language Guidelines

This section discusses VHDL language guidelines.

Processes

- A process must have either a sensitivity list or one wait statement.
- Each sequential process can be triggered from exactly one clock and only one edge of clock (and optional sets and resets).
- Avoid combinational loops in processes. Make sure all signals assigned in a combinational process are explicitly assigned values every time the process executes; otherwise, the synthesis tool needs to insert level-sensitive latches in your design to hold the last value for the paths that do not assign values. This might represent a mistake on your part, so the synthesis tool issues a warning message that level-sensitive latches are being inserted into your design. You will get an warning message if you have combinational loops in your design that are not recognized as level-sensitive latches (for example, if you have an asynchronous state machine).

Assignments

- Assigning an 'X' or '-' to a signal is interpreted as a “don't care”, so the synthesis tool creates the hardware that is the most efficient design.

Copyright © 2013 Synopsys, Inc.

Data Types

- Integers are 32-bit quantities. If you declare a port as an integer data type, specify a range (for example, my_input: in integer range 0 to 7). Otherwise, your synthesis result file will contain a 32-bit port.
- Enumeration types are represented as a vector of bits. The encoding can be sequential, gray, or one hot. You can manually choose the encoding for ports with an enumeration type.

Model Template

You can place any number of concurrent statements (signal assignments, processes, component instantiations, and generate statements) in your architecture body as shown in the following example. The order of these statements within the architecture is not significant, as all can execute concurrently.

- The statements between the begin and the end in a process execute sequentially, in the order you type them from top to bottom.
- You can add comments in VHDL by proceeding your comment text with two dashes "--". Any text from the dashes to the end of the line is treated as a comment, and ignored by the synthesis tool.

```
-- List libraries/packages that contain definitions you use
library <library_name> ;
use <library_name>.<package_name>.all ;

-- The entity describes the interface for your design.
entity <entity_name> is
   generic ( <define_interface_constants_here> ) ;
      port ( <port_list_information_goes_here> ) ;
end <entity_name> ;

-- The architecture describes the functionality (implementation)
-- of your design
architecture <architecture_name> of <entity_name> is

-- Architecture declaration region.
-- Declare internal signals, data types, and subprograms here
```


Copyright © 2013 Synopsys, Inc.

```
-- If you will create hierarchy by instantiating a
-- component (which is just another architecture), then
-- declare its interface here with a component declaration;
component <entity_name_instantiated_below>
   port (<port_list_information_as_defined_in_the_entity>) ;
end component ;

begin -- Architecture body, describes functionality

-- Use concurrent statements here to describe the functionality
-- of your design. The most common concurrent statements are the
-- concurrent signal assignment, process, and component
-- instantiation.

-- Concurrent signal assignment (simple form):
<result_signal_name> <= <expression> ;

-- Process:
process <sensitivity list>)
-- Declare local variables, data types,
-- and other local declarations here
begin
-- Sequential statements go here, including:
   -- signal and variable assignments
   -- if and case statements
   -- while and for loops
   -- function and procedure calls
end process;

-- Component instantiation
<instance_name> : <entity_name>
   generic map (<override values here >)
   port map (<port list>) ;
end <architecture_name> ;
```

Constraint Files for VHDL Designs

In previous versions of the software, all object names output by the compiler were converted to lower case. This means that any constraints files created by dragging from the RTL view or through the SCOPE UI contained object names using only lower case. Case is preserved on design object names. If you use mixed-case names in your VHDL source, for constraints to be applied correctly, you must manually update any older constraint files or re-create constraints in the SCOPE editor.

Creating Flip-flops and Registers Using VHDL Processes

It is easy to create flip-flops and registers using a process in your VHDL design.

process Template

```
process (<sensitivity list>)
begin
   <sequential statement(s)>
end;
```

To create a flip-flop:

1. List your clock signal in the sensitivity list. Recall that if the value of any signal listed in the sensitivity list changes, the process is triggered, and executes. For example,

   ```
   process (clk)
   ```

2. Check for rising_edge or falling_edge as the first statement inside the process. For example,

   ```
   process (clk)
   begin
      if rising_edge(clk) then
         <sequential statement(s)>
   ```

   or

   ```
   process (clk)
   begin
      if falling_edge(clk) then
         <sequential statement(s)>
   ```

   Alternatively, you could use an if clk'event and clk = '1' then statement to test for a rising edge (or if clk'event and clk = '0' then for a falling edge). Although these statements work, for clarity and consistency, use the rising_edge and falling_edge functions from the VHDL 1993 standard.

3. Set your flip-flop output to a value, with no delay, if the clock edge occurred. For example, q <= d ;.

Copyright © 2013 Synopsys, Inc.

Complete Example

```
library ieee;
use ieee.std_logic_1164.all;

entity dff_or is
   port (a, b, clk: in std_logic;
         q: out std_logic );
end dff_or;

architecture sensitivity_list of dff_or is
begin
   process (clk) -- Clock name is in sensitivity list
   begin
      if rising_edge(clk) then
         q <= a or b;
      end if;
   end process;
end sensitivity_list ;
```

In this example, if clk has an event on it, the process is triggered and starts executing. The first statement (the if statement) then checks to see if a rising edge has occurred for clk. If the if statement evaluates to true, there was a rising edge on clk and the q output is set to the value of a or b. If the clk changes from 1 to 0, the process is triggered and the if statement executes, but it evaluates to false and the q output is not updated. This is the functionality of a flip-flop, and synthesis correctly recognizes it as such and connects the result of the a or b expression to the data input of a D-type flip-flop and the q signal to the q output of the flip-flop.

Note: The signals you set inside the process will drive the data inputs of D-type flip-flops.

Clock Edges

There are many ways to correctly represent clock edges within a process including those shown here.

The typical rising clock edge representation is:

```
rising_edge(clk)
```

Other supported rising clock edge representations are:

```
clk = '1' and clk'event
clk'last_value = '0' and clk'event
clk'event and clk /= '0'
```

The typical falling clock edge representation is:

```
falling_edge(clk)
```

Other supported falling clock edge representations are:

```
clk = '0' and clk'event
clk'last_value = '1' and clk'event
clk'event and clk /= '1'
```

Incorrect or Unsupported Representations for Clock Edges

Rising clock edge:

```
clk = '1'
clk and clk'event -- Because clk is not a Boolean
```

Falling clock edge:

```
clk = '0'
not clk and clk'event -- Because clk is not a Boolean
```

Defining an Event Outside a Process

The 'event attribute can be used outside of a process block. For example, the process block

```
process (clk,d)
begin
   if (clk='1' and clk'event) then
      q <= d;
   end if;
end process;
```

can be replaced by including the following line outside of the process statement:

```
q <= d when (clk='1' and clk'event);
```


Copyright © 2013 Synopsys, Inc.

Using a WAIT Statement Inside a Process

The synthesis tool supports a wait statement inside a process to create flip-flops, instead of using a sensitivity list.

Example

```
library ieee;
use ieee.std_logic_1164.all;

entity dff_or is
   port (a, b, clk: in std_logic;
         q: out std_logic );
end dff_or;

architecture wait_statement of dff_or is
begin
   process -- Notice the absence of a sensitivity list.
   begin
-- The process waits here until the condition becomes true
      wait until rising_edge(clk);
         q <= a or b;
   end process;
end wait_statement;
```

Rules for Using wait Statements Inside a Process

- It is illegal in VHDL to have a process with a wait statement and a sensitivity list.
- The wait statement must either be the first or the last statement of the process.

Clock Edge Representation in wait Statements

The typical rising clock edge representation is:

```
wait until rising_edge(clk);
```

Other supported rising clock edge representations are:

```
wait until clk = '1' and clk'event
wait until clk'last_value = '0' and clk'event
wait until clk'event and clk /= '0'
```

Copyright © 2013 Synopsys, Inc.

The typical falling clock edge representation is:

```
wait until falling_edge(clk)
```

Other supported falling clock edge representations are:

```
wait until clk = '0' and clk'event
wait until clk'last_value = '1' and clk'event
wait until clk'event and clk /= '1'
```

Level-sensitive Latches Using Concurrent Signal Assignments

To model level-sensitive latches in VHDL, use a concurrent signal assignment statement with the conditional signal assignment form (also known as when-else).

Syntax

signal **<=** *value1* **when** *condition1* **else**
value2 **when** *condition2* **else**
valueN-1 **when** *conditionN-1* **else**
valueN **;**

Example

In VHDL, you are not allowed to read the value of ports of mode out inside of an architecture that it was declared for. Ports of mode buffer can be read from and written to, but must have no more than one driver for the port in the architecture. In the following port statement example, q is defined as mode buffer.

```
library ieee;
use ieee.std_logic_1164.all;

entity latchor1 is
   port (a, b, clk : in std_logic;
-- q has mode buffer so it can be read inside architecture
         q: buffer std_logic );
end latchor1;

architecture behave of latchor1 is
begin
   q <= a or b when clk = '1' else q;
end behave;
```


Copyright © 2013 Synopsys, Inc.

Whenever clk, a, or b changes, the expression on the right side re-evaluates. If clk becomes true (active, logic 1), the value of a or b is assigned to the q output. When the clk changes and becomes false (deactivated), q is assigned to q (holds the last value of q). If a or b changes, and clk is already active, the new value of a or b is assigned to q.

Level-sensitive Latches Using VHDL Processes

Although it is simpler to specify level-sensitive latches using concurrent signal assignment statements, you can create level-sensitive latches with VHDL processes. Follow the guidelines given here for the sensitivity list and assignments.

process Template

```
process (<sensitivity list>)
begin
   <sequential statement(s)>
end process;
```

Sensitivity List

The sensitivity list specifies the clock signal, and the signals that feed into the data input of the level-sensitive latch. The sensitivity list must be located immediately after the process keyword.

Syntax

process (*clock_name***,** *signal1***,** *signal2***,** ...**)**

Example

```
process (clk, data)
```

process Template for a Level-sensitive Latch

```
process (<clock, data_signals ... > ...)
begin
   if (<clock> = <active_value>)
      <signals> <= <expression involving data signals> ;
   end if;
end process ;
```

All data signals assigned in this manner become logic into data inputs of level-sensitive latches.

Whenever level-sensitive latches are generated from a process, the synthesis tool issues a warning message so that you can verify if level-sensitive latches are really what you intended. Often a thorough simulation of your architecture will reveal mistakes in coding style that can cause the creation of level-sensitive latches during synthesis.

Example: Creating Level-sensitive Latches that You Want

```
library ieee;
use ieee.std_logic_1164.all;

entity latchor2 is
   port (a, b, clk : in std_logic ;
         q: out std_logic );
end latchor2;

architecture behave of latchor2 is
begin
   process (clk, a, b)
   begin
      if clk = '1' then
         q <= a or b;
      end if;
   end process ;
end behave;
```

If there is an event (change in value) on either clk, a or b, and clk is a logic 1, set q to a or b.

What to do when clk is a logic 0 is not specified (there is no else), so when clk is a logic zero, the last value assigned is maintained (there is an implicit q=q). The synthesis tool correctly recognizes this as a level-sensitive latch, and creates a level-sensitive latch in your design. It will issue a warning message when you compile this architecture, but after examination, this warning message can safely be ignored.

Example: Creating Unwanted Level-sensitive Latches

This design demonstrates the level-sensitive latch warning caused by a missed assignment in the when two => case. The message generated is:

```
"Latch generated from process for signal odd, probably caused by a
missing assignment in an if or case statement".
```


Copyright © 2013 Synopsys, Inc.

This information will help you find a functional error even before simulation.

```
library ieee;
use ieee.std_logic_1164.all;

entity mistake is
   port (inp: in std_logic_vector (1 downto 0);
         outp: out std_logic_vector (3 downto 0);
         even, odd: out std_logic );
end mistake;

architecture behave of mistake is
   constant zero: std_logic_vector (1 downto 0):= "00";
   constant one: std_logic_vector (1 downto 0):= "01";
   constant two: std_logic_vector (1 downto 0):= "10";
   constant three: std_logic_vector (1 downto 0):= "11";
begin
   process (inp)
   begin
      case inp is
         when zero =>
            outp <= "0001";
            even <= '1';
            odd <= '0';
         when one =>
            outp <= "0010";
            even <= '0';
            odd <= '1';
         when two =>
            outp <= "0100";
            even <= '1';
-- Notice that assignment to odd is mistakenly commented out next.

            -- odd <= '0';
         when three =>
            outp <= "1000";
            even <= '0';
            odd <= '1';
      end case;
   end process;
end behave;
```

Signed mod Support for Constant Operands

The synthesis tool supports signed mod for constant operands. Additionally, division operators (/, rem, mod), where the operands are compile-time constants and greater than 32 bits, are supported.

Example of using signed mod operator with constant operands

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;
LIBRARY ieee; USE ieee.numeric_std.all;

ENTITY divmod IS
   PORT (tstvec: out  signed(7 DOWNTO 0) );
END divmod;

ARCHITECTURE structure OF divmod IS
   CONSTANT NOMINATOR   : signed(7 DOWNTO 0) := "10000001";
   CONSTANT DENOMINATOR : signed(7 DOWNTO 0) := "00000011";
   CONSTANT RESULT      : signed(7 DOWNTO 0) := NOMINATOR mod
      DENOMINATOR;
BEGIN
   tstvec <= result;

END ARCHITECTURE structure;
```

Example of a signed division with a constant right operand.

```
LIBRARY ieee ; USE ieee.std_logic_1164.ALL;
LIBRARY ieee ; USE ieee.numeric_std.all;

ENTITY divmod IS
   PORT (tstvec: out  signed(7 DOWNTO 0) );
END divmod;

ARCHITECTURE structure OF divmod IS
   CONSTANT NOMINATOR   : signed(7 DOWNTO 0) := "11111001";
   CONSTANT DENOMINATOR : signed(7 DOWNTO 0) := "00000011";
   CONSTANT RESULT      : signed(7 DOWNTO 0) := NOMINATOR /
      DENOMINATOR;
BEGIN
   tstvec <= result;

END ARCHITECTURE structure;
```


Copyright © 2013 Synopsys, Inc.

An example where the operands are greater than 32 bits

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;
LIBRARY ieee; USE ieee.numeric_std.all;

ENTITY divmod IS
   PORT (tstvec: out  unsigned(33 DOWNTO 0) );
END divmod;

ARCHITECTURE structure OF divmod IS
   CONSTANT NOMINATOR   : unsigned(33 DOWNTO 0) :=
   "100000000000000000000000000000000";
   CONSTANT DENOMINATOR : unsigned(32 DOWNTO 0) :=
   "000000000000000000000000000000011";
   CONSTANT RESULT      : unsigned(33 DOWNTO 0) := NOMINATOR /
      DENOMINATOR;
BEGIN
   tstvec <= result;
END ARCHITECTURE structure;
```

Copyright © 2013 Synopsys, Inc.

Sets and Resets

This section describes VHDL sets and resets, both asynchronous and synchronous. A set signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic one. A reset signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic zero.

The topics include:

- Asynchronous Sets and Resets, on page 893
- Synchronous Sets and Resets, on page 894

Asynchronous Sets and Resets

By definition, asynchronous sets and resets are independent of the clock and do not require an active clock edge. Therefore, you must include the set and reset signals in the sensitivity list of your process so they trigger the process to execute.

Sensitivity List

The sensitivity list is a list of signals (including ports) that, when there is an event (change in value), triggers the process to execute.

Syntax

process (*clk_name***,** *set_signal_name***,** *reset_signal_name* **)**

The signals are listed in any order, separated by commas.

Example: process Template with Asynchronous, Active-high reset, set

```
process (clk, reset, set)
begin
   if reset = '1' then
-- Reset the outputs to zero.
   elsif set = '1' then
-- Set the outputs to one.
   elsif rising_edge(clk) then -- Rising clock edge clock
-- Clocked logic goes here.
   end if;
end process;
```


Copyright © 2013 Synopsys, Inc.

Example: D Flip-flop with Asynchronous, Active-high reset, set

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
   port (data, clk, reset, set : in std_logic;
         qrs: out std_logic );
end dff1;

architecture async_set_reset of dff1 is
begin
   setreset: process(clk, reset, set)
   begin
      if reset = '1' then
         qrs <= '0';
      elsif set = '1' then
         qrs <= '1';
      elsif rising_edge(clk) then
         qrs <= data;
      end if;
   end process setreset;
end async_set_reset;
```

Synchronous Sets and Resets

Synchronous sets and resets set flip-flop outputs to logic '1' or '0' respectively on an active clock edge.

Do not list the set and reset signal names in the sensitivity list of a process so they will not trigger the process to execute upon changing. Instead, trigger the process when the clock signal changes, and check the reset and set as the first statements.

RTL View Primitives

The VHDL compiler can detect and extract the following flip-flops with synchronous sets and resets and display them in the RTL schematic view:

- sdffr – f lip-flop with synchronous reset
- sdffs – flip-flop with synchronous set
- sdffrs – flip-flop with both synchronous set and reset
- sdffpat – vectored flip-flop with synchronous set/reset pattern

- sdffre – enabled flip-flop with synchronous reset
- sdffse – enabled flip-flop with synchronous set
- sdffpate – enabled, vectored flip-flop with synchronous set/reset pattern

You can check the name (type) of any primitive by placing the mouse pointer over it in the RTL view: a tooltip displays the name.

Sensitivity List

The sensitivity list is a list of signals (including ports) that, when there is an event (change in value), triggers the process to execute.

Syntax

process (*clk_signal_name* **)**

Copyright © 2013 Synopsys, Inc.

Example: process Template with Synchronous, Active-high reset, set

```
process(clk)
begin
   if rising_edge(clk) then
      if reset = '1' then
         -- Set the outputs to '0'.
      elsif set = '1' then
         -- Set the outputs to '1'.
      else
         -- Clocked logic goes here.
      end if ;
   end if ;
end process;
```

Example: D Flip-flop with Synchronous, Active-high reset, set

```
library ieee;
use ieee.std_logic_1164.all;

entity dff2 is
   port (data, clk, reset, set : in std_logic;
         qrs: out std_logic );
end dff2;

architecture sync_set_reset of dff2 is
begin
   setreset: process (clk)
   begin
      if rising_edge(clk) then
         if reset = '1' then
            qrs <= '0';
         elsif set = '1' then
            qrs <= '1';
         else
            qrs <= data;
         end if;
      end if;
   end process setreset;
end sync_set_reset;
```

Copyright © 2013 Synopsys, Inc.

VHDL State Machines

This section describes VHDL state machines: guidelines for using them, defining state values with enumerated types, and dealing with asynchrony. The topics include:

- State Machine Guidelines, on page 897
- Using Enumerated Types for State Values, on page 901
- Simulation Tips When Using Enumerated Types, on page 902
- Asynchronous State Machines in VHDL, on page 904

State Machine Guidelines

A finite state machine (FSM) is hardware that advances from state to state at a clock edge.

The synthesis tool works best with synchronous state machines. You typically write a fully synchronous design, avoiding asynchronous paths such as paths through the asynchronous reset of a register. See *Asynchronous State Machines in VHDL*, on page 904 for information about asynchronous state machines.

The following are guidelines for coding FSMs:

- The state machine must have a synchronous or asynchronous reset, to be inferred. State machines must have an asynchronous or synchronous reset to set the hardware to a valid state after power-up, and to reset your hardware during operation (asynchronous resets are available freely in most FPGA architectures).
- The synthesis tool does not infer implicit state machines that are created using multiple wait statements in a process.
- Separate the sequential process statements from the combinational ones. Besides making it easier to read, it makes what is being registered very obvious. It also gives better control over the type of register element used.
- Represent states with defined labels or enumerated types.

Copyright © 2013 Synopsys, Inc.

- Use a case statement in a process to check the current state at the clock edge, advance to the next state, and set the output values. You can also use if-then-else statements.
- Assign default values to outputs derived from the FSM before the case statement. This helps prevent the generation of unwanted latches and makes it easier to read because there is less clutter from rarely used signals.
- If you do not have case statements for all possible combinations of the selector, use a when others assignment as the last assignment in your case statement and set the state vector to some valid state. If your state vector is not an enumerated type, set the value to X. Assign the state to X in the default clause of the case statement, to avoid mismatches between pre- and post-synthesis simulations. See *Example: Default Assignment*, on page 901.
- If a state machine defined in the code feeds sequential elements in a different clock domain, some encoding values can cause metastability. By default, the synthesis tools choose the optimal encoding value based on the number of states in the state machine. This can introduce additional decode logic that could cause metastability when it feeds sequential elements in a different clock domain. The prevent this instability, use syn_encoding = "original" to guide the synthesis tool for these cases.
- Override the default encoding style with the syn_encoding attribute. The default encoding is determined by the number of states, except in the case of Xilinx designs, where a non-default encoding is implemented if it produces better results. See *syn_encoding Values*, on page 1305 for a list of default and other encodings. When you specify a particular encoding style with syn_encoding, that value is used during the mapping stage to determine encoding style.

  ```
  attribute syn_encoding : string;
  attribute syn_encoding of <typename> : type is "sequential";
  ```

  See Chapter 16, *Attribute and Directive Syntax*, for details about the syntax and values.

  One-hot implementations are not always the best choice for state machines, even in FPGAs and CPLDs. For example, one-hot state machines might result in higher speeds in CPLDs, but could cause fitting problems because of the larger number of global signals. An example in an FPGA with ineffective one-hot implementation is a state machine that drives a large decoder, generating many output signals. In a 16-state state machine, for example, the output decoder logic might

Copyright © 2013 Synopsys, Inc.

reference sixteen signals in a one-hot implementation, but only four signals in an encoded representation.

In general, do not use the directive syn_enum_encoding to set the encoding style. Use syn_encoding instead. The value of syn_enum_encoding is used by the compiler to interpret the enumerated data types but is ignored by the mapper when the state machine is actually implemented.

The directive syn_enum_encoding affects the final circuit only when you have turned off the FSM Compiler. Therefore, if you are not using FSM Compiler or the syn_state_machine attribute, which use syn_encoding, you can use syn_enum_encoding to set the encoding style. See Chapter 16, *Attribute and Directive Syntax,* for details about the syntax and values.

- Implement user-defined enumeration encoding, beyond the one-hot, gray, and sequential styles. Use the directive syn_enum_encoding to set the state encoding. See *Example: FSM User-Defined Encoding,* on page 900.

Example: FSM Coding Style

```
architecture behave of test is
   type state_value is (deflt, idle, read, write);
   signal state, next_state: state_value;
begin
-- Figure out the next state
   process (clk, rst)
   begin
      if rst = '0' then
         state <= idle;
      elsif rising_edge(clk) then
         state <= next_state;
      end if;
   end process;

   process (state, enable, data_in)
   begin
      data_out <= '0';
      -- Catch missing assignments to next_state
      next_state <= idle;
      state0 <= '0';
      state1 <= '0';
      state2 <= '0';
      case state is
         when idle =>
            if enable = '1' then
```


Copyright © 2013 Synopsys, Inc.

```
                  state0 <= '1' ;data_out <= data_in(0);
                  next_state <= read;
               else next_state <= idle;
               end if;
            when read =>
               if enable = '1' then
                  state1 <= '1'; data_out <= data_in(1);
                  next_state <= write;
               else next_state <= read;
               end if;
            when deflt =>
               if enable = '1' then
                  state2 <= '1' ;data_out <= data_in(2);
                  next_state <= idle;
               else next_state <= write;
               end if;
            when others => next_state <= deflt;
         end case;
      end process;
   end behave;
```

Example: FSM User-Defined Encoding

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
   port (clk, rst : bit;
         O : out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
   process (clk, rst)
   begin
      if rst = '1' then
         machine <= S0;
      elsif clk = '1' and clk'event then
         case machine is
            when S0 => machine <= S1;
```

```
            when S1 => machine <= S2;
            when S2 => machine <= S0;
         end case;
      end if;
   end process;

   with machine select
      O <= "001" when S0,
      "010" when S1,
      "101" when S2;
end behave;
```

Example: Default Assignment

The second others keyword in the following example pads (covers) all the bits. In this way, you need not remember the exact number of X's needed for the state variable or output signal.

```
when others =>
   state := (others => 'X') ;
```

Assigning X to the state variable (a "don't care" for synthesis) tells the synthesis tool that you have specified all the used states in your case statement, and any unnecessary decoding and gates related to other cases can therefore be removed. You do not have to add any special, non-VHDL directives.

If you set the state to a used state for the when others case (for example: when others => state <= delft), the synthesis tool generates the same logic as if you assign X, but there will be pre- and post-synthesis simulation mismatches until you reset the state machine. These mismatches occur because all inputs are unknown at start up on the simulator. You therefore go immediately into the when others case, which sets the state variable to state1. When you power up the hardware, it can be in a used state, such as state2, and then advance to a state other than state1. Post-synthesis simulation behaves more like hardware with respect to initialization.

Using Enumerated Types for State Values

Generally, you represent states in VHDL with a user-defined enumerated type.

Copyright © 2013 Synopsys, Inc.

Syntax

type *type_name* **is (** *state1_name*, *state2_name*, ... , *stateN_name* **) ;**

Example

```
type states is (st1, st2, st3, st4, st5, st6, st7, st8);
begin
-- The statement region of a process or subprogram.
next_state := st2 ;
-- Setting the next state to st2
```

Simulation Tips When Using Enumerated Types

You want initialization in simulation to mimic the behavior of hardware when it powers up. Therefore, do not initialize your state machine to a known state during simulation, because the hardware will not be in a known state when it powers up.

Creating an Extra Initialization State

If you use an enumerated type for your state vector, create an extra initialization state in your type definition (for example, stateX), and place it first in the list, as shown in the example below.

```
type state is (stateX, state1, state2, state3, state4);
```

In VHDL, the default initial value for an enumerated type is the leftmost value in the type definition (in this example, stateX). When you begin the simulation, you will be in this initial (simulation only) state.

Detecting Reset Problems

In your state machine case statement, create an entry for staying in stateX when you get in stateX. For example:

```
when stateX => next_state := stateX;
```

Look for your design entering stateX. This means that your design is not resetting properly.

Note: The synthesis tool does not create hardware to represent this initialization state (stateX). It is removed during optimization.

Copyright © 2013 Synopsys, Inc.

Detecting Forgotten Assignment to the Next State

Assign your next state value to stateX immediately before your state machine case statement.

Example

```
next_state := stateX;
case (current_state) is
...
   when state3 =>
      if (foo = '1') then
         next_state := state2;
      end if;
...
end case;
```

Asynchronous State Machines in VHDL

Avoid defining asynchronous state machines in VHDL. An asynchronous state machine has states, but no clearly defined clock, and has combinational loops. However, if you must use asynchronous state machines, you can do one of the following:

- Create a netlist of the technology primitives from the target library for your technology vendor. Any instantiated primitives that are left in the netlist are not removed during optimization.
- Use a schematic editor for the asynchronous state machine part of your design.

Do not use the synthesis tool to design asynchronous state machines; the tool might remove your hazard-suppressing logic when it performs logic optimization, causing your asynchronous state machine to work incorrectly.

The synthesis tool displays a “found combinational loop” warning message for an asynchronous FSM when it detects combinational loops in continuous assignment statements, processes and built-in gate-primitive logic.

Copyright © 2013 Synopsys, Inc.

Asynchronous State Machines that Generate Error Messages

In this example, both async1 and async2 will generate combinational loop errors, because of the recursive definition for output.

```
library ieee;
use ieee.std_logic_1164.all;

entity async is
-- output is a buffer mode so that it can be read
   port (output : buffer std_logic ;
         g, d : in std_logic ) ;
end async ;

-- Asynchronous FSM from concurrent assignment statement
architecture async1 of async is
begin
   -- Combinational loop error, due to recursive output definition.
   output <= (((((g and d) or (not g)) and output) or d) and
      output);
end async1;

-- Asynchronous FSM created within a process
architecture async2 of async is
begin
   process(g, d, output)
   begin
-- Combinational loop error, due to recursive output definition.
      output <= (((((g and d) or (not g)) and output) or d) and
      output);
   end process;
end async2;
```


Copyright © 2013 Synopsys, Inc.

Hierarchical Design Creation in VHDL

Creating hierarchy is similar to creating a schematic. You place available parts from a library onto a schematic sheet and connect them.

To create a hierarchical design in VHDL, you instantiate one design unit inside of another. In VHDL, the design units you instantiate are called components. Before you can instantiate a component, you must declare it (step 2, below).

The basic steps for creating a hierarchical VHDL design are:

1. Write the design units (entities and architectures) for the parts you wish to instantiate.
2. Declare the components (entity interfaces) you will instantiate.
3. Instantiate the components, and connect (map) the signals (including top-level ports) to the formal ports of the components to wire them up.

Step 1 – Write Entities and Architectures

Write entities and architectures for the design units to instantiate.

```
library ieee;
use ieee.std_logic_1164.all;

entity muxhier is
   port (outvec: out std_logic_vector (7 downto 0);
         a_vec, b_vec: in std_logic_vector (7 downto 0);
         sel: in std_logic );
end muxhier;

architecture mux_design of muxhier is
begin
-- <mux functionality>
end mux_design;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
   port (q: buffer std_logic_vector (7 downto 0);
         data: in std_logic_vector (7 downto 0);
         clk, rst: in std_logic );
end reg8;

architecture reg8_design of reg8 is -- 8-bit register
begin
-- <8-bit register functionality>
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
   port (q: buffer std_logic_vector (7 downto 0);
         data: in std_logic_vector (7 downto 0);
         clk, rst, r_l: in std_logic );
end rotate;

architecture rotate_design of rotate is
begin
-- Rotates bits or loads
-- When r_l is high, it rotates; if low, it loads data
-- <Rotation functionality>
end rotate_design;
```

Step 2 – Declare the Components

Components are declared in the declarative region of the architecture with a component declaration statement.

The component declaration syntax is:

component *entity_name*
  port (*port_list* **) ;**
end component ;

The entity_name and port_list of the component must match exactly that of the entity you will be instantiating.

Copyright © 2013 Synopsys, Inc.

Example

```
architecture structural of top_level_design is
-- Component declarations are placed here in the
-- declarative region of the architecture.

component muxhier -- Component declaration for mux
   port (outvec: out std_logic_vector (7 downto 0);
         a_vec, b_vec: in std_logic_vector (7 downto 0);
         sel: in std_logic );
end component;

component reg8 -- Component declaration for reg8
   port (q: out std_logic_vector (7 downto 0);
         data: in std_logic_vector (7 downto 0);
         clk, rst: in std_logic );
end component;

component rotate -- Component declaration for rotate
   port (q: buffer std_logic_vector (7 downto 0);
         data: in std_logic_vector (7 downto 0);
         clk, rst, r_l: in std_logic );
end component;
begin
-- The structural description goes here.
end structural;
```

Step 3 – Instantiate the Components

Use the following syntax to instantiate your components:

unique_instance_name **:** *component_name*
  [**generic map (***override_generic_values* **)]**
    port map (*port_connections* **) ;**

You can connect signals either with positional mapping (the same order declared in the entity) or with named mapping (where you specify the names of the lower-level signals to connect). Connecting by name minimizes errors, and especially advantageous when the component has many ports. To use configuration specification and declaration, refer to *Configuration Specification and Declaration*, on page 910.

Example

```
library ieee;
use ieee.std_logic_1164.all;

entity top_level is
   port (q: buffer std_logic_vector (7 downto 0);
         a, b: in std_logic_vector (7 downto 0);
         sel, r_l, clk, rst: in std_logic );
end top_level;

architecture structural of top_level is
-- The component declarations shown in Step 2 go here.
-- Declare the internal signals here
signal mux_out, reg_out: std_logic_vector (7 downto 0);

begin
-- The structural description goes here.
-- Instantiate a mux, name it inst1, and wire it up.
-- Map (connect) the ports of the mux using positional association.
inst1: muxhier port map (mux_out, a, b, sel);

-- Instantiate a rotate, name it inst2, and map its ports.
inst2: rotate port map (q, reg_out, clk, r_l, rst);

-- Instantiate a reg8, name it inst3, and wire it up.
-- reg8 is connected with named association.
-- The port connections can be given in any order.
-- Notice that the actual (local) signal names are on
-- the right of the '=>' mapping operators, and the
-- formal signal names from the component
-- declaration are on the left.
inst3: reg8 port map (
   clk => clk,
   data => mux_out,
   q => reg_out,
   rst => rst );

end structural;
```


Copyright © 2013 Synopsys, Inc.

Configuration Specification and Declaration

A configuration declaration or specification can be used to define binding information of component instantiations to design entities (entity-architecture pairs) in a hierarchical design. After the structure of one level of a design has been fully described using components and component instantiations, a designer must describe the hierarchical implementation of each component.

A configuration declaration or specification can also be used to define binding information of design entities (entity-architecture pairs) that are compiled in different libraries.

This section discusses usage models of the configuration declaration statement supported by the synthesis tool. The following topics are covered:

- Configuration Specification, on page 910
- Configuration Declaration, on page 914
- VHDL Configuration Statement Enhancement, on page 920

Component declarations and component specifications are not required for a component instantiation where the component name is the same as the entity name. In this case, the entity and its last architecture denote the default binding. In direct-entity instantiations, the binding information is available as the entity is specified, and the architecture is optionally specified. Configuration declaration and/or configuration specification are required when the component name does not match the entity name. If configurations are not used in this case, VHDL simulators give error messages, and the synthesis tool creates a black box and continues synthesis.

Configuration Specification

A configuration specification associates binding information with component labels that represent instances of a given component declaration. A configuration specification is used to bind a component instance to a design entity, and to specify the mapping between the local generics and ports of the component instance and the formal generics and ports of the entity. Optionally, a configuration specification can bind an entity to one of its architectures. The synthesis tool supports a subset of configuration specification commonly used in RTL synthesis; this section discusses that support.

Copyright © 2013 Synopsys, Inc.

The following Backus-Naur Form (BNF) grammar is supported (VHDL-93 LRM pp.73-79):

configuration_specification ::=

for component_specification binding_indication **;**

component_specification ::=

instantiation_list : component_name

instantiation_list ::=

instantiation_label {, instantiation_label } | **others** | **all**

binding_indication ::= [**use** entity_aspect]
[generic_map_aspect]
[port_map_aspect]

entity_aspect ::=

entity entity_name [(architecture_identifier)] |
configuration configuration_name

```
for others: AND_GATE use entity work.AND_GATE(structure);
for all: XOR_GATE use entity work.XOR_GATE;
```

Example: Configuration Specification

In the following example, two architectures (RTL and structural) are defined for an adder. There are two instantiations of an adder in design top. A configuration statement defines the adder architecture to use for each instantiation.

Copyright © 2013 Synopsys, Inc.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity adder is
   port (a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         s : out std_logic;
         cout : out std_logic );
end adder;

library IEEE;
use IEEE.std_logic_unsigned.all;

architecture rtl of adder is
signal tmp : std_logic_vector(1 downto 0);
begin
   tmp <= ('0' & a) - b - cin;
   s <= tmp(0);
   cout <= tmp(1);
end rtl;

architecture structural of adder is
begin
   s <= a xor b xor cin;
   cout <= ((not a) and b and cin) or ( a and (not b) and cin)
      or (a and b and (not cin)) or ( a and b and cin);
end structural;

library IEEE;
use IEEE.std_logic_1164.all;

entity top is
   port (a : in std_logic_vector(1 downto 0);
         b : in std_logic_vector(1 downto 0);
         c : in std_logic;
         cout : out std_logic;
         sum : out std_logic_vector(1 downto 0));
end top;

architecture top_a of top is
component myadder
   port (a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         s : out std_logic;
         cout : out std_logic);
end component;
```

```
signal carry : std_logic;
for s1 : myadder use entity work.adder(structural);
for r1 : myadder use entity work.adder(rtl);
begin
   s1 : myadder port map ( a(0), b(0), c, sum(0), carry);
   r1 : myadder port map ( a(1), b(1), carry, sum(1), cout);
end top_a;
```

Results

Unsupported Constructs for Configuration Specification

The following configuration specification construct is *not* supported by the synthesis tool. An appropriate message is issued in the log file when this construct is used.

- The VHDL-93 LRM defines entity_aspect in the binding indication as:

  entity_aspect ::=

  entity entity_name [(architecture_identifier)] |
  configuration configuration_name | **open**

  The synthesis tool supports entity_name and configuration_name in the entity_aspect of a binding indication. The tool does not yet support the open construct.

Copyright © 2013 Synopsys, Inc.

Configuration Declaration

Configuration declaration specifies binding information of component instantiations to design entities (entity-architecture pairs) in a hierarchical design. Configuration declaration can bind component instantiations in an architecture, in either a block statement, a for…generate statement, or an if…generate statement. It is also possible to bind different entity-architecture pairs to different indices of a for…generate statement.

The synthesis tool supports a subset of configuration declaration commonly used in RTL synthesis. The following Backus-Naur Form (BNF) grammar is supported (VHDL-93 LRM pp.11-17):

configuration_declaration ::=

configuration identifier **of** entity_name **is**

block_configuration

end [**configuration**] [configuration_simple_name] **;**

block_configuration ::=

for block_specification

{ configuration_item }

end for ;

block_specification ::=

achitecture_name | block_statement_label |
generate_statement_label [(index_specification)]

index_specification ::=

discrete_range | static_expression

configuration_item ::=

block_configuration | component_configuration

component_configuration ::=

for component_specification
[binding_indication **;**]
[block_configuration]
end for ;

The BNF grammar for component_specification and binding_indication is described in *Configuration Specification*, on page 910.

Configuration Declaration within a Hierarchy

The following example shows a configuration declaration describing the binding in a 3-level hierarchy, for…generate statement labeled label1, within block statement blk1 in architecture arch_gen3. Each architecture implementation of an instance of my_and1 is determined in the configuration declaration and depends on the index value of the instance in the for…generate statement.

```
entity and1 is
   port(a,b: in bit ; o: out bit);
end;

architecture and_arch1 of and1 is
begin
   o <= a and b;
end;

architecture and_arch2 of and1 is
begin
   o <= a and b;
end;

architecture and_arch3 of and1 is
begin
   o <= a and b;
end;

library WORK; use WORK.all;
entity gen3_config is
   port(a,b: in bit_vector(0 to 7);
       res: out bit_vector(0 to 7));
end;
```


Copyright © 2013 Synopsys, Inc.

```
library WORK; use WORK.all;
architecture arch_gen3 of gen3_config is
component my_and1 port(a,b: in bit ; o: out bit); end component;
begin
   label1: for i in 0 to 7 generate
      blk1: block
      begin
         a1: my_and1 port map(a(i),b(i),res(i));
      end block;
   end generate;
end;

library work;
configuration config_gen3_config of gen3_config is
   for arch_gen3 -- ARCHITECTURE block_configuration "for
         -- block_specification"
      for label1 (0 to 3) --GENERATE block_config "for
         -- block_specification"
         for blk1 -- BLOCK block_configuration "for
         -- block_specification"
         -- {configuration_item}
            for a1 : my_and1 -- component_configuration
            -- Component_specification "for idList : compName"
               use entity work.and1(and_arch1); --
binding_indication
            end for; -- a1 component_configuration
         end for; -- blk1 BLOCK block_configuration
      end for; -- label1 GENERATE block_configuration
      for label1 (4) -- GENERATE block_configuration "for
         -- block_specification"
         for blk1
            for a1 : my_and1
               use entity work.and1(and_arch3);
            end for;
         end for;
      end for;

      for label1 (5 to 7)
         for blk1
            for a1 : my_and1
               use entity work.and1(and_arch2);
            end for;
         end for;
      end for;
   end for; -- ARCHITECTURE block_configuration
end config_gen3_config;
```

Selection with Configuration Declaration

In the following example, two architectures (RTL and structural) are defined for an adder. There are two instantiations of an adder in design top. A configuration declaration defines the adder architecture to use for each instantiation. This example is similar to the configuration specification example.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity adder is
   port (a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         s : out std_logic;
         cout : out std_logic );
end adder;

library IEEE;
use IEEE.std_logic_unsigned.all;

architecture rtl of adder is
signal tmp : std_logic_vector(1 downto 0);
begin
   tmp <= ('0' & a) - b - cin;
   s <= tmp(0);
   cout <= tmp(1);
end rtl;

architecture structural of adder is
begin
   s <= a xor b xor cin;
   cout <= ((not a) and b and cin) or ( a and (not b) and cin) or
      (a and b and (not cin)) or ( a and b and cin);
end structural;

library IEEE;
use IEEE.std_logic_1164.all;

entity top is
   port (a : in std_logic_vector(1 downto 0);
         b : in std_logic_vector(1 downto 0);
         c : in std_logic;
         cout : out std_logic;
         sum : out std_logic_vector(1 downto 0) );
end top;
```


Copyright © 2013 Synopsys, Inc.

```
architecture top_a of top is
component myadder
   port (a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         s : out std_logic;
         cout : out std_logic);
end component;

signal carry : std_logic;
begin
   s1 : myadder port map ( a(0), b(0), c, sum(0), carry);
   r1 : myadder port map ( a(1), b(1), carry, sum(1), cout);
end top_a;

library work;
configuration config_top of top is -- configuration_declaration
   for top_a -- block_configuration "for block_specification"
   -- component_configuration
      for s1: myadder -- component_specification
         use entity work.adder (structural); -- binding_indication
      end for; -- component_configuration
   -- component_configuration
      for r1: myadder -- component_specification
         use entity work.adder (rtl); -- binding_indication
      end for; -- component_configuration
   end for; -- block_configuration
end config_top;
```

Results

Direct Instantiation of Entities Using Configuration

In this method, a configured entity (i.e., an entity with a configuration declaration) is directly instantiated by writing a component instantiation statement that directly names the configuration.

Syntax

```
label : configuration configurationName
    [generic map (actualGeneric1, actualGeneric2, ... ) ]
    [port map ( port1, port2, ... ) ] ;
```

Example – Direct Instantiation Using Configuration Declaration

Unsupported Constructs for Configuration Declaration

The following are the configuration declaration constructs that are *not* supported by the synthesis tool. Appropriate messages are displayed in the log file if these constructs are used.

1. The VHDL-93 LRM defines the configuration declaration as:

   configuration_declaration ::=

   configuration identifier **of** entity_name **is**
   configuration_declarative_part
   block_configuration
   end [**configuration**] [configuration_simple_name] **;**

   configuration_declarative_part ::= { configuration_declarative_item }

   configuration_declarative_item ::=

   use_clause | attribute_specification | group_declaration

   The synthesis tool does not support the configuration_declarative_part. It parses the use_clause and attribute_specification without any warning message. The group_declaration is not supported and an error message is issued.

2. VHDL-93 LRM defines entity aspect in the binding indication as:

   entity_aspect ::=

Copyright © 2013 Synopsys, Inc.

entity entity_name [(architecture_identifier)] |
configuration configuration_name | **open**

block_configuration ::=

for block_specification
{ use_clause }
{ configuration_item }
end for ;

The synthesis tool does not support use_clause in block_configuration. This construct is parsed and ignored.

VHDL Configuration Statement Enhancement

This section highlights the VHDL configuration statement support and handling component declarations with corresponding entity descriptions. Topics include:

- Generic mapping, on page 920
- Port Mapping, on page 921
- Mapping Multiple Entity Names to the Same Component, on page 922
- Generics Assigned to Configurations, on page 923
- Arithmetic Operators and Functions in Generic Maps, on page 928
- Ports in Component Declarations, on page 929

Generic mapping

Generics and ports can have different names and sizes at the entity and component levels. You use the configuration statement to bind them together with a configuration specification or a configuration declaration. The binding priority follows this order:

- Configuration specification
- Component specification
- Component declaration

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity test is
generic ( range1 : integer := 11);
   port (a, a1 : in std_logic_vector( range1 - 1 downto 0);
         b, b1 : in std_logic_vector( range1 - 1 downto 0);
         c, c1 : out std_logic_vector( range1 - 1 downto 0) );
end test;

architecture test_a of test is
component submodule1 is
generic ( size : integer := 6);
   port (a : in std_logic_vector(size -1 downto 0);
         b : in std_logic_vector(size -1 downto 0);
         c : out std_logic_vector(size -1 downto 0) );
end component;

for all : submodule1
use entity work.sub1(rtl)
generic map (size => range1);
begin
   UUT1 : submodule1 generic map (size => 4)
   port map (a => a,b => b,c => c);

end test_a;
```

If you define the following generic map for sub1, it takes priority:

```
entity sub1 is
generic(size: integer:=1);
   port (a: in std_logic_vector(size -1 downto 0);
         b : in std_logic_vector(size -1 downto 0);
         c : out std_logic_vector(size -1 downto 0 );
end sub1;
```

Port Mapping

See *Generic mapping*, on page 920 for information about using the configuration statement and binding priority.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
generic ( range1 : integer := 1);
   port (ta, ta1 : in std_logic_vector( range1 - 1 downto 0);
         tb, tb1 : in std_logic_vector( range1 - 1 downto 0);
         tc, tc1 : out std_logic_vector( range1 - 1 downto 0) );
end test;
```


Copyright © 2013 Synopsys, Inc.

```
architecture test_a of test is
component submodule1
generic ( my_size1 : integer := 6; my_size2 : integer := 6);
   port (d : in std_logic_vector(my_size1 -1 downto 0);
         e : in std_logic_vector(my_size1 -1 downto 0);
         f : out std_logic_vector(my_size2 -1 downto 0) );
end component;

for UUT1 : submodule1
use entity work1.sub1(rtl)
generic map (size1 => my_size1, size2 => my_size2)
port map ( a => d, b => e, c => f);

   begin
   UUT1 : submodule1 generic map (my_size1 => 1, my_size2 => 1)
   port map (d => ta, e => tb,f => tc);
   end test_a;
```

If you define the following port map for sub1, it overrides the previous definition:

```
entity sub1 is
generic(size1: integer:=6; size2:integer:=6);
port (a: in std_logic_vector (size1 -1 downto 0);
      b : in std_logic_vector (size1 -1 downto 0);
      c : out std_logic_vector (size2 -1 downto 0 );
end sub1:
```

Mapping Multiple Entity Names to the Same Component

When a single component has multiple entities, you can use the configuration statement and the for label clause to bind them. The following is an example:

```
entity test is
generic ( range1 : integer := 1);
   port (ta, ta1 : in std_logic_vector(range1 - 1 downto 0);
         tb, tb1 : in std_logic_vector(range1 - 1 downto 0);
         tc, tc1 : out std_logic_vector(range1 - 1 downto 0));
end test;

architecture test_a of test is
component submodule
generic (my_size1 : integer := 6; my_size2 : integer := 6);
   port (d,e : in std_logic_vector(my_size1 -1 downto 0);
         f : out std_logic_vector(my_size2 -1 downto 0));
end component;
```

```
begin
UUT1 : submodule generic map (1, 1)
   port map (d => ta, e => tb, f => tc);
UUT2 : submodule generic map (1, 1) port map
   (d => ta1, e => tb1, f => tc1)
end test_a;

configuration my_config of test is
for test_a
   for UUT1 : submodule
      use entity work.sub1(rtl)
      generic map (my_size1, my_size2)
      port map (d, e, f);
   end for;
   for others : submodule
      use entity work.sub2(rtl)
      generic map (my_size1, my_size2)
      port map ( d, e, f);
   end for;
end for;
end my_config;
```

You can map multiple entities to the same component, as shown here:

```
entity sub1 is
generic(size1: integer:=6; size2:integer:=6);
port (a: in std_logic_vector (size1 -1 downto 0);
      b : in std_logic_vector (size1 -1 downto 0);
      c : out std_logic_vector (size2 -1 downto 0 );
end sub1:

entity sub2 is
generic(width1: integer; width2:integer);
port (a1: in std_logic_vector(width1 -1 downto 0);
      b1 : in std_logic_vector (width1 -1 downto 0);
      c1 : out std_logic_vector (width2 -1 downto 0 );
end sub1:
```

Generics Assigned to Configurations

Generics can be assigned to configurations instead of entities.

Entities can contain more generics than their associated component declarations. Any additional generics on the entities must have default values to be able to synthesize.

Copyright © 2013 Synopsys, Inc.

Entities can also contain fewer generics than their associated component declarations. The extra generics on the component have no affect on the implementation of the entity.

Following are some examples.

Example1

Configuration conf_module1 contains a generic map on configuration conf_c. The component declaration for submodule1 does not have the generic use_extraSYN_ff, however, the entity has it.

```
library ieee;
use IEEE.std_logic_1164.all;

entity submodule1 is
generic (width : integer := 16;
use_extraSYN_ff : boolean := false);
   port (clk : in std_logic;
         b : in std_logic_vector(width - 1 downto 0);
         c : out std_logic_vector(width - 1 downto 0) );
end submodule1;

architecture rtl of submodule1 is
signal d : std_logic_vector(width - 1 downto 0);
begin
no_resynch : if use_extraSYN_ff = false generate
   d <= b;
end generate no_resynch;

resynch : if use_extraSYN_ff = true generate
   process (clk)
   begin
      if falling_edge(clk) then
         d <= b;
      end if;
   end process;
end generate resynch;

   process (clk)
   begin
      if rising_edge(clk) then
         c <= d;
      end if;
   end process;
end rtl;
```

```
configuration conf_c of submodule1 is
   for rtl
   end for;
end configuration conf_c;

library ieee;
use ieee.std_logic_1164.all;

entity module1 is
generic ( width: integer := 16);
   port (clk : in std_logic;
         b : in std_logic_vector(width - 1 downto 0);
         c : out std_logic_vector(width - 1 downto 0) );
end module1;

architecture rtl of module1 is
component submodule1
generic (width: integer := 8);
   port (clk : in std_logic;
         b : in std_logic_vector(width - 1 downto 0);
         c : out std_logic_vector(width - 1 downto 0) );
end component;

begin
UUT2 : submodule1 port map (clk => clk,
   b => b,
   c => c);
end rtl;

library ieee;
configuration conf_module1 of module1 is
   for rtl
      for UUT2 : submodule1
         use configuration conf_c generic map( width => 16,
         use_extraSYN_ff => true);
      end for;
   end for;
end configuration conf_module1;
```


Copyright © 2013 Synopsys, Inc.

Example2

The component declaration for mod1 has the generic size, which is not in the entity. A component declaration can have more generics than the entity, however, extra component generics have no affect on the entity's implementation.

```
library ieee;
use ieee.std_logic_1164.all;

entity module1 is
generic (width: integer := 16;
use_extraSYN_ff : boolean := false);
   port (clk : in std_logic;
         b : in std_logic_vector ( width - 1 downto 0);
         c : out std_logic_vector( width - 1 downto 0) );
end module1;

architecture rtl of module1 is
signal d : std_logic_vector(width - 1 downto 0);
begin
   no_resynch : if use_extraSYN_ff = false generate
      d <= b ;
end generate no_resynch;

resynch : if use_extraSYN_ff = true generate -- insert pipeline
   -- registers
   process (clk)
   begin
      if falling_edge(clk) then
         d <= b;
      end if;
   end process;
end generate resynch;

   process (clk)
   begin
      if rising_edge(clk) then
         c <= d;
      end if;
   end process;
end rtl;

configuration module1_c of module1 is
   for rtl
   end for;
end module1_c;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
   port (clk : in std_logic;
         tb : in std_logic_vector( 7 downto 0);
         tc : out std_logic_vector( 7 downto 0) );
end test;

architecture test_a of test is
component mod1
generic (width: integer := 16;
use_extraSYN_ff: boolean := false;
size : integer := 8);
   port (clk : in std_logic;
         b : in std_logic_vector(width - 1 downto 0);
         c : out std_logic_vector(width - 1 downto 0) );
end component;

begin
UUT1 : mod1 generic map (width => 18)
   port map (clk => clk,
      b => tb,
      c => tc);
end test_a;

Configuration test_c of test is
for test_a
   for UUT1 : mod1
      use configuration module1_c
      generic map (width => 8, use_extraSYN_ff => true);
   end for;
end for;
end test_c;
```


Copyright © 2013 Synopsys, Inc.

Arithmetic Operators and Functions in Generic Maps

Arithmetic operators and functions can be used in generic maps. Following is an example.

Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity sub is
generic (width : integer:= 16);
   port (clk : in std_logic;
         a : in std_logic_vector (width - 1 downto 0);
         y : out std_logic_vector (width - 1 downto 0) );
end sub;

architecture rtl1 of sub is
begin
   process (clk, a)
   begin
      if (clk = '1' and clk'event) then
         y <= a;
      end if;
   end process;
end rtl1;

architecture rtl2 of sub is
begin y <= a;
end rtl2;

configuration sub_c of sub is
for rtl1 end for;
end sub_c;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

Copyright © 2013 Synopsys, Inc.

```
entity test is
generic (mcu_depth : integer:=1;
mcu_width : integer:=16 );
   port (clk : in std_logic;
         a : in std_logic_vector
            ((mcu_depth*mcu_width)-1 downto 0);
         y : out std_logic_vector
            ((mcu_depth*mcu_width)-1downto 0));
end test;

architecture RTL of test is
constant CWIDTH : integer := 2;
constant size : unsigned := "100";
component sub generic ( width : integer );
   port (clk : in std_logic;
         a : in std_logic_vector (CWIDTH - 1 downto 0);
         y : out std_logic_vector (CWIDTH - 1 downto 0) );
end component;

begin i_sub : sub
generic map (width => CWIDTH ) port map (clk => clk,
   a => a,
   y => y );
end RTL;

library ieee;
use ieee.std_logic_arith.all;

configuration test_c of test is
   for RTL
      for i_sub : sub use
         configuration sub_c
         generic map(width => (CWIDTH ** (conv_integer (size))));
      end for;
   end for;
end test_c;
```

Ports in Component Declarations

Entities can contain more or fewer ports than their associated component declarations. Following are some examples.

Example1

```
library ieee;
use ieee.std_logic_1164.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity module1 is
generic ( width: integer := 16; use_extraSYN_ff : boolean :=
false);
   port (clk : in std_logic;
         b : in std_logic_vector ( width - 1 downto 0);
         a : out integer range 0 to 15; --extra output port
            on entity
         e : out integer range 0 to 15; -- extra output port
            on entity
         c : out std_logic_vector( width - 1 downto 0));
end module1;

architecture rtl of module1 is
signal d : std_logic_vector(width - 1 downto 0);
begin
e <= width;
a <= width;
no_resynch : if use_extraSYN_ff = false generate
   d <= b ;
end generate no_resynch;

resynch : if use_extraSYN_ff = true generate
   process (clk)
   begin
      if falling_edge(clk) then
         d <= b;
      end if;
   end process;
end generate resynch;

   process (clk)
   begin
      if rising_edge(clk) then
         c <= d;
      end if;
   end process;
end rtl;

configuration module1_c of module1 is
for rtl
end for;
end module1_c;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
   port (clk : in std_logic;
         tb : in std_logic_vector( 7 downto 0);
         tc : out std_logic_vector( 7 downto 0) );
end test;

architecture test_a of test is
component mod1
generic (width: integer := 16);
   port (clk : in std_logic;
         b : in std_logic_vector(width - 1 downto 0);
         c : out std_logic_vector(width - 1 downto 0) );
end component;

begin
UUT1 : mod1 generic map (width => 18)
port map (clk => clk,
   b => tb,
   c => tc);
end test_a;

Configuration test_c of test is
for test_a
   for UUT1 : mod1
      use configuration module1_c
      generic map (width => 8, use_extraSYN_ff => true);
   end for;
end for;
end test_c;
```

Copyright © 2013 Synopsys, Inc.

In the figure above, the entity module1 has extra ports a and e that are not defined in the corresponding component declaration mod1. The additional ports are not connected during synthesis.

Example2

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY sub1 IS
GENERIC(
size1 : integer := 11;
size2 : integer := 12);
   PORT (r : IN std_logic_vector(size1 -1 DOWNTO 0);
         s : IN std_logic_vector(size1 -1 DOWNTO 0);
         t : OUT std_logic_vector(size2 -1 DOWNTO 0) );
END sub1;

ARCHITECTURE rtl OF sub1 IS
BEGIN
   t <= r AND s;
END ARCHITECTURE rtl;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY test IS
GENERIC (range1 : integer := 12);
   PORT (ta0 : IN std_logic_vector(range1 - 1 DOWNTO 0);
         tb0 : IN std_logic_vector(range1 - 1 DOWNTO 0);
         tc0 : OUT std_logic_vector(range1 - 1 DOWNTO 0) );
END test;

ARCHITECTURE test_a OF test IS
COMPONENT submodule
GENERIC (
my_size1 : integer := 4;
my_size2 : integer := 5);
   PORT (d : IN std_logic_vector(my_size1 -1 DOWNTO 0);
         e : IN std_logic_vector(my_size1 -1 DOWNTO 0);
         ext_1 : OUT std_logic_vector(my_size1 -1 DOWNTO 0);
         ext_2 : OUT std_logic_vector(my_size1 -1 DOWNTO 0);
         f : OUT std_logic_vector(my_size2 -1 DOWNTO 0) );
END COMPONENT;
```

Copyright © 2013 Synopsys, Inc.

```
BEGIN
UUT1 : submodule
GENERIC MAP (
my_size1 => range1,
my_size2 => range1)
   PORT MAP (ext_1 => open,
      ext_2 => open,
      d => ta0,
      e => tb0,
      f => tc0 );
END test_a;

CONFIGURATION my_config OF test IS
   FOR test_a
      FOR UUT1 : submodule
      USE ENTITY work.sub1(rtl)
      GENERIC MAP (
         size1 => my_size1,
         size2 => my_size2)
      PORT MAP (r => d,
         s => e,
         t => f );
      END FOR;
   END FOR; -- test_a
END my_config;
```

In the figure above, the component declaration has more ports (ext_1 ext_2) than entity sub1. The component is synthesized based on the number of ports on the entity.

Copyright © 2013 Synopsys, Inc.

Scalable Designs

This section describes creating and using scalable VHDL designs. You can create a VHDL design that is scalable, meaning that it can handle a user-specified number of bits or components.

- Creating a Scalable Design Using Unconstrained Vector Ports, on page 934
- Creating a Scalable Design Using VHDL Generics, on page 935
- Using a Scalable Architecture with VHDL Generics, on page 936
- Creating a Scalable Design Using Generate Statements, on page 938

Creating a Scalable Design Using Unconstrained Vector Ports

Do not size (constrain) the ports until you need them. This first example is coding the adder using the - operator, and gives much better synthesized results than the second and third scalable design examples, which code the adders as random logic.

Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity addn is
-- Notice that a, b, and result ports are not constrained.
-- In VHDL, they automatically size to whatever is connected
-- to them.
   port (result : out std_logic_vector;
         cout : out std_logic;
         a, b : in std_logic_vector;
         cin : in std_logic );
end addn;

architecture stretch of addn is
   signal tmp : std_logic_vector (a'length downto 0);
begin
-- The next line works because "-" sizes to the largest operand
-- (also, you need only pad one argument).
tmp <= ('0' & a) - b - cin;
```

```
result <= tmp(a'length - 1 downto 0);
cout <= tmp(a'length);
assert result'length = a'length;
assert result'length = b'length;
end stretch;

-- Top level design
-- Here is where you specify the size for a, b,
-- and result. It is illegal to leave your top
-- level design ports unconstrained.

library ieee;
use ieee.std_logic_1164.all;

entity addntest is
   port (result : out std_logic_vector (7 downto 0);
         cout : out std_logic;
         a, b : in std_logic_vector (7 downto 0);
         cin : in std_logic );
end addntest;

architecture top of addntest is
component addn
   port (result : std_logic_vector;
         cout : std_logic;
         a, b : std_logic_vector;
         cin : std_logic );
end component;

begin
test : addn port map (result => result,
   cout => cout,
   a => a,
   b => b,
   cin => cin );
end;
```

Creating a Scalable Design Using VHDL Generics

Create a VHDL generic with default value. The generic is used to represent bus sizes inside a architecture, or a number of components. You can define more than one generic per declaration by separating the generic definitions with semicolons (;).

Copyright © 2013 Synopsys, Inc.

Syntax

generic (*generic_1_name* **:** *type* **[:=** *default_value***]) ;**

Examples

```
generic (num : integer := 8) ;
generic (top : integer := 16; num_bits : integer := 32);
```

Using a Scalable Architecture with VHDL Generics

Instantiate the scalable architecture, and override the default generic value with the generic map statement.

Syntax

generic map (*list_of_overriding_values* **)**

Examples

Generic map construct

```
generic map (16)
-- These values will get mapped in order given.
generic map (8, 16)
```

Creating a scalable adder

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
   generic(num_bits : integer := 4); -- Default adder
      -- size is 4 bits
   port (a : in std_logic_vector (num_bits downto 1);
         b : in std_logic_vector (num_bits downto 1);
         cin : in std_logic;
         sum : out std_logic_vector (num_bits downto 1);
         cout : out std_logic );
end adder;
```

Copyright © 2013 Synopsys, Inc.

```
architecture behave of adder is
begin
   process (a, b, cin)
   variable vsum : std_logic_vector (num_bits downto 1);
   variable carry : std_logic;
   begin
   carry := cin;
      for i in 1 to num_bits loop
         vsum(i) := (a(i) xor b(i)) xor carry;
         carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));
      end loop;
   sum <= vsum;
   cout <= carry;
   end process;
end behave;
```

Scaling the Adder by Overriding the generic Statement

```
library ieee;
use ieee.std_logic_1164.all;

entity adder16 is
   port (a : in std_logic_vector (16 downto 1);
         b : in std_logic_vector (16 downto 1);
         cin : in std_logic;
         sum : out std_logic_vector (16 downto 1);
         cout : out std_logic );
end adder16;

architecture behave of adder16 is
-- The component declaration goes here.
-- This allows you to instantiate the adder.
component adder
-- The default adder size is 4 bits.
generic(num_bits : integer := 4);
   port (a : in std_logic_vector ;
         b : in std_logic_vector;
         cin : in std_logic;
         sum : out std_logic_vector;
         cout : out std_logic );
end component;

begin
my_adder : adder
   generic map (16) -- Use a 16 bit adder
   port map(a, b, cin, sum, cout);
end behave;
```


Copyright © 2013 Synopsys, Inc.

Creating a Scalable Design Using Generate Statements

A VHDL generate statement allows you to repeat logic blocks in your design without having to write the code to instantiate each one individually.

Creating a 1-bit Adder

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
   port (a, b, cin : in std_logic;
         sum, cout : out std_logic );
end adder;

architecture behave of adder is
begin
   sum <= (a xor b) xor cin;
   cout <= (a and b) or (cin and a) or (cin and b);
end behave;
```

Instantiating the 1-bit Adder Many Times with a Generate Statement

```
library ieee;
use ieee.std_logic_1164.all;

entity addern is
generic(n : integer := 8);
   port (a, b : in std_logic_vector (n downto 1);
         cin : in std_logic;
         sum : out std_logic_vector (n downto 1);
         cout : out std_logic );
end addern;

architecture structural of addern is
-- The adder component declaration goes here.
component adder
   port (a, b, cin : in std_logic;
         sum, cout : out std_logic);
end component;
```

Copyright © 2013 Synopsys, Inc.

```
signal carry : std_logic_vector (0 to n);
begin
-- Generate instances of the single-bit adder n times.
-- You need not declare the index 'i' because
-- indices are implicitly declared for all FOR
-- generate statements.

gen: for i in 1 to n generate
   add: adder port map(
      a => a(i),
      b => b(i),
      cin => carry(i - 1),
      sum => sum(i),
      cout => carry(i));
end generate;

carry(0) <= cin;
cout <= carry(n);

end structural;
```


Copyright © 2013 Synopsys, Inc.

VHDL Guidelines for RAM Inference

This section provides guidelines for synthesis using RAMS and covers the following topics:

- RAM Inference in VHDL Designs, on page 940
- Limited RAM Resources, on page 942
- Additional Components, on page 942
- Synchronous READ RAMs, on page 944
- Multi-Port RAM Extraction, on page 942

RAM Inference in VHDL Designs

The synthesis tool can automatically infer synchronous and synchronously resettable RAMs from your VHDL source code and, where appropriate, generate technology-specific single or dual-port RAMs. You do not need any special input, like attributes, in your source code for this inference. To override the automatic inference or specify implementations, use the syn_ramstyle attribute (syn_ramstyle, on page 1467).

Automatic RAM inference and implementation is a two-step process:

- The compiler infers an RTL-level RAM component. This always has an asynchronous READ. The RAM component that is inferred can be one of the following:

RAM1	RAM component
RAM2	RAM component with reset
NRAM	RAM component with multiple ports

- During synthesis, the mappers convert the RTL-level RAM inferred by the compiler to the appropriate technology-specific RAM primitives. Depending on the technology used, the synthesized design can include RAM primitives with either synchronous or asynchronous READs. See *Synchronous READ RAMs*, on page 944, for information on coding your VHDL description to ensure that you get technology-specific RAM primitives with synchronous READs.

Copyright © 2013 Synopsys, Inc.

For certain technologies, the mappers implement single-port, dual-port, or multi-port RAMs. The following table lists the supported technology-specific RAMs that can be generated by the synthesis tool.

Technology vendor	Family	RAM Type
Microsemi	ProASIC, ProASIC^{PLUS}, ProASIC3/3E/3L	Single and dual port
Altera	Stratix, Stratix II, Stratix III, Stratix IV, Cyclone, Cyclone II, Cyclone III, Cyclone IV E, Cyclone IV GX	Single and dual port
Altera	Stratix, Stratix II, Stratix III	Multi-port
Lattice	ORCA FPSC, ORCA Series 2, 3, MachXO, LatticeECP2M/ECP2, LatticeECP/EC, LatticeXP2/XP, LatticeSC/SCM	Single and dual port
Xilinx	Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, Virtex-6, Spartan-3, Spartan-6, XC4000E/EX/XL/XLA/XV, XC9500/XL	Single and dual port
Xilinx	Spartan-II, Spartan-XL	Multi-port (true dual-port)

Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_test is
   port (d : in std_logic_vector(7 downto 0);
         a : in std_logic_vector(6 downto 0);
         we : in std_logic;
         clk : in std_logic;
         q : out std_logic_vector(7 downto 0) );
end ram_test;

architecture rtl of ram_test is
type mem_type is array (127 downto 0) of
   std_logic_vector (7 downto 0);
signal mem: mem_type;
begin
   process (clk)
   begin
```


Copyright © 2013 Synopsys, Inc.

```
            if rising_edge(clk) then
               if (we = '1') then
                  mem(conv_integer (a)) <= d;
               end if;
            end if;
         end process;
      q <= mem(conv_integer (a));

      end rtl ;
```

Limited RAM Resources

If your RAM resources are limited, designate additional instances of inferred RAMs as flip-flops and logic using the syn_ramstyle attribute. This attribute takes the string argument of registers, placed on the RAM instance.

Additional Components

After inferring a RAM for some technologies, you might notice that the synthesis tool has generated a few additional components (glue logic) adjacent to the RAM. These components assure accuracy in your post placement and routing simulation results.

Multi-Port RAM Extraction

The compiler extracts multi-write port RAMs, creating an nram primitive which can have n write ports. The nrams are implemented differently depending on your technology. For information about the implementations, see the appropriate vendor chapter.

The following aspects of multi-port RAM extraction are described here:

- Prerequisites for Multi-Port RAM Extraction, on page 943
- Write Processes and Multi-Port RAM Inference, on page 943
- Multi-port RAM Recommended Coding Style, on page 946

Prerequisites for Multi-Port RAM Extraction

In order to infer the nram, you must do the following:

- The compiler will infer the nram even if you do not register the read address, as subsequent examples show. However, for many technology families, you must register the read address for the nram to be implemented as vendor RAMs.
- If the writes are in different processes, you no longer need to specify no_rw_check for the syn_ramstyle attribute in order to infer the RAM. Note that the simulator produces x's when the two writes occur at the same time while the synthesis tool assumes that one write gets priority over the other based on the code.

Write Processes and Multi-Port RAM Inference

The compiler infers the multi-port RAMs from the write processes as follows:

- When all the writes are made in one process, there are no address conflicts, and the compiler generates an nram.
- When writes are made in multiple processes, the software does not infer a multi-port RAM unless there is an explicit syn_ramstyle attribute. If there is no attribute, the software does not infer a RAM. If there is an attribute, the software infers a RAM with multiple write ports. You might get a warning about simulation mismatches when the two addresses are the same.

Copyright © 2013 Synopsys, Inc.

Multi-Port RAM Recommended Coding Style

Currently, you must use a registered read address when you code the RAM. If you do not, you could get error messages.

There are two situations which can result in the error message: "@E:MF216 : ram.v(29) | Found NRAM mem_1[7:0] with multiple processes":

- An nram with two clocks and two write addresses has syn_ramstyle set to a value of registers. The software cannot implement this, because there is a physical FPGA limitation that does not allow registers with multiple writes.
- You have a registered output for an nram with two clocks and two write addresses.

Synchronous READ RAMs

All RAM primitives that the synthesis tool generates for inferred RAMs have asynchronous READs, except for Altera Stratix RAMs, single-port Altera FLEX10K RAMs and Xilinx Virtex block SelectRAM+ devices. For these exceptions, the tool generates a synchronous RAM.

To generate a synchronous RAM, you must either register the read address or the RAM output. The following are examples of the VHDL code needed to do this. For details of the corresponding procedures, see *Implementing Altera RAMs Automatically*, on page 376, *Implementing Xilinx RAMs Automatically*, on page 379 and *Inferring Xilinx Block RAMs Using Registered Output*, on page 387 in the *User Guide*.

Example: READ Address Registered

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_test is
   port (d : in std_logic_vector(7 downto 0);
         a : in std_logic_vector(6 downto 0);
         we : in std_logic;
         clk : in std_logic;
         q : out std_logic_vector(7 downto 0) );
end ram_test;
```

```
architecture rtl of ram_test is
type mem_type is array (127 downto 0) of
   std_logic_vector (7 downto 0);
signal mem: mem_type;
signal read_add : std_logic_vector(6 downto 0);
begin
   process (clk)
   begin
      if rising_edge(clk) then
         if (we = '1') then
            mem(conv_integer(a)) <= d;
         end if;
         read_add <= a;
      end if;
   end process;

q <= mem(conv_integer(read_add));
end rtl ;
```

Example: Data Output Registered

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_test is
   port (d: in std_logic_vector(7 downto 0);
         a: in integer range 127 downto 0;
         we: in std_logic;
         clk: in std_logic;
      q: out std_logic_vector(7 downto 0) );
end ram_test;

architecture rtl of ram_test is
type mem_type is array (127 downto 0) of
   std_logic_vector (7 downto 0);
signal mem: mem_type;
begin
   process(clk)
   begin
      if (clk'event and clk='1') then
         q <= mem(a);
            if (we='1') then
```


Copyright © 2013 Synopsys, Inc.

```
                mem(a) <= d;
            end if;
        end if;
    end process;
end rtl;
```

Multi-port RAM Extraction

The compiler extracts multi-write port RAMs, creating an nram primitive which can have n write ports. The nrams are implemented differently depending on your technology. For more information about the implementations, see the appropriate vendor chapter.

In order for the compiler to infer an nram, you must do the following:

- Register the read address.
- Add the syn_ramstyle attribute.

The compiler then infers the multi-port RAMs from the write processes as follows:

- When all the writes are made in one process, there are no address conflicts, and the compiler generates an nram.
- When writes are made in multiple processes, the software does not infer a multi-port RAM unless there is an explicit syn_ramstyle attribute. If there is no attribute, the software does not infer a RAM. If there is an attribute, the software infers a RAM with multiple write ports. You might get a warning about simulation mismatches when the two addresses are the same.
- If the writes are in different processes, you no longer need to specify no_rw_check for the syn_ramstyle attribute in order to infer the RAM. Note that the simulator produces x's when the two writes occur at the same time while the synthesis tool assumes that one write gets priority over the other based on the code.

Multi-port RAM Recommended Coding Style

You must use a registered read address when you code the RAM or have writes to one process. If you have writes to multiple processes, you must use the syn_ramstyle attribute to infer the RAM.

There are two situations which can result in this error message: "@E:MF216 : ram.v(29) | Found NRAM mem_1[7:0] with multiple processes":

- An nram with two clocks and two write addresses has syn_ramstyle set to a value of registers. The software cannot implement this, because there is a physical FPGA limitation that does not allow registers with multiple writes.
- You have a registered output for an nram with two clocks and two write addresses.

Two-write Port RAM Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity one is
generic (data_width : integer := 4;
address_width :integer := 5 );
   port (data_a:in std_logic_vector(data_width-1 downto 0);
         data_b:in std_logic_vector(data_width-1 downto 0);
         addr_a:in std_logic_vector(address_width-1 downto 0);
         addr_b:in std_logic_vector(address_width-1 downto 0);
         wren_a:in std_logic;
         wren_b:in std_logic;
         clk:in std_logic;
         q_a:out std_logic_vector(data_width-1 downto 0);
         q_b:out std_logic_vector(data_width-1 downto 0) );
end one;

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);
signal mem : mem_array;
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "no_rw_check" ;
signal addr_a_reg : std_logic_vector(address_width-1 downto 0);
signal addr_b_reg : std_logic_vector(address_width-1 downto 0);
begin
   WRITE_RAM : process (clk)
   begin
      if rising_edge(clk) then
         if (wren_a = '1') then
            mem(to_integer(unsigned(addr_a))) <= data_a;
         end if;
         if (wren_b='1') then
```


Copyright © 2013 Synopsys, Inc.

```
            mem(to_integer(unsigned(addr_b))) <= data_b;
         end if;
         addr_a_reg <= addr_a;
         addr_b_reg <= addr_b;
      end if;
   end process WRITE_RAM;
q_a <= mem(to_integer(unsigned(addr_a_reg)));
q_b <= mem(to_integer(unsigned(addr_b_reg)));
end rtl;
```

VHDL Coding for Altera RAM Inference

See the following for VHDL guidelines and examples for implementing Altera RAMs:

- Stratix, Arria and Cyclone Dual-Port VHDL RAM Example, on page 948
- VHDL Coding for StratixIII and Stratix IV RAMs with Control Signals, on page 951
- VHDL Coding for Cyclone III RAMs with Control Signals, on page 951

Stratix, Arria and Cyclone Dual-Port VHDL RAM Example

There are several ways to write RTL code to describe an Altera dual-port RAM. The software only maps an inferred RAM to Altera's dedicated synchronous RAM resources when it can construct hardware that exactly matches the behavior of the RTL.

The Altera dual-port RAM has two ports that can be used for read and write. The synthesis software maps to true dual-port (where both ports are used for reading and writing simultaneously) in the following cases:

- An inferred RAM with two writes and one read. The read shares an address with only one of the write ports
- Two inferred RAMs with exactly the same write addresses, clocks, and enables, but with different read address can be paired together and mapped

Copyright © 2013 Synopsys, Inc.

Altera dual-port RAM supports only two clocks, one for each port. If an inferred RAM with more than two write ports uses more than two clocks for read and write operations, the RAM is mapped to logic.

The following is an example or read-first dual-port RAM. For a Verilog example of write-first dual-port RAM, see *Write-First Dual-Port RAM*, on page 723.

Read First Dual-Port RAM

If your coding style permits mismatches between the RTL behavior and the hardware behavior when you simultaneously write to and read from the same address, the software does not map it to a RAM.

When a dual-port RAM is inferred in RTL using non-recommended coding styles, the software issues a warning message like this one:

```
"<memory> instance will be mapped into logic. Please refer to the
online help section of stratix_ram_block for the recommended
coding style to map to ALTSYNCRAM."
```

To map into RAM, the RTL behavior must match the hardware behavior. Make the value of the read data a don't care for simultaneous read and write operations to the same address. This ensures that the synthesis software infers a RAM and saves the user from spending time debugging the differences between the RTL RAM and hardware RAM behaviors. The following example shows the recommended VHDL coding style.

VHDL Altera Read-First Dual-Port Example

```
Library IEEE ;
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use IEEE.std_logic_unsigned.all ;

entity Dual_Port_ReadFirst is
   generic (data_width: integer :=4;
   address_width: integer :=10);
```


Copyright © 2013 Synopsys, Inc.

```
port (write_enable: in std_logic;
   write_clk, read_clk: in std_logic;
   data_in: in std_logic_vector (data_width-1 downto 0);
   data_out: out std_logic_vector (data_width-1 downto 0);
   write_address: in std_logic_vector (address_width-1 downto 0);
   read_address: in std_logic_vector (address_width-1 downto 0)
   );
end Dual_Port_ReadFirst;

architecture behavioral of Dual_Port_ReadFirst is
type memory is array (2**(address_width-1) downto 0) of
   std_logic_vector (data_width-1 downto 0);
signal mem : memory;

signal reg_write_address : std_logic_vector (address_width-1 downto 0);
signal reg_write_enable: std_logic;

attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "block_ram";

begin
register_enable_and_write_address:
   process (write_clk,write_enable,write_address,data_in)
   begin
      if (rising_edge(write_clk)) then
         reg_write_address <= write_address;
         reg_write_enable <= write_enable;
      end if;
   end process;

write:
   process (read_clk,write_enable,write_address,data_in)
   begin
      if (rising_edge(write_clk)) then
         if (write_enable='1') then
            mem(conv_integer(write_address))<=data_in;
         end if;
      end if;
   end process;

read:
   process (read_clk,write_enable,read_address,write_address)
   begin
      if (rising_edge(read_clk)) then
         if (reg_write_enable='1') and (read_address =
            reg_write_address) then data_out <= "XXXX";
```

```
            else
               data_out<=mem(conv_integer(read_address));
            end if;
         end if;
      end process;

   end behavioral;
```

VHDL Coding for StratixIII and Stratix IV RAMs with Control Signals

The enable pins on Altera StratixIII and StratixIV RAM blocks provide built-in functionality for control signals. You can set up your VHDL code so that the synthesis tool extracts control signals for single-port and simple dual-port RAMs with a single common clock, and maps the logic to StratixIII and StratixIV LutRAMs instead of registers. This avoids inferring extra logic, and improves resource utilization and QOR. The control signals can be read enables, write enables and clock enables. See *Verilog Coding for StratixIII and StratixIV RAMs with Control Signals*, on page 724 for details.

VHDL Coding for Cyclone III RAMs with Control Signals

You can extract the control signals and take advantage of the built-in functionality for control signals provided with CycloneIII RAMs. The synthesis tool extracts control signals for single-port, simple dual-port and true dual-port RAMs and maps the logic to CycloneIII dedicated RAMs instead of registers.

See *Verilog Coding for Cyclone III RAMs with Control Signals*, on page 730 for details and examples.

Copyright © 2013 Synopsys, Inc.

VHDL Coding for Xilinx RAM Inference

See the following for VHDL guidelines and examples for implementing Xilinx RAMs:

- VHDL Coding for Xilinx Single-port and Dual-port Block RAMs, on page 952
- VHDL Guidelines for Byte-Enable Block RAMs, on page 954
- VHDL Guidelines for Byte-Wide Write Enable RAMs, on page 955

VHDL Coding for Xilinx Single-port and Dual-port Block RAMs

Block RAMs support multiple clocks and use enables and resets to control RAM read and write operations. By default, the tool infers block SelectRAM, but you can force block RAM inference by specifying the syn_ramstyle attribute with a value of block_ram or no_rw_check.

For syntax information, see *syn_ramstyle*, on page 1467; for usage, see *Specifying RAM Implementation Styles*, on page 375 of the *User Guide*. See *VHDL Guidelines for RAM Inference*, on page 940 for other details.

Single-Port Block RAMs

In single-port block RAMs, the enable signal has the highest priority. This table describes what happens with different enable and reset settings:

Enable	Reset	Results
Inactive	-	All read and write operations are disabled.
Active	Active	Write operations and memory contents are unaffected (data_out is 0).
Active	Inactive	If Write Enable is active, data_in goes to data_out and is updated in memory. If Write Enable is inactive, data_out reflects the memory contents

For a code segment to map to a single-port block RAM, there must be a one-to-one correspondence between the following signals:

- Read and write clocks
- Read and write addresses
- Enable signal
- Write Enable signal

There are three inferred write modes, and they determine what appears at data_out when Write Enable is active.

Writefirst	data_in goes to data_out
Readfirst	memory goes to data_out
Nochange	data_out remains unchanged

The following table shows device support for the inferred modes:

Writefirst	All Virtex families and all Spartan families
Readfirst	All Virtex families; all Spartan-3 and later families
Nochange	All Virtex families; all Spartan-3 and later families

This table shows that a code segment that can be mapped to a single-port block RAM in a more recent Xilinx device family where you may not be able to map it to a block RAM with an earlier Xilinx family. These code segments are mapped to distributed RAM.

For information about implementing single-port RAMs, see *Specifying RAM Implementation Styles*, on page 375 and *Implementing Xilinx RAMs Automatically*, on page 379 in the *User Guide*.

Dual-Port Block RAMs

Except for additional address collision recovery logic, a dual-port RAM is essentially two independent single-port RAMs that share a common memory. Dual-port RAMs have only one write port.

Copyright © 2013 Synopsys, Inc.

For information about implementing dual-port RAMs, see *Specifying RAM Implementation Styles*, on page 375 and *Implementing Xilinx RAMs Automatically*, on page 379 in the *User Guide*.

VHDL Guidelines for Byte-Enable Block RAMs

For Xilinx designs, the tool infers byte enable pins automatically and maps it to block RAM, as long as the it is coded using an if-else-if (mutually exclusive) style.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test is
   port (clock: in std_logic;
         wren: in std_logic;
         rden: in std_logic;
         address : in std_logic_vector (8 downto 0);
         byteena: in std_logic_vector (0 downto 0);
         data: in std_logic_vector (7 downto 0);
         q: out std_logic_vector (7 downto 0) );
end test;

architecture rtl of test is
type memory is array(0 to 127) of std_logic_vector(7 downto 0);
signal mem : memory := (others => (others => '0'));
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "block_ram";
begin
   mem_write : process (clock)
   begin
      if (clock'event and clock = '1') then
         if (wren = '1') then
            if (byteena(0) = '1') then
               mem(conv_integer(address))(7 downto 0) <=
               data(7 downto 0);
            end if;
         end if;
      end if;
   end process mem_write;
```

```
   mem_read : process (clock)
   begin
      if (clock'event and clock = '1') then
         if (rden = '1') then
            q <= mem(conv_integer(address));
         end if;
      end if;
   end process mem_read;

end architecture rtl;
```

VHDL Guidelines for Byte-Wide Write Enable RAMs

For Xilinx designs, the synthesis tool infers byte-wide write enable for RAMs with mutually exclusive writes and maps it to block RAMs. You can specify a dual-port RAM with only one read and one write.

True dual-port RAMs with byte-wide enables cannot be implemented in block RAM.

Example 1

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test is
   port (
      clock : in std_logic;
      address : in std_logic_vector (8 downto 0);
      byteena : in std_logic_vector (3 downto 0);
      data : in std_logic_vector (31 downto 0);
      q : out std_logic_vector (31 downto 0) );
end test;

architecture rtl of test is
   type memory is array(255 downto 0) of std_logic_vector
      (31 downto 0);
   signal mem : memory := (others => (others => '0'));
   attribute syn_ramstyle : string;
   attribute syn_ramstyle of mem : signal is
      "block_ram, no_rw_check";
```


Copyright © 2013 Synopsys, Inc.

```
begin
   mem_write : process (clock)
   begin
      if (clock'event and clock = '1') then
         if (byteena(3) = '1') then
         mem(conv_integer(address))(7 downto 0) <= data
            (7 downto 0);
      end if;
      if (byteena(2) = '1') then
         mem(conv_integer(address))(15 downto 8) <= data
            (15 downto 8);
      end if;
      if (byteena(1) = '1') then
         mem(conv_integer(address))(23 downto 16) <= data
            (23 downto 16);
      end if;
      if (byteena(0) = '1') then
         mem(conv_integer(address))(31 downto 24) <= data
            (31 downto 24);
      end if;
   end if;
end process mem_write;
mem_read : process (clock)
   begin
   if (clock'event and clock = '1') then
      q <= mem(conv_integer(address));
   end if;
end process mem_read;

end architecture rtl;
```

Example 2

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test is
   port (
      clock : in std_logic;
      address : in std_logic_vector (8 downto 0);
      byteena : in std_logic_vector (3 downto 0);
      data : in std_logic_vector (31 downto 0);
      q : out std_logic_vector (31 downto 0) );
end test;
```

```
architecture rtl of test is
   type memory is array(255 downto 0) of std_logic_vector
      (31 downto 0);
   signal mem : memory := (others => (others => '0'));
   attribute syn_ramstyle : string;
   attribute syn_ramstyle of mem : signal is
      "block_ram, no_rw_check";
begin
   mem_write : process (clock)
   begin
      if (clock'event and clock = '1') then
         if (byteena(3) = '1') then
            mem(conv_integer(address))(7 downto 0) <= data
               (7 downto 0);
      elsif (byteena(2) = '1') then
            mem(conv_integer(address))(15 downto 8) <= data
               (15 downto 8);
      elsif (byteena(1) = '1') then
            mem(conv_integer(address))(23 downto 16) <= data
               (23 downto 16);
      elsif (byteena(0) = '1') then
            mem(conv_integer(address))(31 downto 24) <= data
               (31 downto 24);
      end if;
   end if;
end process mem_write;

mem_read : process (clock)
   begin
      if (clock'event and clock = '1') then
         q <= mem(conv_integer(address));
   end if;
end process mem_read;

end architecture rtl;
```

Instantiating RAMs with SYNCORE

The SYNCORE Memory Compiler is available under the IP Wizard to help you generate HDL code for your specific RAM implementation requirements. For information on using the SYNCORE Memory Compiler, see *Specifying RAMs with SYNCore*, on page 939 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

ROM Inference

As part of BEST (Behavioral Extraction Synthesis Technology) feature, the synthesis tool infers ROMs (read-only memories) from your HDL source code, and generates block components for them in the RTL view.

The data contents of the ROMs are stored in a text file named `rom.info`. To quickly view `rom.info` in read-only mode, synthesize your HDL source code, open an RTL view, then push down into the ROM component. For an example of the ROM data, refer to *ROM Table Data (rom.info File)*, on page 747.

Generally, the synthesis tool infers ROMs from HDL source code that uses case statements, or equivalent if statements, to make 16 or more signal assignments using constant values (words). The constants must all be the same width.

Another requirement for ROM inference is that values must be specified for at least half of the address space. For example, if the ROM has 5 address bits, then the address space is 32 and at least 16 of the different addresses must be specified.

Example

```
library ieee;
use ieee.std_logic_1164.all;

entity rom4 is
   port (a : in std_logic_vector(4 downto 0);
         z : out std_logic_vector(3 downto 0) );
end rom4;

architecture behave of rom4 is
begin
   process(a)
   begin
      if a = "00000" then
         z <= "0001";
      elsif a = "00001" then
         z <= "0010";
      elsif a = "00010" then
         z <= "0110";
      elsif a = "00011" then
         z <= "1010";
      elsif a = "00100" then
         z <= "1000";
```

```
      elsif a = "00101" then
         z <= "1001";
      elsif a = "00110" then
         z <= "0000";
      elsif a = "00111" then
         z <= "1110";
      elsif a = "01000" then
         z <= "1111";
      elsif a = "01001" then
         z <= "1110";
      elsif a = "01010" then
         z <= "0001";
      elsif a = "01011" then
         z <= "1000";
      elsif a = "01100" then
         z <= "1110";
      elsif a = "01101" then
         z <= "0011";
      elsif a = "01110" then
         z <= "1111";
      elsif a = "01111" then
         z <= "1100";
      elsif a = "10000" then
         z <= "1000";
      elsif a = "10001" then
         z <= "0000";
      elsif a = "10010" then
         z <= "0011";
      else
         z <= "0111";
      end if;
   end process;
end behave;
```


Copyright © 2013 Synopsys, Inc.

ROM Table Data (rom.info file)

```
Note: This data is for viewing only

ROM work.rom4(behave)-z_1[3:0]
address width: 5
data width: 4
inputs:
0: a[0]
1: a[1]
2: a[2]
3: a[3]
4: a[4]
outputs:
0: z_1[0]
1: z_1[1]
2: z_1[2]
3: z_1[3]

data:
00000 -> 0001
00001 -> 0010
00010 -> 0110
00011 -> 1010
00100 -> 1000
00101 -> 1001
00110 -> 0000
00111 -> 1110
01000 -> 1111
01001 -> 1110
01010 -> 0001
01011 -> 1000
01100 -> 1110
01101 -> 0011
01110 -> 0010
01111 -> 0010
10000 -> 0010
10001 -> 0010
10010 -> 0010
default -> 0111
```

Copyright © 2013 Synopsys, Inc.

Instantiating Black Boxes in VHDL

Black boxes are design units with just the interface specified; internal information is ignored by the synthesis tool. Black boxes can be used to directly instantiate:

- Technology-vendor primitives and macros (including I/Os).
- User-defined macros whose functionality was defined in a schematic editor, or another input source (when the place-and-route tool can merge design netlists from different sources).

Black boxes are specified with the syn_black_box synthesis directive, in conjunction with other directives. If the black box is a technology-vendor I/O pad, use the black_box_pad_pin directive instead.

Here is a list of the directives that you can use to specify modules as black boxes, and to define design objects on the black box for consideration during synthesis:

- syn_black_box
- black_box_pad_pin
- black_box_tri_pins
- syn_isclock
- syn_tco<*n*>
- syn_tpd<*n*>
- syn_tsu<*n*>

For descriptions of the black-box attributes and directives, see Chapter 16, *Attribute and Directive Syntax*.

For information on how to instantiate black boxes and technology-vendor I/Os, see *Defining Black Boxes for Synthesis*, on page 330 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Black-Box Timing Constraints

You can provide timing information for your individual black box instances. The following are the three predefined timing constraints available for black boxes.

- syn_tpd<*n*> – Timing propagation for combinational delay through the black box.
- syn_tsu<*n*> – Timing setup delay required for input pins (relative to the clock).
- syn_tco<*n*>– Timing clock to output delay through the black box.

Here, *n* is an integer from 1 through 10, inclusive. See *syn_black_box*, on page 1235, for details about constraint syntax.

VHDL Attribute and Directive Syntax

Synthesis attributes and directives can be defined in the VHDL source code to control the way the design is analyzed, compiled, and mapped. *Attributes* direct the way your design is optimized and mapped during synthesis. *Directives* control the way your design is analyzed prior to compilation. Because of this distinction, directives must be included in your VHDL source code while attributes can be specified either in the source code or in a constraint file.

The synthesis tool directives and attributes are predefined in the attributes package in the synthesis tool library. This library package contains the built-in attributes, along with declarations for timing constraints (including black-box timing constraints) and vendor-specific attributes. The file is located here:

installDirectory/lib/vhd/synattr.vhd

There are two ways to specify VHDL attributes and directives:

- Using the attributes Package, on page 963
- Declaring Attributes, on page 964

You can either use the attributes package or redeclare the types of directives and attributes each time you use them. You typically use the attributes package.

Using the attributes Package

This is the most typical way to specify the attributes, because you only need to specify the package once. You specify the attributes package, using the following code:

library synplify;
use synplify.attributes.all;
-- design_unit_declarations
attribute *productname_attribute* **of** *object* **:** *object_type* **is** *value* **;**

The following is an example using syn_noclockbuf from the attributes package:

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


Copyright © 2013 Synopsys, Inc.

```
entity simpledff is
   port (q : out bit_vector(7 downto 0);
         d : in bit_vector(7 downto 0);
         clk : in bit );

// No explicit type declaration is necessary
attribute syn_noclockbuf of clk : signal is true;

-- Other code
```

Declaring Attributes

The alternative method is to declare the attributes to explicitly define them. You must do this each time you use an attribute. Here is the syntax for declaring directives and attributes in your code, without referencing the attributes package:

-- *design_unit_declarations*
attribute *attribute_name* **:** *data_type* **;**
attribute *attribute_name* **of** *object* **:** *object_type* **is** *value* **;**

Here is an example using the syn_noclockbuf attribute:

```
entity simpledff is
   port (q : out bit_vector(7 downto 0);
         d : in bit_vector(7 downto 0);
         clk : in bit);

// Explicit type declaration
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk : signal is true;

-- Other code
```

Case Sensitivity

Although VHDL is case-insensitive, directives, attributes, and their values are case sensitive and must be declared in the code using the correct case. This rule applies especially for port names in directives.

For example, if a port in VHDL is defined as GIN, the following code does not work:

```
attribute black_box_tri_pin : string;
attribute black_box_tri_pin of BBDLHS : component is "gin";
```

The following code is correct because the case of the port name is correct:

```
attribute black_box_tri_pin : string;
attribute black_box_tri_pin of BBDLHS : component is "GIN";
```

VHDL Synthesis Examples

This section describes the VHDL examples that are provided with the synthesis tool. The topics include:

- Combinational Logic Examples, on page 965
- Sequential Logic Examples, on page 966

Combinational Logic Examples

The following combinational logic synthesis examples are included in the *installDirectory*/examples/vhdl/common_rtl/combinat directory:

- Adders
- ALU
- Bus Sorter (illustrates using procedures in VHDL)
- 3-to-8 Decoders
- 8-to-3 Priority Encoders
- Comparator
- Interrupt Handler (coded with an if-then-else statement for the desired priority encoding)
- Multiplexers (concurrent signal assignments, case statements, or if-then-else statements can be used to create multiplexers; the synthesis tool automatically creates parallel multiplexers when the conditions in the branches are mutually exclusive)
- Parity Generator
- Tristate Drivers

Copyright © 2013 Synopsys, Inc.

Sequential Logic Examples

The following sequential logic synthesis examples are included in the *install-Directory*/examples/vhdl/common_rtl/sequentl directory:

- Flip-flops and level-sensitive latches
- Counters (up, down, and up/down)
- Register file
- Shift register
- State machines

For additional information on synthesizing flip-flops and latches, see:

- *Creating Flip-flops and Registers Using VHDL Processes*, on page 883
- *Level-sensitive Latches Using Concurrent Signal Assignments*, on page 887
- *Level-sensitive Latches Using VHDL Processes*, on page 888
- *Asynchronous Sets and Resets*, on page 893
- *Synchronous Sets and Resets*, on page 894

Copyright © 2013 Synopsys, Inc.

PREP VHDL Benchmarks

PREP (Programmable Electronics Performance) Corporation distributes benchmark results that show how FPGA vendors compare with each other in terms of device performance and area.

The following PREP benchmarks are included in the *install-Directory*/examples/vhdl/common_rtl/prep directory:

- PREP Benchmark 1, Data Path (prep1.vhd)
- PREP Benchmark 2, Timer/Counter (prep2.vhd)
- PREP Benchmark 3, Small State Machine (prep3.vhd)
- PREP Benchmark 4, Large State Machine (prep4.vhd)
- PREP Benchmark 5, Arithmetic Circuit (prep5.vhd)
- PREP Benchmark 6, 16-Bit Accumulator (prep6.vhd)
- PREP Benchmark 7, 16-Bit Counter (prep7.vhd)
- PREP Benchmark 8, 16-Bit Pre-scaled Counter (prep8.vhd)
- PREP Benchmark 9, Memory Map (prep9.vhd)

The source code for the benchmarks can be used for design examples for synthesis or for doing your own FPGA vendor comparisons.

Copyright © 2013 Synopsys, Inc.

CHAPTER 12

VHDL 2008 Language Support

This chapter describes support for the VHDL 2008 standard in the Synopsys FPGA synthesis tools. For information on the VHDL standard, see Chapter 11, *VHDL Language Support* and the IEEE 1076™-2008 standard. The following sections describe the current level of VHDL 2008 support.

Features

The support for VHDL 2008 includes:

- Operators, on page 970
- Unconstrained Data Types, on page 973
- Unconstrained Record Elements, on page 975
- Predefined Functions, on page 976
- Packages, on page 978
- Generics in Packages, on page 981
- Context Declarations, on page 981
- Case-generate Statements, on page 982
- Else/elsif Clauses, on page 985
- Syntax Conventions, on page 985

Copyright © 2013 Synopsys, Inc.

Operators

VHDL 2008 includes support for the following operators:

- Logical Reduction operators – the logic operators: and, or, nand, nor, xor, and xnor can now be used as unary operators
- Condition operator (??) – converts a bit or std_ulogic value to a boolean value
- Matching Relational operators (?=, ?/=, ?<, ?<=, ?>, ?>=) – similar to the normal relational operators, but return bit or std_ulogic values in place of Boolean values
- Bit-string literals – bit-string characters other than 0 and 1 and string formats including signed/unsigned and string length

Logical Reduction Operators

The logical operators and, or, nand, nor, xor, and xnor can be used as unary operators.

Example – Logical Operators

Condition Operator

The condition operator (??) converts a bit or std_ulogic value to a boolean value. The operator is implicitly applied in a condition where the expression would normally be interpreted as a boolean value as shown in the if statement in the two examples below.

Example – VHDL 2008 Style Conditional Operator

Example – VHDL 1993 Style Conditional Operator

In the VHDL 2008 example, the statement

```
if sel then
```

is equivalent to:

```
if (?? sel) then
```

The implicit use of the ?? operator occurs in the following conditional expressions:

- after if or elsif in an if statement
- after if in an if-generate statement
- after until in a wait statement
- after while in a while loop
- after when in a conditional signal statement
- after assert in an assertion statement
- after when in a next statement or an exit statement

Matching Relational Operators

The matching relational operators return a bit or std_ulogic result in place of a Boolean.

Example – Relational Operators

Bit-string Literals

Bit-string literal support in VHDL 2008 includes:

- Support for characters other than 0 and 1 in the bit string, such as X or Z.

  For example:

  `X"Z45X"` is equivalent to `"ZZZZ01000101XXXX"`

  `B"0001-"` is equivalent to `"0001-"`

  `O"75X"` is equivalent to `"111101XXX"`

- Optional support for a length specifier that determines the length of the string to be assigned.

  Syntax: [*length*] *baseSpecifier* **"***bitStringvalue***"**

  For example:

  `12X"45"` is equivalent to `"000001000101"`

Copyright © 2013 Synopsys, Inc.

`5O"17"` is equivalent to `"01111"`

- Optional support for a signed (S) or unsigned (U) qualifier that determines how the bit-string value is expanded/truncated when a length specifier is used.

  Syntax: [*length*] **S**|**U** *baseSpecifier* **"***bitStringvalue***"**

  For example:

  `12UB"101"` is equivalent to `"000000000101"`

  `12SB"101"` is equivalent to `"111111111101"`

  `12UX"96"` is equivalent to `"000010010110"`

  `12SX"96"` is equivalent to `"111110010110"`

- Additional support for a base specifier for decimal numbers (D). The number of characters in the bit string can be determined by using the expression $(\log_2 n)+1$; where n is the decimal integer.

  Syntax: [*length*] **D "***bitStringvalue***"**

  For example:

  `D"10"` is equivalent to `"1010"`

  `10D"35"` is equivalent to `"0000100011"`

For complete descriptions of bit-string literal requirements, see the VHDL 2008 LRM.

Unconstrained Data Types

VHDL 2008 allows the element types for arrays and the field types for records to be unconstrained. In addition, VHDL 2008 includes support for partially constrained subtypes in which some elements of the subtype are constrained, while others elements are unconstrained. Specifically, VHDL 2008:

- Supports unconstrained arrays of unconstrained arrays (i.e., element types of arrays can be unconstrained)
- Supports the VHDL 2008 syntax that allows a new subtype to be declared that constrains any element of an existing type that is not yet constrained
- Supports the ‘element attribute that returns the element subtype of an array object
- Supports the new ‘subtype attribute that returns the subtype of an object

Example – Unconstrained Element Types

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

package myTypes is
   type memUnc is array (natural range <>) of std_logic_vector;
   function summation(varx: memUnc) return std_logic_vector;
end package myTypes;

package body myTypes is
   function summation(varx: memUnc) return std_logic_vector is
      variable sum: varx’element;
   begin
      sum := (others => '0');
         for I in 0 to varx’length - 1 loop
            sum := sum + varx(I);
         end loop;
      return sum;
   end function summation;
end package body myTypes;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use work.myTypes.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity sum is
   port (in1: memUnc(0 to 2)(3 downto 0);
         out1: out std_logic_vector(3 downto 0) );
end sum;

architecture uncbehv of sum is
begin
   out1 <= summation(in1);
end uncbehv;
```

Example – Unconstrained Elements within Nested Arrays

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

package myTypes is
   type t1 is array (0 to 1) of std_logic_vector;
   type memUnc is array (natural range <>) of t1;
   function doSum(varx: memUnc) return std_logic_vector;
end package myTypes;

package body myTypes is
   function addVector(vec: t1) return std_logic_vector is
      variable vecres: vec'element := (others => '0');
   begin
      for I in vec'Range loop
         vecres := vecres + vec(I);
      end loop;
      return vecres;
   end function addVector;
   function doSum(varx: memUnc) return std_logic_vector is
      variable sumres: varx'element'element;
   begin
      if (varx'length = 1) then
         return addVector(varx(varx'low));
      end if;
      if (varx'Ascending) then
         sumres := addVector(varx(varx'high)) +
            doSum(varx(varx'low to varx'high-1));
      else
         sumres := addVector(varx(varx'low)) +
            doSum(varx(varx'high downto varx'low+1));
      end if;
      return sumres;
   end function doSum;
end package body myTypes;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use work.myTypes.all;

entity uncfunc is
   port (in1: in memUnc(1 downto 0)(open)(0 to 3);
         in2: in memUnc(0 to 2)(open)(5 downto 0);
         in3: in memUnc(3 downto 0)(open)(2 downto 0);
         out1: out std_logic_vector(5 downto 0);
         out2: out std_logic_vector(0 to 3);
         out3: out std_logic_vector(2 downto 0) );
end uncfunc;

architecture uncbehv of uncfunc is
begin
   out1 <= doSum(in2);
   out2 <= doSum(in1);
   out3 <= doSum(in3);
end uncbehv;
```

Unconstrained Record Elements

VHDL 2008 allows element types for records to be unconstrained (earlier versions of VHDL required that the element types for records be fully constrained). In addition, VHDL 2008 supports the concept of partially constrained subtypes in which some parts of the subtype are constrained, while others remain unconstrained.

Example – Unconstrained Record Elements

```
library ieee;
use ieee.std_logic_1164.all;

entity unctest is
   port (in1: in std_logic_vector (2 downto 0);
         in2: in std_logic_vector (3 downto 0);
         out1: out std_logic_vector(2 downto 0) );
end unctest;
```


Copyright © 2013 Synopsys, Inc.

```
architecture uncbehv of unctest is
   type zRec is record
      f1: std_logic_vector;
      f2: std_logic_vector;
   end record zRec;
subtype zCnstrRec is zRec(f1(open), f2(3 downto 0));
subtype zCnstrRec2 is zCnstrRec(f1(2 downto 0), f2(open));
signal mem: zCnstrRec2;
begin
   mem.f1 <= in1;
   mem.f2 <= in2;
   out1 <= mem.f1 and mem.f2(2 downto 0);
end uncbehv;
```

Predefined Functions

VHDL 2008 adds the minimum and maximum predefined functions. The behavior of these functions is defined in terms of the "<" operator for the operand type. The functions can be binary to compare two elements, or unary when the operand is an array type.

Example – Minimum/Maximum Predefined Functions

```
entity minmaxTest is
   port (ary1, ary2: in bit_vector(3 downto 0);
         minout, maxout: out bit_vector(3 downto 0);
         unaryres: out bit );
end minmaxTest;

architecture rtlArch of minmaxTest is
begin
   maxout <= maximum(ary1, ary2);
   minout <= minimum(ary1, ary2);
   unaryres <= maximum(ary1);
end rtlArch;
```

Copyright © 2013 Synopsys, Inc.

Generic Types

VHDL 2008 introduces several types of generics that are not present in VHDL IEEE Std 1076-1993. These types include generic types, generic packages, and generic subprograms.

Generic Types

Generic types allow logic descriptions that are independent of type. These descriptions can be declared as a generic parameter in both packages and entities. The actual type must be provided when instantiating a component or package.

Example of a generic type declaration:

```
entity mux is
   generic (type dataType);
   port (sel: in bit; za, zb: in dataType; res: out dataType);
end mux;
```

Example of instantiating an entity with a type generic:

```
inst1:  mux generic map (bit_vector(3 downto 0))
   port map (selval,in1,in2,out1);
```

Generic Packages

Generic packages allow descriptions based on a formal package. These descriptions can be declared as a generic parameter in both packages and entities. An actual package (an instance of the formal package) must be provided when instantiating a component or package.

Example of a generic package declaration:

```
entity mux is generic (
   package argpkg is new dataPkg generic map (<>);
);
   port (sel: in bit; za, zb: in bit_vector(3 downto 0);
      res: out bit_vector(3 downto 0) );
end mux;
```


Copyright © 2013 Synopsys, Inc.

Example of instantiating a component with a package generic:

```
package memoryPkg is new dataPkg generic map (4, 16);

...

inst1: entity work.mux generic map (4, 16, argPkg => memoryPkg)
```

Generic Subprograms

Generic subprograms allow descriptions based on a formal subprogram that provides the function prototype. These descriptions can be declared as a generic parameter in both packages and entities. An actual function must be provided when instantiating a component or package.

Example of a generic subprogram declaration:

```
entity mux is
   generic (type dataType; function filter(datain: dataType)
      return dataType);
   port (sel: in bit; za, zb: in dataType; res: out dataType);
end mux;
```

Example of instantiating a component with a subprogram generic:

```
architecture myarch2 of myTopDesign is
   function intfilter(din: integer) return integer is
   begin
      return din + 1;
   end function intfilter;

...

begin
   inst1: mux generic map (integer, intfilter)
      port map (selval,intin1,intin2,intout);
```

Packages

VHDL 2008 includes several new packages and modifies some of the existing packages. The new and modified packages are located in the $LIB/vhd2008 folder instead of $LIB/vhd.

New Packages

The following packages are supported in VHDL 2008:

- fixed_pkg.vhd, float_pkg.vhd, fixed_generic_pkg.vhd, float_generic_pkg.vhd, fixed_float_types.vhd – IEEE fixed and floating point packages
- numeric_bit_unsigned.vhd – Overloads for bit_vector to have all operators defined for ieee.numeric_bit.unsigned
- numeric_std_unsigned.vhd – Overloads for std_ulogic_vector to have all operators defined for ieee.numeric_std.unsigned

String and text I/O functions in the above packages are not be supported. These functions include read(), write(), to_string().

Modified Packages

The following modified IEEE packages are supported with the exception of the new string and text I/O functions (the previously supported string and text I/O functions are unchanged):

- std.vhd – new overloads
- std_logic_1164.vhd – std_logic_vector is now a subtype of std_ulogic_vector; new overloads
- numeric_std.vhd – new overloads
- numeric_bit.vhd – new overloads

Unsupported Packages/Functions

The following packages and functions are not currently supported:

- string and text I/O functions in the new packages
- The fixed_pkg_params.vhd or float_pkg_params.vhd packages, which were temporarily supported to allow the default parameters to be changed for fixed_pkg.vhd and float_pkg.vhd packages, have been obsoleted by the inclusion of the fixed_generic_pkg.vhd or float_generic_pkg.vhd packages.

Copyright © 2013 Synopsys, Inc.

Using the Packages

A switch for VHDL 2008 is located in the GUI on the VHDL panel (Implementation Options dialog box) to enable use of these packages and the ?? operator.

You can also enable the VHDL 2008 packages by including the following command in the compiler options section of your project file:

```
set_option -vhdl2008 1
```

Generics in Packages

In VHDL 2008, packages can include generic clauses. These generic packages can then be instantiated by providing values for the generics as shown in the following example.

Example – Including Generics in Packages

Context Declarations

VHDL 2008 provides a new type of design unit called a context declaration. A context is a collection of library and use clauses. Both context declarations and context references are supported as shown in the following example.

Example – Context Declaration

In VHDL 2008, a context clause cannot precede a context declaration. The following code segment results in a compiler error.

```
library ieee; -- Illegal context clause before a
              -- context declaration
context zcontext is
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;
end context zcontext;
```

Similarily, VHDL 2008 does not allow reference to the library name work in a context declaration. The following code segment also results in a compiler error.

```
context zcontext is
   use work.zpkg.all; -- Illegal reference to library work
                      -- in a context declaration
   library ieee;
   use ieee.numeric_std.all;
end context zcontext;
```


Copyright © 2013 Synopsys, Inc.

VHDL 2008 supports the following two, standard context declarations in the IEEE package:

- IEEE_BIT_CONTEXT
- IEEE_STD_CONTEXT

Case-generate Statements

The case-generate statement is a new type of generate statement incorporated into VHDL 2008. Within the statement, alternatives are specified similar to a case statement. A static (computable at elaboration) select statement is compared against a set of choices as shown in the following syntax:

```
caseLabel: case expression generate
           when choice1 =>
                -- statement list
           when choice2 =>
                -- statement list
           …
           end generate caseLabel;
```

To allow for configuration of alternatives in case-generate statements, each alternative can include a label preceding the choice value (e.g., labels L1 and L2 in the syntax below):

```
caseLabel: case expression generate
           when L1: choice1 =>
                -- statement list
           when L2: choice2 =>
                -- statement list
           …
           end generate caseLabel;
```

Example – Case-generate Statement with Alternatives

```
entity myTopDesign is
   generic (instSel: bit_vector(1 downto 0) := "10");
   port (in1, in2, in3: in bit; out1: out bit);
end myTopDesign;
```

```
architecture myarch2 of myTopDesign is
component mycomp
   port (a: in bit; b: out bit);
end component;

begin
a1: case instSel generate
   when "00" =>
      inst1: component mycomp port map (in1,out1);
   when "01" =>
      inst1: component mycomp port map (in2,out1);
   when others =>
      inst1: component mycomp port map (in3,out1);
   end generate;
end myarch2;
```

Example – Case-generate Statement with Labels for Configuration

```
entity myTopDesign is
generic (selval: bit_vector(1 downto 0) := "10");
   port (in1, in2, in3: in bit; tstIn: in bit_vector(3 downto 0);
         out1: out bit);
end myTopDesign;

architecture myarch2 of myTopDesign is
   component mycomp
      port (a: in bit; b: out bit);
   end component;
begin
a1: case selval generate
   when spec1: "00" | "11"=> signal inRes: bit;
      begin
         inRes <= in1 and in3;
         inst1: component mycomp port map (inRes,out1);
      end;
   when spec2: "01" =>
      inst1: component mycomp port map (in1, out1);
   when spec3: others =>
      inst1: component mycomp port map (in3,out1);
   end generate;
end myarch2;

entity mycomp is
   port (a : in bit;
         b : out bit);
end mycomp;
```


Copyright © 2013 Synopsys, Inc.

```
architecture myarch of mycomp is
begin
   b <= not a;
end myarch;

architecture zarch of mycomp is
begin
   b <= '1';
end zarch;

configuration myconfig of myTopDesign is
for myarch2
   for a1 (spec1)
      for inst1: mycomp use entity mycomp(myarch);
      end for;
   end for;
   for a1 (spec2)
      for inst1: mycomp use entity mycomp(zarch);
      end for;
   end for;
   for a1 (spec3)
      for inst1: mycomp use entity mycomp(myarch);
      end for;
   end for;
end for;
end configuration myconfig;
```

Matching case and select Statements

Matching case and matching select statements are supported – case? (matching case statement) and select? (matching select statement). The statements use the ?= operator to compare the case selector against the case options.

Example – Use of case? Statement

Example – Use of select? Statement

Else/elsif Clauses

In VHDL 2008, else and elsif clauses can be included in if-generate statements. You can configure specific if/else/elsif clauses using configurations by adding a label before each condition. In the code example below, the labels on the branches of the if-generate statement are spec1, spec2, and spec3. These labels are later referenced in the configuration myconfig to specify the appropriate entity/architecture pair. This form of labeling allows statements to be referenced in configurations.

Example – Else/elsif Clauses in If-Generate Statements

Syntax Conventions

The following syntax conventions are supported in VHDL 2008:

- All keyword
- Comment delimiters
- Extended character set

All Keyword

VHDL 2008 supports the use of an all keyword in place of the list of input signals to a process in the sensitivity list.

Copyright © 2013 Synopsys, Inc.

Example – All Keyword in Sensitivity List

Comment Delimiters

VHDL 2008 supports the /* and */ comment-delimiter characters. All text enclosed between the beginning /* and the ending */ is treated as a comment, and the commented text can span multiple lines. The standard VHDL "--" comment-introduction character string is also supported.

Extended Character Set

The extended ASCII character literals (ASCII values from 128 to 255) are supported.

Example – Extended Character Set

Copyright © 2013 Synopsys, Inc.

CHAPTER 13

Utilities

This chapter describes some Synopsys FPGA tools and utilities that help productivity and smooth out the design process. See the following topics for information:

- sdc2fdc Tcl Shell Command, on page 988
- Altera qsf2sdc Utility, on page 994
- Altera SOPC2Syn Utility, on page 998
- Altera qsf2syn Utility, on page 1003
- Xilinx ucf2sdc Utility, on page 1004
- Xilinx ise2syn Utility, on page 1017
- Xilinx Vivado Tcl Commands, on page 1018
- Synplicity Archive Utility, on page 1025
- SYNCore FIFO Compiler, on page 1026
- SYNCore RAM Compiler, on page 1043
- SYNCore Byte-Enable RAM Compiler, on page 1053
- SYNCore ROM Compiler, on page 1058
- SYNCore Adder/Subtractor Compiler, on page 1064
- SYNCore Counter Compiler, on page 1076
- Encryption Scripts, on page 1081

Copyright © 2013 Synopsys, Inc.

sdc2fdc Tcl Shell Command

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. For details, see:

- sdc2fdc Tcl Shell Command, on page 988
- Examples of sdc2fdc Translation, on page 988
- FPGA Design Constraint (FDC) File, on page 989
- Troubleshooting, on page 991

sdc2fdc Tcl Shell Command

From the Tcl command line, type:

sdc2fdc

For information about how to run this command, see *Converting SDC to FDC*, on page 171.

Examples of sdc2fdc Translation

```
% sdc2fdc

INFO: Translation successful.
See:"D:/bugs/timing_88/clk_prior/scratch/FDC_constraints/rev_2/top_translated.fdc"
Replace your current *.sdc files with this one.

INFO: Automatically updating your project to reflect the new constraint file(s)
Do "Ctrl+S" to save the new settings.
```

```
% sdc2fdc

ERROR: Bad -from list for define_false_path: {my_inst}
Missing qualifier(s) (i: p: n: ...)
ERROR: Translation problems were found.
See:"D:/bugs/timing_88/clk_prior/scratch/FDC_constraints/rev_2/top_translate.log"
for details.
```

```
_translate.log
```

```
ERROR:  Bad -from list for define_false_path  {my_inst}
        Missing qualifier(s) (i: p: n: ...)

"define_false_path -from {my_inst} -to i:abc.def.g_reg -through {n:bar}"
Synplicity SDC source file: D:/bugs/timing_88/clk_prior/scratch/top.sdc.
Line number: 79
```

FPGA Design Constraint (FDC) File

FPGA Design Constraints (FDC) are input constraints for the FPGA synthesis tool. The file format is migrating from legacy FPGA timing constraints (for example, define_clock, define_input_delay, and define_false_path) to *Synopsys SDC Standard*-compliant timing constraints (for example, create_clock, set_input_delay, and set_false_path). FDC combines the FPGA design constraints (for example, define_attribute, define_scope_collection, and define_io_standard) with the Synopsys standard timing constraints.

The FDC constraint file is generated after running sdc2fdc that translates legacy FPGA design constraints (SDC) to Synopsys FPGA design constraints (FDC). This file is divided into two sections. The first section contains the FPGA design constraints that are valid (for example, define_scope_collection and define_attribute) and the legacy timing constraints that are not translated because they were specified with -disable. The second section contains the timing constraints that were translated into Synopsys SDC standard format. This file also provides the following:

- Each source `sdc` file has its separate subhead.
- Each compile point is treated as a top level, so its `sdc` file has its own _translated.fdc file.
- The translator adds the naming rule, set_rtl_ff_names, so that the synthesis tool knows these constraints are not from the Synopsys Design Compiler.

The following example shows the contents of the FDC file.

```
##############################################################################
####This file contains constraints from Synplicity SDC files that have been
####translated into Synopsys FPGA Design Constraints (FDC.
####Translated FDC output file:
####D:/bugs/timing_88/clk_prior/scratch/FDC_constraints/rev_2/top_translated.fdc
####Source SDC files to the translation:
####D:/bugs/timing_88/clk_prior/scratch/top.sdc
##############################################################################
```


Copyright © 2013 Synopsys, Inc.

```
##############################################################################
####Source SDC file to the translation:
####D:/bugs/timing_88/clk_prior/scratch/top.sdc
##############################################################################

#Legacy constraint file
#C:\Clean_Demos\Constraints_Training\top.sdc
#Written on Mon May 21 15:58:35 2012
#by Synplify Pro, Synplify Pro Scope Editor
#
#Collections
#
define_scope_collection all_grp {define_collection \
        [find -inst {i:FirstStbcPhase}] \
        [find -inst {i:NormDenom[6:0]}] \
        [find -inst {i:NormNum[7:0]}] \
        [find -inst {i:PhaseOut[9:0]}] \
        [find -inst {i:PhaseOutOld[9:0]}] \
        [find -inst {i:PhaseValidOut}] \
        [find -inst {i:ProcessData}] \
        [find -inst {i:Quadrant[1:0]}] \
        [find -inst {i:State[2:0]}] \
        }
#
#Clocks

#define_clock -disable -name {clkc} -virtual -freq 150 -clockgroup default_clkgroup_1

#Clock to Clock
#
#
#Inputs/Outputs
#
define_input_delay -disable {b[7:0]} 2.00 -ref clka:r
define_input_delay -disable {c[7:0]} 0.20 -ref clkb:r
define_input_delay -disable {d[7:0]} 0.30 -ref clkb:r
define_output_delay -disable {x[7:0]} -improve 0.00 -route 0.00
define_output_delay -disable {y[7:0]} -improve 0.00 -route 0.00
#
#Registers
#
#
#Multicycle Path
#
#
#False Path
#
#
define_false_path -disable -from {i:x[1]}
#

#Path Delay
#
#
#Attributes
#
define_io_standard -default_input -delay_type input syn_pad_type {LVCMOS_33}#

#I/O standards
#
```

```
#
#Compile Points
#
#
#Other Constraints

##############################################################################
#SDC compliant constraints translated from Legacy Timing Constraints
##############################################################################
#
set_rtl_ff_names {#}

create_clock -name {clka} [get_ports {clka}] -period 10 -waveform {0 5.0}
create_clock -name {clkb} [get_ports {clkb}] -period 6.666666666666667
    -waveform {0 3.3333333333333335}
set_input_delay -clock [get_clocks {clka}] -clock_fall -
add_delay 0.000 [all_inputs]
set_output_delay -clock [get_clocks {clka}] -add_delay 0.000 [all_outputs]
set_input_delay -clock [get_clocks {clka}] -
add_delay 2.00 [get_ports {a[7:0]}]
set_input_delay -clock [get_clocks {clka}] -add_delay 0 [get_ports {rst}]
set mcp 4
set_multicycle_path $mcp -start  \
     -from \
       [get_ports \
       {a* \
       b*} \
       ] \
     -to \
       [find -seq -hier {q?[*]} ]

set_multicycle_path 3 -end  \
     -from \
       [find -seq {*y*.q[*]} ]

set_clock_groups -name default_clkgroup_0 -asynchronous \
    -group [get_clocks {clka dcm|clk0_derived_clock dcm|
    clk2x_derived_clock dcm|clk0fx_derived_clock}]
set_clock_groups -name default_clkgroup_1 -asynchronous \
    -group [get_clocks {clkb}]
```

Troubleshooting

The following table contains common types of error messages you might encounter when running the sdc2fdc Tcl shell command. The error messages include descriptions of how you can resolve these problems.

Problem and Solution Examples

Cannot translate a translated file	Remove/disable D:FDC_constraints/rev_FDC/top_translated.fdc from the current implementation.
No active constraint files	Add/enable one or more SDC constraint files.

Copyright © 2013 Synopsys, Inc.

Clock not translated	• Add clock object qualifier (p: n: ...) for: "define_clock -name {clka {clka} -period 10 -clockgroup {default_clkgroup_0}" Synplicity_SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 32 • Specify -name for: "define_clock {p:clkb} -period 20 -clockgroup {default_clkgroup_1}" Synplicity SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 33
Bad -from list for define_multicycle_path {a* b*}	Missing qualifier(s) (i: p: n: ...) "define_multicycle_path 4 -from {a* b*} -to $fdc_cmd_0 -start" Synplicity SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 76
Bad -to list for define_multicycle_path {i: *y* .q[*] p:ena}	Mixing of object types not permitted "define_multicycle_path -to {i:*y*.q[*] p:ena} 3" Synplicity SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 77
Bad -from list for define_multicycle_path {i:*y* .q[*] p:ena enab}	Mixing of object types and missing qualifiers not permitted "define_multicycle_path -from {i:*y*.q[*] p:ena enab} 3" Synplicity SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 77
No period or frequency found	Default 1000. "create_clock -name {clkb} {p:clkb} -period 1000 -waveform {0 500.0}" Synplicity SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 33
Must specify both -rise and -fall or neither	"create_clock -name {clka} {p:clka} -period 10 -rise 5 -clockgroup {default_clkgroup_0" Synplicity SDC source file: D:.../clk_prior/scratch/top.sdc. Line number: 32

Besides these error messages, make sure that your files have read/write permissions set properly and there is sufficient disk space. Fix any issues in the SDC source file and rerun the sdc2fdc command.

Copyright © 2013 Synopsys, Inc.

Batch Mode

If you run sdc2fdc -batch, then the following occurs:

- The two `Clock not translated` messages in the table above are not generated.
- When the translation is successful, the SDC file is disabled and the FDC file is enabled and saved automatically in the Project file.

  However, if the -batch option is *not* used and the translation is successful, then the SDC file is disabled and the FDC file is enabled but *not* automatically saved in the Project file. A message is generated that states this condition in the Tcl shell window.

Copyright © 2013 Synopsys, Inc.

Altera qsf2sdc Utility

A standard logic synthesis flow requires that timing constraints be applied during synthesis and any required physical constraints be applied during the place-and-route stage. However, for a physical synthesis flow, logic synthesis is combined with place-and-route. Therefore, a unified set of timing and physical constraints is required.

The qsf2sdc utility facilitates the conversion of a Quartus settings file (`qsf`) into a Synopsys FPGA `sdc` constraints file. Run the utility from any shell window. For a step-by-step procedure, see *Translating Altera QSF Constraints*, on page 193 in the *User Guide*. The following provide syntax and other details:

- QSF2SDC Syntax, on page 994
- Supported QSF Constraints, on page 995
- Example of Translated QSF Constraints, on page 996

QSF2SDC Syntax

Here is the syntax:

install_directory/**bin/qsf2sdc -iqsf** *input_qsf_file* **-osdc** *output_sdc_file* [**-oqsf** *output_residue_qsf_file*] [**-all**] [**-silent**]

install_directory	Defines the path of the directory containing the Synplicity software
-iqsf	Defines the input Quartus settings file (`qsf`)
-osdc	Defines the output constraint file (`sdc`)
-oqsf	Defines the residual constraints that cannot be converted and must remain in a Quartus settings file (`qsf`). This file is optional.
-all	Converts any instances with location assignments. By default, only pin location assignments and IO standards are automatically converted.
-silent	Suppresses all the successfully translated (#Supported) and unsuccessfully translated (#Unsupported) messages in the `sdc` file.

Copyright © 2013 Synopsys, Inc.

Assumptions

Keep in mind the following limitations:

- Quartus constraints are specified completely on a single line without spanning multiple lines.
- Currently, only two Quartus constraints are translated by this utility: See *Supported QSF Constraints*, on page 995 for more information.

Supported QSF Constraints

This section summarizes the two QSF constraints currently translated by the qsf2sdc script, pin location and I/O standards.

Pin Location Constraints

The QSF syntax for pin locations is as follows:

set_location_assignment -to *pin pin_value*

The following examples show QSF pin location syntax and their SDC equivalents. See *Example of Translated QSF Constraints*, on page 996 for a file example.

QSF Syntax	SDC Syntax
`set_location_assignment -to CLK PIN_P23`	`define_attribute {CLK} syn_loc {PIN_P23}`
`set_location_assignment PIN_P23 -to CLK`	`define_attribute {CLK} syn_loc {PIN_P23}`
`set_location_assignment PIN_AA11 -to TXWRADDR[10]`	`define_attribute {TXWRADDR[10]} syn_loc {PIN_AA11}`
`set_location_assignment -to TXWRADDR[10] PIN_AA11`	`define_attribute {TXWRADDR[10]} syn_loc {PIN_AA11}`

Copyright © 2013 Synopsys, Inc.

I/O Standard Constraints

This is the QSF syntax for I/O standards:

set_instance_assignment -name IO_STANDARD -to *instance_name IO_STD_value*

The following examples show QSF I/O standard constraints and their SDC equivalents.See *Example of Translated QSF Constraints*, on page 996 for a file example.

QSF Syntax	SDC Syntax
`set_instance_assignment -name IO_STANDARD -to CLK LVTTL`	`define_attribute {CLK} syn_pad_type {LVTTL}`
`set_instance_assignment -name IO_STANDARD LVTTL -to CLK`	`define_attribute {CLK} syn_pad_type {LVTTL}`
`set_instance_assignment -name IO_STANDARD -to TXWRADDR[10] LVTTL`	`define_attribute {TXWRADDR[10]} syn_pad_type {LVTTL}`
`set_instance_assignment -name IO_STANDARD LVTTL -to TXWRADDR[10]`	`define_attribute {TXWRADDR[10]} syn_pad_type {LVTTL}`

Example of Translated QSF Constraints

The following example illustrates how the QSF constraints are translated.

Original QSF Constraint File

```
set_location_assignment -to TXWRADDR[10] PIN_AA11
set_location_assignment PIN_AA11 -to TXWRADDR[10]
set_location_assignment -to CLK PIN_P23
set_location_assignment PIN_P23 -to CLK
set_instance_assignment -name IO_STANDARD LVTTL -to TXWRADDR[10]
set_instance_assignment -name IO_STANDARD LVTTL -to CLK
set_instance_assignment -name IO_STANDARD -to CLK LVTTL
```

Copyright © 2013 Synopsys, Inc.

Results for SDC Constraints

After you run the qsf2sdc utility, the .sdc file contains these constraints:

```
define_attribute {TXWRADDR[10]} syn_loc {PIN_AA11}
#SUPPORTED# set_location_assignment -to TXWRADDR[10] PIN_AA11

define_attribute {TXWRADDR[10]} syn_loc {PIN_AA11}
#SUPPORTED# set_location_assignment PIN_AA11 -to TXWRADDR[10]

define_attribute {CLK} syn_loc {PIN_P23}
#SUPPORTED# set_location_assignment -to CLK PIN_P23

define_attribute {CLK} syn_loc {PIN_P23}
#SUPPORTED# set_location_assignment PIN_P23 -to CLK

define_attribute {TXWRADDR[10]} syn_pad_type {LVTTL}
#SUPPORTED# set_instance_assignment -name IO_STANDARD LVTTL -to
TXWRADDR[10]

define_attribute {CLK} syn_pad_type {LVTTL}
#SUPPORTED# set_instance_assignment -name IO_STANDARD LVTTL -to
CLK

define_attribute {CLK} syn_pad_type {LVTTL}
#SUPPORTED# set_instance_assignment -name IO_STANDARD -to CLK
LVTTL
```


Copyright © 2013 Synopsys, Inc.

Altera SOPC2Syn Utility

The sopc2syn utility is the batch mode equivalent of the Import IP->Import Altera SOPC Project command and lets you include subsystems created with the Altera SOPC Builder in your design.

The sopc2syn utility reads an SOPC Builder project file (`ptf`) and script file (`scr`), and uses information extracted from them to construct a Synplify project file. In the process, the utility reads several other SOPC Builder files (`qpf`, `qsf`, `bdf`, `bsf`, etc.) and uses them to add definition and constraint files to the Synplify project file it generates.

See the following for details:

- Features of the sopc2syn Utility, on page 998
- Syntax Options for the sopc2syn Utility, on page 999

Features of the sopc2syn Utility

This section describes salient features of the sopc2syn utility:

Input files	The utility reads the following files to extract required information and construct a synthesis project: • `ptf` file • `scr` file (script file) • `qsf`, `edf`, and `bsf` files from the SOPC Builder project
Output files	The utility generates the following output files: • Synplify project file (`prj`) • Synplify Constraint file (`sdc`) file • Core wrapper file for black boxes • Directories such as the unused directory containing the HDL files used in the project, and the quartus_synthesis directory which contains the HDL files for the encrypted cores and forced black boxes. • PAR directory files: `qsf`, `qof`, `bdf`, `bsf`

Copyright © 2013 Synopsys, Inc.

Initialization files	The initialization file for sopc2syn is called sopc2syn.ini. This file can be located in two places: • *synplifyInstallDirectory*/lib/sopc2syn.ini • *SOPC_Builder_proj_dir*/sopc2syn.ini The `ini` files contain default settings for the utility. The utility reads the files before each synthesis run and propagates the settings to the synthesis project file it generates. Currently, the only supported settings are the project file options. To set a particular project option for the project file, add that option to one of the `ini` files. If you have two `ini` files, the file in *SOPC_Builder_proj_dir* overrides the one in *synplifyInstallDirectory*. Use this fact to manage project settings. Put general project settings for all Synplify projects generated from sopc2syn.exe in *synplifyInstallDirectory*/lib/sopc2syn.ini. Put project settings for specific projects in the project directory `ini` file.
sopc2syn Options	See *Syntax Options for the sopc2syn Utility*, on page 999.

Syntax Options for the sopc2syn Utility

This section describes the options to invoke the sopc2syn utility from the command line. This is the syntax for the command:

sopc2syn
- **-ptf** *sopcProjectFileName*
- **-clearbox 1|0**
- **-hdl_lang vhdl|verilog**
- **-force_wbox** *fileName|string*
- **-wrap 1|0**
- **-log** *logfileName*
- **-quiet**
- **-noexit**
- **-verbose 0|1|2|3**
- **-help**

Example

installDirectory`/bin/sopc2syn -ptf` *sopc_builder_prj_dir*/*system*`.ptf -log` *sopc_builder_prj_dir*/*system*`.log -verbose 3`

Copyright © 2013 Synopsys, Inc.

The following table describes the options for sopc2syn utility.

-ptf *sopcProjectFileName*	Specifies the name of the SOPC Builder project (`ptf`) file.
-clearbox 1|0	This is an advanced option that uses the Synplify Clearbox flow for encrypted cores. • 0 turns the Clearbox flow off. This adds black box wrapper files for encrypted cores to the project and add the core files to PAR directory. • 1 turns the Clearbox flow on. This adds black box wrapper files for encrypted cores to the project. It also adds the core HDL files to the project, but only uses them for P&R. This is the recommended setting and the default. See *SOPC2syn Black Boxes*, on page 1002, *Encrypted HDL Files for SOPC2syn Black Boxes*, on page 1001, and *SOPC2syn White Boxes*, on page 1002.
-hdl_lang vhdl|verilog	This basic option sets the preferred HDL language in case of duplicate files. Verilog is the default.
-force_wbox *fileName|string*	This is an advanced option that forces listed cores to be treated as white boxes. You can specifies the cores in a file (*fileName*), or in a comma-separated string of core names (*string*). The -clearbox value determines how the core is processed. • If -clearbox is 1,the white box goes through the Clearbox flow. • If -clearbox is 0, the white box does not go through the Clearbox flow, and is treated as a black box. See *SOPC2syn Black Boxes*, on page 1002 and *SOPC2syn White Boxes*, on page 1002.
-wrap 1|0	This is an advanced option that enables or disables the generation of missing core wrappers. • 1 generates missing core wrappers. • 0 does not generate missing core wrappers. This is the recommended setting and the default.

Copyright © 2013 Synopsys, Inc.

-log *logfileName*	This is an advanced option that converts the PTF project file that writes out all the messages to the logfile. If you specify -log *logfileName*, the utility writes out all the messages to the specified file. STDOUT prints important messages such as the version, command options, error messages, and summary information on STDOUT and redirects all STDOUT messages to the log file. If you specify the -quiet option, the utility does not write any messages to STDOUT.
-quiet	This is an advanced option that suppresses messages from being written to the STDOUT.
-noexit	This is a debug option that writes out the project file even if there are errors in translation. See *SOPC2syn Debug Mode*, on page 1002.
-verbose 0|1|2|3	This is a debug option that controls the messages reported in the log file during debug or normal runs. • 0 writes out the important messages. This is the default. • 1 reports all the messages reported by -verbose 0, and also reports all the files read during translation. • 2 reports all messages reported by -verbose 1, as well as the lines processed from each of the files read during translation and the number of lines processed. • 3 reports all messages reported by -verbose 2, as well as more details on the state of run. See *SOPC2syn Debug Mode*, on page 1002.
-help	This basic option prints the help.

Encrypted HDL Files for SOPC2syn Black Boxes

If the `ptf` file points to an encrypted IP core definition file, one of the following happens:

- If you specified -clearbox 1, the utility adds black box wrapper files for encrypted cores to the projects. It also adds the HDL files for the encrypted cores to the project, but only uses them for P&R. It issues a warning message for encrypted cores with a black box.
- If you specified -clearbox 0, the utility adds black box wrapper files for encrypted cores to the project and copies the encrypted core files to the P&R directory.

Copyright © 2013 Synopsys, Inc.

SOPC2syn Black Boxes

If you want to treat a core as a black box during synthesis, specify the -force_wbox option with a list of cores to be black-boxed, and set -clearbox 0.

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.

SOPC2syn White Boxes

If you want a core to be a white box during synthesis, specify the -force_wbox option with list of cores to be white-boxed, and set -clearbox 1.

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

SOPC2syn Debug Mode

You can use the following options to debug translation issues:

Specify the -noexit option if you want the utility to write out the project file, even if there is an error.

Use the -verbose option to better control the reporting of messages. All messages reported by these options are written to the sopc2syn.log file.

Copyright © 2013 Synopsys, Inc.

Altera qsf2syn Utility

The qsf2syn utility takes a Quartus project and converts it into a synthesis project. You can then run this project through logic synthesis or physical synthesis.

You can run this utility from the GUI or as a Tcl command:

- GUI:
  Select the Import->Import Altera QSF Project command.
- Tcl command:
  qsf2syn

Syntax

The syntax for the qsf2syn Tcl command is as follows:

qsf2syn [**-qsf** *value*] [**-synproject** *value*] [**-constraints_only**]

-qsf *value*	The Altera `qsf` file to import.
-synproject *value*	The name and path to the synthesis project to be created.
-constraints_only	Lets you continue the conversion of the project from the previous point when you have to debug and fix errors. See *Importing Quartus Projects*, on page 1033 for information on how to use it.

For detailed information about using the qsf2syn utility, see *Importing Projects from Quartus*, on page 1033.

Copyright © 2013 Synopsys, Inc.

Xilinx ucf2sdc Utility

The ucf2sdc functionality is the basis for the Project->Convert Vendor Constraints command, and converts logic synthesis projects to Synplify Premier projects for synthesis. Some parts of it, like the translation of constraints, is also used by other utilities, like ise2syn.

For more information about this functionality, see the following:

- ucftosdc Tcl Command, on page 1004
- UCF Supported Constraints Summary, on page 1005
- Examples of Translated UCF Constraints, on page 1006
- Unsupported UCF Constraints, on page 1016
- Converting and Using Xilinx UCF Constraints, on page 200 in the *User Guide*

ucftosdc Tcl Command

This command converts a Synplify Pro project to a Synplify Premier project for physical synthesis. It is the Tcl equivalent of the Project->Convert Vendor Constraints command; it is not the legacy command-line ucf2sdc_old utility.

For information about how to use the ucf2sdc functionality, see *Converting and Using Xilinx UCF Constraints*, on page 200 in the *User Guide*.

Syntax

ucftosdc [**-ucf_files** *ucfFileList*] [**-ncf_files** *ncfFileList*] [**-xcf_files** *xcfFileList*] **-project_name** *projName* [**-constraint_chk**]

-ucf_files *ucfFileList*	List of ucf constraint files to be converted
-ncf_files *ncfFileList*	List of ncf constraint files to be converted
-xcf_files *xcfFileList*	List of xcf constraint files to be converted
-project_name *projName*	Name for the new Synplify Premier project
-constraint_chk	Run constraint checker on the converted project

Copyright © 2013 Synopsys, Inc.

UCF Supported Constraints Summary

This section provides a summary of the UCF constraints translated by the ucf2sdc utility. Refer to the Xilinx documentation for syntax details.

Constraint	Description
Grouping Constraints	The following constraints specify how objects are grouped: • TNM is a basic grouping constraint used to identify the elements that make up a group to be used in a timing specification. TNM tags specific FFs, RAMs, LATCHES, PADS and MULTS as members of a group. The TNM constraint can be attached to a net, an element pin, or a macro. See *TNM Constraint Translation Example*, on page 1007 for an example. • TNM_NET is a basic grouping constraint. It identifies the elements that make up a group to be used in a timing specification. TNM_NET can only be used with nets. See *TNM_NET Constraint Translation Example*, on page 1008 for an example. • TIMEGRP is a basic grouping constraint. In addition to naming groups with the TNM identifier, you can also define groups in terms of other groups. A group can be created which is a combination of existing groups by defining a TIMEGRP constraint. See *TIMEGRP Constraint Translation Example*, on page 1008 for an example.
LOC	These are basic placement constraints and a synthesis constraints that defines where a design element can be placed within an FPGA. It specifies the absolute placement of a design element on the FPGA die, as a single location, a range of locations, or a list of locations. You can also use it to specify an area in which to place a design element or group of design elements. See *LOC Constraint Translation Examples*, on page 1009 for examples of single-location, multiple-location, and range-of-locations LOC constraints.
MAX DELAY	MAXDELAY FROM-TO-THRU is an advanced timing constraint, and is associated with the PERIOD constraint of the high or low time. From synchronous paths, a FROM-TO-THRU constraint only controls the setup path, not the hold path. This constraint applies to a specific path that begins at a source group, passes through intermediate points, and ends at a destination group. The source and destination groups can be either user or predefined groups. You must define an intermediate path using TPTHRU before using THRU. See *MAXDELAY Constraint Translation Examples*, on page 1010 for examples.

Copyright © 2013 Synopsys, Inc.

Constraint	Description
OFFSET	OFFSET constraints are basic timing constraints that specify the timing relationship between an external clock and its associated data-in or data-out pin. You use OFFSET only for pad-related signals. Do not use it to extend the arrival time specification method for the internal signals in a design. This constraint allows you to • Calculate whether a setup time is being violated at a flip-flop whose data and clock inputs are derived from external nets. • Specify the delay of an external output net derived from the Q output of an internal flip-flop being clocked from an external device pin. See *OFFSET Constraint Translation Examples*, on page 1012 for examples, and *Unsupported UCF Constraints*, on page 1016 for information about currently unsupported constraints.
PERIOD	Basic timing constraint and synthesis constraint that specifies a clock period and checks the timing between all synchronous elements within the clock domain, as defined in the destination element group. This constraint is attached to the clock net. The PERIOD constraint is applied to the clock net in the UCF file. See *PERIOD Constraint Translation Examples*, on page 1013 for examples.
PROP	Property constraints/attributes are attached to a net, instance, or globally for P&R tool to use that information for proper configuration of that particular net, instance or entire design.We can also specify an area in which to place a design element or group of design elements. See *PROP Constraint Translation Examples*, on page 1014 for examples of general, instantiated RAM, IOB, and DCM PROP constraints.
TIG	TIG (Timing IGnore) is a basic timing constraint and a synthesis constraint. It causes paths that fan forward from the point of application (of TIG) to be treated as if they do not exist (for the purposes of the timing model) during implementation.See *TIG Constraint Examples*, on page 1015 for examples.

Examples of Translated UCF Constraints

See the following for examples of translated constraints:

- TNM Constraint Translation Example, on page 1007
- TNM_NET Constraint Translation Example, on page 1008

- TIMEGRP Constraint Translation Example, on page 1008
- LOC Constraint Translation Examples, on page 1009
- MAXDELAY Constraint Translation Examples, on page 1010
- OFFSET Constraint Translation Examples, on page 1012
- PERIOD Constraint Translation Examples, on page 1013
- PROP Constraint Translation Examples, on page 1014
- TIG Constraint Examples, on page 1015

TNM Constraint Translation Example

UCF Syntax	Converted SDC Constraint
NET "clk1" TNM = "mytn1"; NET "clk2" TNM = "mytn1"; TIMESPEC "TS_mytn1" = PERIOD "mytn1" 5 ns;	define_scope_collection mytn1__0_0 {find -net clk1} define_scope_collection mytn1__0_fr {expand -hier -from $mytn1__0_0} define_scope_collection mytn1 {find -seq * -in $mytn1__0_fr} define_scope_collection mytn1__1_0 {find -net clk2} define_scope_collection mytn1__1_fr {expand -hier -from $mytn1__1_0} define_scope_collection mytn1__1 {find -seq * -in $mytn1__1_fr} define_scope_collection mytn1 {c_union $mytn1 $mytn1__1} define_path_delay -to {{$mytn1}} -max 5.000 See *Predefined Groups*, on page 1008 for the syntax for predefined groups.

Copyright © 2013 Synopsys, Inc.

TNM_NET Constraint Translation Example

UCF Syntax	Converted SDC Constraint
NET "clk1" TNM_NET = FFS (*2) "mytn1"; TIMESPEC "TS_mytn1" = TO "mytn1" 5 ns;	define_scope_collection mytn1__0_0 {find -net clk1} define_scope_collection mytn1__0_fr {expand -hier -from $mytn1__0_0} define_scope_collection mytn1__0_ff {find -inst * -in $mytn1__0_fr -filter @view==dff* || @view==sdff* || @view==FD} define_scope_collection mytn1__0_ff_fr {expand -hier -from $mytn1__0_ff -level 1} define_scope_collection mytn1__0_ff_fr_net {find -net *2 -in $mytn1__0_ff_fr} define_scope_collection mytn1__0_ff_plus {expand -hier -to $mytn1__0_ff_fr_net -level 1} define_scope_collection mytn1 {find -inst * -in $mytn1__0_ff_plus -filter @view==dff* \|\| @view==sdff* \|\| @view==FD} define_path_delay -to {{$mytn1}} -max 5.000 See *Predefined Groups*, on page 1008 for the syntax for predefined groups.

TIMEGRP Constraint Translation Example

UCF Syntax	Converted SDC Constraint
TIMEGRP "group1" = FFS; TIMESPEC "TS_group1" = PERIOD "group1" 5 ns;	define_scope_collection group1 {c_union $FFS $FFS} define_path_delay -to {{$group1}} -max 5.000 See *Predefined Groups*, on page 1008 for the syntax for the predefined group $FFS.

Predefined Groups

The following table describes the SDC syntax for predefined groups.

LOC Constraint Translation Examples

Single Location (Net)

The following shows how a LOC constraint for a single location attached to a net is translated:

UCF Syntax	Converted SDC Constraint
NET "clock" LOC = "C2";	define_attribute {clock} xc_loc {C2}

Single Location (Instance)

The following shows how a LOC constraint for a single location attached to an instance is translated:

UCF Syntax	Converted SDC Constraint
INST "glbbuffer" LOC="GCLKBUF1";	define_attribute {glbbuffer} xc_loc {GCLKBUF1}

Multiple Locations (Net)

This is an example of the translation of LOC constraints for multiple locations attached to a net:

UCF Syntax	Converted SDC Constraint
NET "DVI_TX_BLUE_OUT[*]" LOC = H22,W2, W1,W7,H21,H19;	define_attribute {DVI_TX_BLUE_OUT[*]} xc_loc {H22,W2,W1,W7,H21,H19}

Multiple Locations (Instance)

This is an example of the translation of LOC constraints for multiple locations attached to an instance:

UCF Syntax	Converted SDC Constraint
INST "u?/*" LOC= CLB_R3C1, CLB_R3C2;	define_attribute {u?.*} xc_loc { CLB_R3C1, CLB_R3C2}

Copyright © 2013 Synopsys, Inc.

Range of Locations

This is an example of the translation of a LOC constraints for a range of locations attached to an instance:

UCF Syntax	Converted SDC Constraint
INST "u?/*" LOC= CLB_R3C1:CLB_R3C2;	define_attribute {u?.*} xc_loc { CLB_R3C1: CLB_R3C2}

MAXDELAY Constraint Translation Examples

These examples show how group-based MAXDELAY TPTHRU constraints are translated:

UCF Syntax	Converted SDC Constraint
NET "dout1" TPTHRU = "thru1"; TIMESPEC "TS_basic1" = FROM FFS THRU "thru1" TO PADS 4.5;	define_scope_collection thru1 {find -net dout1} define_path_delay -from {{$FFS}} -through {$thru1} -to {{$PADS}} -max 4.500
TIMESPEC "TS_group1" = FROM RAMS 5;	define_path_delay -from {{$RAMS}} -max 5.000

Copyright © 2013 Synopsys, Inc.

The following shows how TNM-based MAXDELAY TPTHRU constraints are translated:

UCF Syntax	Converted SDC Constraint
NET "clk" TNM_NET = "clk"; TIMESPEC "TS_clk" = PERIOD "clk" 2.000 ns; NET "din" TNM_NET = "start"; NET "data_int" TNM_NET = "end"; NET "int" TPTHRU = "thru1"; INST "data_int" TPTHRU = "thru2"; TIMESPEC "TS_basic2" = MAXDELAY FROM "start" THRU "thru1" THRU "thru2" TO "end" TS_clk*2;	define_clock -name {clk} -period 2.000 -clockgroup clkgroup_clk define_scope_collection start__0_0 {find -net din} define_scope_collection start__0_fr {expand -hier -from $start__0_0} define_scope_collection start {find -seq * -in $start__0_fr} define_scope_collection end__0_0 {find -net data_int} define_scope_collection end__0_fr {expand -hier -from $end__0_0} define_scope_collection end__0_fr {find -seq * -in $end__0_fr} define_scope_collection thru1 {find -net int} define_scope_collection thru2__0_0 {find -inst i:data_int} define_scope_collection thru2__0_fr {expand -hier -from $thru2__0_0 -level 1} define_scope_collection thru2 {find -net * -in $thru2__0_fr} define_path_delay -from {{$start}} -through {$thru1} -through {$thru2} -to {{$end}} -max 4.000
NET "din" TNM = "din_ff"; PIN "out.D" TPTHRU = "mypin"; TIMESPEC "TS_basic2" = MAXDELAY FROM "din_ff" THRU "mypin" 333333 KHz;	define_scope_collection din_ff__0_0 {find -net din} define_scope_collection din_ff__0_fr {expand -hier -from $din_ff__0_0} define_scope_collection din_ff__0_fr {find -seq * -in $din_ff__0_fr} define_scope_collection mypin {find -pin out.D} define_path_delay -from {$din_ff} -through {$mypin} -max 3.000 #WARNING# PIN <pin> TPTHRU is supported only for pins of instantiated components.

Copyright © 2013 Synopsys, Inc.

OFFSET Constraint Translation Examples

Global OFFSET

UCF Syntax	Converted SDC Constraint
OFFSET = OUT 40 ns AFTER "clk";	define_output_delay -default 10.00 -route 0.00 -ref {clk:r} This computation is based on a clock period of 50ns: output_delay=(period-offset).

Net OFFSET

UCF Syntax	Converted SDC Constraint
NET "sften" OFFSET = IN : 6ns : BEFORE : clk;	define_input_delay {sften} 4.00 -route 0.00 -ref {clk:r} This computation is based on a clock period of 10ns: input_delay=(period-offset).

Copyright © 2013 Synopsys, Inc.

Group OFFSET

PERIOD Constraint Translation Examples

PERIOD in TIMESPEC

UCF Syntax	Converted SDC Constraint
NET:"clock":TNM_NET = "clock"; TIMESPEC:"TS_clock" = PERIOD:"clock":50:ns:HIGH:40%: INPUT_JITTER:50:ps:PRIORITY:5;	define_clock -name {n:clock} -period 50.000 -clockgroup clkgroup_clock -rise 0.00 -fall 20.00 #WARNING# Keyword INPUT_JITTER is not supported. #WARNING# Keyword PRIORITY is not supported. #WARNING# Nearest Equivalent: TIMESPEC: "TS_clock" = PERIOD:"clock":50:ns:HIGH:40%; Synthesis ignores INPUT_JITTER and PRIORITY, which is why the nearest equivalents are used.
TIMEGRP "group1" = RAMS; TIMESPEC "TS_group1" = PERIOD "group1"5.0 ns HIGH 50.00%;	define_scope_collection RAMS {find -hier -inst * -filter @view==ram1 || @view==nram || @view==RAM* || @view==ram*} define_scope_collection group1 {c_union $RAMS $RAMS} define_path_delay -to {$group1} -max 5.000

PERIOD Attached to a Net

UCF Syntax	Converted SDC Constraint

NET "clock":PERIOD = 28 ns:
HIGH:60%INPUT_JITTER:10:ps;
	define_clock -name {n:clock} -period 28.000 -clockgroup
	clkgroup_clock -rise 0.000 -fall 16.800
	#WARNING# Keyword INPUT_JITTER is not supported.
	#WARNING# Nearest Equivalent: NET "clock":
	PERIOD = 28 ns:HIGH:60%;
	
	Synthesis ignores INPUT_JITTER, which is why the nearest equivalents are used.

Copyright © 2013 Synopsys, Inc.

Derived Clocks

UCF Syntax	Converted SDC Constraint
NET "clock":TNM_NET = "clock"; TIMESPEC:"TS_clock" = PERIOD "clock" 7.100 ns HIGH 50%; NET "count1/clk" TNM_NET = "count1_clk"; TIMESPEC "TS_count1_clk" = PERIOD:"count1_clk": "TS_clock":/:5:PHASE:+:10:ps: INPUT_JITTER:20:ps: PRIORITY:4;	define_clock -name {n:clock} -period 7.100 -clockgroup clkgroup_clock define_clock -name {n:count1.clk} -period 1.420 -clockgroup clkgroup_clock -rise 0.010 -fall 0.720 #WARNING# Keyword INPUT_JITTER is not supported #WARNING# Keyword PRIORITY is not supported. #WARNING# Nearest Equivalent: TIMESPEC "TS_count1_clk" = #PERIOD:"count1_clk":"TS_clock":/:5:PHASE:+:10:ps; Synthesis ignores INPUT_JITTER and PRIORITY, which is why the nearest equivalents are used.

PROP Constraint Translation Examples

General PROP Constraints

The following shows how a KEEP property constraint is translated:

UCF Syntax	Converted SDC Constraint
NET "keep1" KEEP;	define_attribute {keep1} KEEP {TRUE}.

The following shows an example of a UCF RAM property constraint and its equivalent `sdc` constraint after translation:

UCF Syntax	Converted SDC Constraint
INST "instRAM" WRITE_MODE_A = READ_FIRST;	define_attribute {i:instRAM} WRITE_MODE_A {READ_FIRST}

RAM PROP Constraint

UCF Syntax	Converted SDC Constraint
INST "instRAM" WRITE_MODE_A = READ_FIRST;	define_attribute {i:instRAM} WRITE_MODE_A {READ_FIRST}

Copyright © 2013 Synopsys, Inc.

IOB PROP Constraint

UCF Syntax	Converted SDC Constraint
NET "in" IOSTANDARD = AGP ;	define_io_standard {in} syn_pad_type {AGP}

DCM PROP Constraint

UCF Syntax	Converted SDC Constraint
INST "dcm1" CLK_FEEDBACK = 1X	define_attribute {dcm1} CLK_FEEDBACK {1X}

TIG Constraint Examples

Global TIG

UCF Syntax	Converted SDC Constraint
NET "reset_l" TIG;	define_false_path -through {n:reset_l}

Single_Tsid TIG

UCF Syntax	Converted SDC Constraint
NET "int" TIG=TS_clk; * UCF can contain multiple timespecs or a single timespec, but when used in conjunction with TIG(TIG=TS_identifiers), the utility ignores it.	#WARNING# NET "int" TIG=TS_clk; Not supported. #WARNING# Nearest equivalent - NET “int” TIG; #define_false_path -through {int} Since the false path has highest priority, the software cannot ignore the path for “TS_clk” and constrain it using MAX DELAY for any other TIMESPEC.

Multiple_Tsid TIG

UCF Syntax	Converted SDC Constraint
NET "reset_l" TIG= TS_2, TS_3;	#WARNING# NET "reset_l" TIG= TS_2, TS_3; Not Supported #WARNING# Nearest equivalent - NET "reset_l" TIG; #define_false_path -through {reset_l} Since the false path has highest priority, the software cannot ignore the path for TS_2 and TS_3 and constrain it using MAX DELAY for any other TIMESPEC.

Copyright © 2013 Synopsys, Inc.

Unsupported UCF Constraints

The following types of OFFSET constraints are currently not translated:

- OFFSET Using Keyword VALID, on page 1016
- OFFSET Using HIGH/LOW Clock Edge, on page 1016

OFFSET Using Keyword VALID

The following warnings are generated when the OFFSET constraint: contains the VALID keyword.

OFFSET with VALID: UCF Syntax	Converted SDC Constraint

OFFSET = IN : 11 ns : VALID 15 ns : BEFORE : clk;;	define_input_delay -default 9.00 -route 0.00 -ref {clk:r}
	
	#WARNING# Keyword VALID is not supported.
	#WARNING# Nearest equivalent#
	OFFSET = IN : 11 ns :BEFORE : clk;

OFFSET Using HIGH/LOW Clock Edge

The following types of warnings are generated when the OFFSET constraint contains the HIGH/LOW keyword for the clock edge:

OFFSET with Logic Level: UCF Syntax	Converted SDC Constraint
INST "y1.PAD" TNM = "y1_output"; TIMEGRP "y1_output" OFFSET = OUT 16 ns AFTER "clk" HIGH ; INST "y2.PAD" TNM = "y2_output"; TIMEGRP "y2_output" OFFSET = OUT 16 ns AFTER "clk" LOW ;	#WARNING# Keyword HIGH/LOW defining “CLK edge” is not supported.

Copyright © 2013 Synopsys, Inc.

Xilinx ise2syn Utility

The ise2syn utility converts a Xilinx project to a Synplify Pro or Synplify Premier project. The following describes the command line syntax for running the ise2syn project conversion utility. For information about using the utility to convert Xilinx projects, see *Converting Xilinx Projects with ise2syn*, on page 1051 in the *User Guide*.

ise2syn [**-ise** *project*] [**-synproject** *project*] [**-force_wbox** *cores*]

-ise *project*	Specifies the Xilinx project (`*.ise`, `*.xise`, or `*.xmp`) to import.
-synproject *project*	Specifies a name for the synthesis project to be created.
-force_wbox *cores*	Forces listed cores to be treated as white boxes. The tool uses the `ngc` files for these cores instead of the HDL files. Separate multiple core names with commas. See *Force White Box Cores Option (EDK)*, on page 227 for details.

This table describes the input and output files for the ise2syn utility:

Input files	The ise2syn utility can use any of the following as input. It extracts the information needed to construct a synthesis project. • `ise` file • `xise` file • `xmp` EDK project file
Output files	The utility generates the following output files: • Synplify Pro/Premier synthesis project file: `prj` • Synthesis constraint file: _conv.sdc, with converted `ucf`, `ncf`, or `xcf` constraints • Xilinx User Constraint file: _unapplied.ucf, that contains unconverted constraints

Copyright © 2013 Synopsys, Inc.

Xilinx Vivado Tcl Commands

Xilinx

The synthesis tools include some Tcl commands to facilitate the flow between the synthesis and Vivado place-and-route tools. These commands automate the importing of Xilinx IP and the conversion of Xilinx constraints, and are described below:

- import_vivado_ip Command, on page 1018
- xdc2fdc Command, on page 1020

import_vivado_ip Command

The import_vivado_ip command imports a Xilinx Vivado IP into a synthesis project. The Xilinx IP to be imported must be generated from the Vivado IP catalog. You can import the IP as a top-level design or a subproject. See *Importing Vivado IP*, on page 1043 in the *User Guide* for a detailed procedure.

The command parses a specified IP source file and builds a synthesis project. If the IP includes XDC constraints, it also translates them to synthesis FDC constraints and adds the fdc file to the synthesis project. The original Xilinx XDC files are tagged as PAR-only in the synthesis project.

import_vivado_ip Syntax

import_vivado_ip [-flat | -hier]
ip_dir **|-xml_file** *file*.**xml | -xci_file** *file*.**xci | -xco_file** *file*.**xco**

-flat	Adds all IP files to the current top-level project. This is an optional argument.
-hier	Creates a subproject for the IP under the currently loaded synthesis project, and adds all IP files to it. This is an optional argument, and has no effect if it is specified for a design where the IP is the top level.
-ip_dir	Specifies the IP directory where the xml file for the IP is located. This file is used to build a synthesis project for the IP. You must specify the directory or the xml, xci, or xco files as input.

-xml_file *file*	Specifies the xml file for the IP. The tool uses this file to generate a synthesis project for the IP. You must specify the IP directory, the xci, or xco files as input.
-xci_file *file*	Specifies the xci file for the IP. The tool uses this file to generate a synthesis project for the IP. You must specify the IP directory, the xml, or xco files as input.
-xco_file *file*	Specifies the xco file for the IP. The tool uses this file to generate a synthesis project for the IP. You must specify the IP directory, the xml, or xci files as input.
Output directory	Specifies the output directory where the import files are saved. You do not need to specify any option. The following default values are used: • If the top-level project is loaded, all the import files generated are saved in the top-level project directory. • If the top-level project is not loaded, a new directory is created in the IP directory with the name synplify. All import files generated are saved in this directory.

Example: Importing Vivado IP as Top Level

In this example, no synthesis project is open, so the IP is imported as a top-level design. The example shows the command entered and the report generated of the process:

% import_vivado_ip C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_v1_8_a_0

```
No project open.

Processing IP XML file C:/Projects/project_mig/project_mig.srcs
/sources_1/ip/mig_7series_v1_8_a_0/mig_7series_v1_8_a_0.xml

Warning: No project open in the workspace. All IP files will be
added to a new top level project.

Info: If you want to import this IP as sub-project, open an
existing/new project as top level project and then import IP.

Creating new top level project
C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_
v1_8_a_0/synplify/mig_7series_v1_8_a_0.prj
```


Copyright © 2013 Synopsys, Inc.

```
Translating XDC to FDC. See translation log file for details
C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_
v1_8_a_0/synplify/constraints/mig_7series_v1_8_a_0.log

IP import log file:
C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_
v1_8_a_0/synplify/mig_7series_v1_8_a_0_import_ip.log

IP import completed successfully
```

Example: Importing Vivado IP as Part of a Larger Design

With this example, the parent synthesis project is open. The example shows the command, and the results of running it.

% import_vivado_ip C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_v1_8_a_0

```
Processing IP XML file
C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_
v1_8_a_0/mig_7series_v1_8_a_0.xml

Creating IP sub-project
C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_
v1_8_a_0/synplify/ipcores/mig_7series_v1_8_a_0/mig_7series_v1_8_a_
0.prj

Translating XDC to FDC. See translation log file for details
C:/mig/synplify/ipcores/mig_7series_v1_8_a_0/constraints/mig_7seri
es_v1_8_a_0.log

IP import log file:
C:/Projects/project_mig/project_mig.srcs/sources_1/ip/mig_7series_
v1_8_a_0/synplify/mig_7series_v1_8_a_0_import_ip.log

IP import completed successfully
```

xdc2fdc Command

The xdc2fdc command translates Xilinx timing constraints file (XDC) to Synopsys FPGA timing constraints file (FDC). The command parses the Xilinx XDC file and tries to convert all the constraints to Synplify FDC constraints.

Copyright © 2013 Synopsys, Inc.

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.

For details about this command and its use, see these topics:

- xdc2fdc Syntax, next
- Example: Translation of All xdc Files in a Project, on page 1022
- Example: Translation of a Single xdc File, on page 1023
- Translating XDC Constraints to FDC, on page 190 in the *User Guide*

xdc2fdc Syntax

You can run this command either on the current project or on a specified xdc file, using the following syntax:

xdc2fdc -project
xdc2fdc -xdc_file *file*.**xdc -fdc_file** *file*.**fdc** [**-hier** *instanceName*]

-project	Translates all xdc files in the current project. This option requires the project to be open.
-xdc_file *file*	Specifies the xdc file to be translated. The file does not have to be in the current project.
-fdc_file *file*	Specifies the fdc file to which the translated constraints from the specified xdc file are written.
-hier *instanceName*	Adds extra hierarchy levels for every constrained object. This is required for IP used in a lower-level module of a large design. Usage examples: • `-hier "*."` – Adds hierarchy for all instances of the IP • `-hier "i1."` – Adds hierarchy with "i1" instances of the IP • `-hier "*i1."` – Adds two levels of hierarchy

If you specify the command without any arguments, it displays help information, as shown below:

% xdc2fdc

Copyright © 2013 Synopsys, Inc.

```
Usage:
   Mode 1: Translate all XDC files in current project.
            xdc2fdc -project

   Mode 2: Translate supplied file. Current project is not
   affected.
            xdc2fdc -xdc_file <file>.xdc -fdc_file <file>.fdc
```

Example: Translation of All xdc Files in a Project

The following example shows the xdc2fdc command run with -project switch to convert all the xdc files in the currently open synthesis project. The translated Synplify FDC files are then added to the active synthesis project, and the original Xilinx xdc files are marked as place-and-route only in the synthesis project. The tool also generates detailed messages, as shown below.

% xdc2fdc -project

```
Opening top level log file
C:/Projects/synplify/mig_7series_v1_8_a_0_xdc_translation.log

Translating XDC files (Xilinx constraints files). Found following
XDC files:
   ./constraints/mig_7series_v1_8_a_0.1.xdc
   ./constraints/mig_7series_v1_8_a_0.2.xdc

Running XDC to FDC translation
Input XDC constraints file
./constraints/mig_7series_v1_8_a_0.1.xdc
Output FDC constraints file
C:/Projects/synplify/FDC_constraints/rev_1/mig_7series_v1_8_a_0.1_
xdc_translated.fdc

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.

Line 310: set_multicycle_path -from [get_cells -hier \
{*.mc0.mc_read_idle_r_reg}] -to [get_cells -hier \
{*.input_\[?\]\.iserdes_dq_\.iserdesdq}] -setup 6

Constraint translation log file
C:/Projects/synplify/FDC_constraints/rev_1/mig_7series_v1_8_a_0.1_
xdc_translated.log
```

```
Constraints translated 184, not translated 0
Constraint translation completed successfully
Adding FDC to project, moving XDC under par-only


Running XDC to FDC translation
Input XDC constraints file
./constraints/mig_7series_v1_8_a_0.2.xdc
Output FDC constraints file
C:/Projects/synplify/FDC_constraints/rev_1/mig_7series_v1_8_a_0.2_
xdc_translated.fdc

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.

Line 310: set_multicycle_path -from [get_cells -hier \
{*.mc0.mc_read_idle_r_reg}] -to [get_cells -hier \
{*.input_\[?\]\.iserdes_dq_\.iserdesdq}] -hold 5

Constraint translation log file
C:/Projects/synplify/FDC_constraints/rev_1/mig_7series_v1_8_a_0.2_
xdc_translated.log

Constraints translated 184, not translated 0
Constraint translation completed successfully
Adding FDC to project, moving XDC under par-only

Please save your project manually
```

Example: Translation of a Single xdc File

The following example translates the constraints in a specific xdc file to fdc. The example also shows a typical report that results from running the command:

% xdc2fdc -xdc_file C:/Projects/project_mig/constraints/mig.xdc -fdc_file C:/Projects/project_mig/constraints/mig.fdc

```
Translating XDC files (Xilinx constraints files). Found following
XDC files:
C:/Projects/project_mig/constraints/mig.xdc
```


Copyright © 2013 Synopsys, Inc.

```
Running XDC to FDC translation
Input XDC constraints file
C:/Projects/project_mig/constraints/mig.xdc
Output FDC constraints file
C:/Projects/project_mig/constraints/mig.fdc

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.

Line 310: set_multicycle_path -from [get_cells -hier \
{*.mc0.mc_read_idle_r_reg}] -to [get_cells -hier \
{*.input_\[?\]\.iserdes_dq_\.iserdesdq}] -setup 6

Constraint translation log file
C:/Projects/project_mig/constraints/mig.log

Constraints translated 184, not translated 0

Constraint translation completed successfully

Please save your project manually
```

Synplicity Archive Utility

The archive utility provides a way to archive, extract, or copy your design projects. An archive file is in Synplicity proprietary format and is saved to a file name using the `sar` extension. You can also use this utility to submit your design along with a request for technical support.

The archive utility is available through the Project menu in the GUI or through the project Tcl command. See the following for details:

For information about...	See...
Archiving, un-archiving, or copying projects	*Archiving Files and Projects, on page 287* in the *User Guide*
Archiving a project for Synplicity technical support	*Tech-Support Menu, on page 353*
Syntax for the associated Tcl commands	project -archive, project -unarchive, and project -copy Tcl commands

Copyright © 2013 Synopsys, Inc.

SYNCore FIFO Compiler

The SYNCore synchronous FIFO compiler offers an IP wizard that generates Verilog code for your FIFO implementation. This section describes the following:

- Synchronous FIFOs, on page 1026
- FIFO Read and Write Operations, on page 1027
- FIFO Ports, on page 1029
- FIFO Parameters, on page 1031
- FIFO Status Flags, on page 1033
- FIFO Programmable Flags, on page 1036

For further information, refer to the following:

- Specifying FIFOs with SYNCore, on page 934 of the *User Guide.*, for information about using the wizard to generate FIFOs
- Launch SYNCore Command, on page 245 and SYNCore FIFO Wizard, on page 247 for descriptions of the interface

Synchronous FIFOs

A FIFO is a First-In-First-Out memory queue. Different control logic manages the read and write operations. A FIFO also has various handshake signals for interfacing with external user modules.

The SYNCore FIFO compiler generates synchronous FIFOs with symmetric ports and one clock controlling both the read and write operations. The FIFO is symmetric because the read and write ports have the same width.

When the Write_enable signal is active and the FIFO has empty locations, data is written into FIFO memory on the rising edge of the clock. A Full status flag indicates that the FIFO is full and that no more write operations can be performed. See *FIFO Write Operation*, on page 1027 for details.

When the FIFO has valid data and Read_enable is active, data is read from the FIFO memory and presented at the outputs. The FIFO Empty status flag indicates that the FIFO is empty and that no more read operations can be performed. See *FIFO Read Operation*, on page 1028 for details.

Copyright © 2013 Synopsys, Inc.

The FIFO is not corrupted by an invalid request: for example, if a read request is made while the FIFO is empty or a write request is received when the FIFO is full. Invalid requests do not corrupt the data, but they cause the corresponding read or write request to be ignored and the Overflow or Underflow flags to be asserted. You can monitor these status flags for invalid requests. These and other flags are described in *FIFO Status Flags*, on page 1033 and *FIFO Programmable Flags*, on page 1036.

At any point in time, Data count reflects the available data inside the FIFO. In addition, you can use the Programmable Full and Programmable Empty status flags for user-defined thresholds.

FIFO Read and Write Operations

This section describes FIFO behavior with read and write operations.

FIFO Write Operation

When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus (Din) and write acknowledge (Write_ack) is asserted. If the FIFO is continuously written without being read, it will fill with data. The status outputs are asserted when the number of entries in the FIFO is greater than or equal to the corresponding threshold, and should be monitored to avoid overflowing the FIFO.

When the FIFO is full, any attempted write operation fails and the overflow flag is asserted.

The following figure illustrates the write operation. Write acknowledge (Write_ack) is asserted on the next rising clock edge after a valid write operation. When Full is asserted, there can be no more legal write operations. This example shows that asserting Write_enable when Full is high causes the assertion of Overflow.

Copyright © 2013 Synopsys, Inc.

FIFO Read Operation

When read enable is asserted and the FIFO is not empty, the next data word in the FIFO is driven on the output bus (Dout) and a read valid is asserted. If the FIFO is continuously read without being written, the FIFO will empty. The status outputs are asserted when the number of entries in the FIFO are less than or equal to the corresponding threshold, and should be monitored to avoid underflowing the FIFO. When the FIFO is empty, all read operations fail and the underflow flag is asserted.

If read and write operation occur simultaneously during the empty state, the write operation will be valid and empty, and is de-asserted at the next rising clock edge. There cannot be a legal read operation from an empty FIFO, so the underflow flag is asserted.

The following figure illustrates a typical read operation. If the FIFO is not empty, Read_ack is asserted at the rising clock edge after Read_enable is asserted and the data on Dout is valid. When Empty is asserted, no more read operations can be performed. In this case, initiating a read causes the assertion of Underflow on the next rising clock edge, as shown in this figure.

Copyright © 2013 Synopsys, Inc.

FIFO Ports

The following figure shows the FIFO ports.

Port Name	Description
Almost_empty	Almost empty flag output (active high). Asserted when the FIFO is almost empty and only one more read can be performed. Can be active high or active low.
Almost_full	Almost full flag output (active high). Asserted when only one more write can be performed into the FIFO. Can be active high or active low.
AReset	Asynchronous reset input. Resets all internal counters and FIFO flag outputs.
Clock	Clock input for write and read. Data is written/read on the rising edge.
Data_cnt	Data word count output. Indicates the number of words in the FIFO in the read clock domain.
Din [width:0]	Data input word to the FIFO.

Copyright © 2013 Synopsys, Inc.

Port Name	Description
Dout [width:0]	Data output word from the FIFO.
Empty	FIFO empty output (active high). Asserted when the FIFO is empty and no additional reads can be performed. Can be active high or active low.
Full	FIFO full output (active high). Asserted when the FIFO is full and no additional writes can be performed. Can be active high or active low.
Overflow	FIFO overflow output flag (active high). Asserted when the FIFO is full and the previous write was rejected. Can be active high or active low.
Prog_empty	Programmable empty output flag (active high). Asserted when the words in the FIFO exceed or equal the programmable empty assert threshold. De-asserted when the number of words is more than the programmable full negate threshold. Can be active high or active low.
Prog_empty_ thresh	Programmable FIFO empty threshold input. User-programmable threshold value for the assertion of the Prog_empty flag. Set during reset.
Prog_empty_ thresh_assert	Programmable FIFO empty threshold assert input. User-programmable threshold value for the assertion of the Prog_empty flag. Set during reset.
Prog_empty_ thresh_negate	Programmable FIFO empty threshold negate input. User programmable threshold value for the de-assertion of the Prog_full flag. Set during reset.
Prog_full	Programmable full output flag (active high). Asserted when the words in the FIFO exceed or equal the programmable full assert threshold. De-asserted when the number of words is less than the programmable full negate threshold. Can be active high or active low.
Prog_full_thresh	Programmable FIFO full threshold input. User-programmable threshold value for the assertion of the Prog_full flag. Set during reset.
Prog_full_thresh_ assert	Programmable FIFO full threshold assert input. User-programmable threshold value for the assertion of the Prog_full flag. Set during reset.

Copyright © 2013 Synopsys, Inc.

Port Name	Description
Prog_full_thresh_negate	Programmable FIFO full threshold negate input. User-programmable threshold value for the de-assertion of the Prog_full flag. Set during reset.
Read_ack	Read acknowledge output (active high). Asserted when valid data is read from the FIFO. Can be active high or active low.
Read_enable	Read enable output (active high). If the FIFO is not empty, data is read from the FIFO on the next rising edge of the read clock.
Underflow	FIFO underflow output flag (active high). Asserted when the FIFO is empty and the previous read was rejected.
Write_ack	Write Acknowledge output (active high). Asserted when there is a valid write into the FIFO. Can be active high or active low.
Write_enable	Write enable input (active high). If the FIFO is not full, data is written into the FIFO on the next rising edge.

FIFO Parameters

Parameter	Description
AEMPTY_FLAG_SENSE	FIFO almost empty flag sense 0 Active Low 1 Active High
AFULL_FLAG_SENSE	FIFO almost full flag sense 0 Active Low 1 Active High
DEPTH	FIFO depth
EMPTY_FLAG_SENSE	FIFO empty flag sense 0 Active Low 1 Active High
FULL_FLAG_SENSE	FIFO full flag sense 0 Active LowOVERFLOW_ 1 Active High

Copyright © 2013 Synopsys, Inc.

Parameter	Description
OVERFLOW_FLAG_SENSE	FIFO overflow flag sense 0 Active Low 1 Active High
PEMPTY_FLAG_SENSE	FIFO programmable empty flag sense 0 Active Low 1 Active High
PFULL_FLAG_SENSE	FIFO programmable full flag sense 0 Active Low 1 Active High
PGM_EMPTY_ATHRESH	Programmable empty assert threshold for PGM_EMPTY_TYPE=2
PGM_EMPTY_NTHRESH	Programmable empty negate threshold for PGM_EMPTY_TYPE=2
PGM_EMPTY_THRESH	Programmable empty threshold for PGM_EMPTY_TYPE=1
PGM_EMPTY_TYPE	Programmable empty type. See *Programmable Empty*, on page 1040 for details. 1 Programmable empty with single threshold constant. 2 Programmable empty with multiple threshold constant 3 Programmable empty with single threshold input 4 Programmable empty with multiple threshold input
PGM_FULL_ATHRESH	Programmable full assert threshold for PGM_FULL_TYPE=2
PGM_FULL_NTHRESH	Programmable full negate threshold for PGM_FULL_TYPE=2
PGM_FULL_THRESH	Programmable full threshold for PGM_FULL_TYPE=1
PGM_FULL_TYPE	Programmable full type. See *Programmable Full*, on page 1037 for details. 1 Programmable full with single threshold constant 2 Programmable full with multiple threshold constant 3 Programmable full with single threshold input 4 Programmable full with multiple threshold input

Copyright © 2013 Synopsys, Inc.

Parameter	Description
RACK_FLAG_SENSE	FIFO read acknowledge flag sense 0 Active Low 1 Active High
UNDERFLOW_FLAG_SENSE	FIFO underflow flag sense 0 Active Low 1 Active High
WACK_FLAG_SENSE	FIFO write acknowledge flag sense 0 Active Low 1 Active High
WIDTH	FIFO data input and data output width

FIFO Status Flags

You can set the following status flags for FIFO read and write operations.

- Full/Almost Full Flags, on page 1033
- Empty/Almost Empty Flags, on page 1034
- Handshaking Flags, on page 1035
- Programmable full and empty flags, which are described in *Programmable Full*, on page 1037 and *Programmable Empty*, on page 1040.

Full/Almost Full Flags

These flags indicate the status of the FIFO memory queue for write operations:

Full	Indicates that the FIFO memory queue is full and no more writes can be performed until data is read. Full is synchronous with the clock (Clock). If a write is initiated when Full is asserted, the write does not succeed and the overflow flag is asserted.
Almost_full	The almost full flag (Almost_full) indicates that there is one location left and the FIFO will be full after one more write operation. Almost full is synchronous to Clock. This flag is guaranteed to be asserted when the FIFO has one remaining location for a write operation.

Copyright © 2013 Synopsys, Inc.

The following figure displays the behavior of these flags. In this example, asserting Wriite_enable when Almost_full is high causes the assertion of Full on the next rising clock edge.

Empty/Almost Empty Flags

These flags indicate the status of the FIFO memory queue for read operations:

Empty	Indicates that the memory queue for the FIFO is empty and no more reads can be performed until data is written. The output is active high and is synchronous to the clock. If a read is initiated when the empty flag is true, the underflow flag is asserted.
Almost_ empty	Indicates that the FIFO will be empty after one more read operation. Almost_empty is active high and is synchronous to the clock. The flag is guaranteed to be asserted when the FIFO has one remaining location for a read operation.

The following figure illustrates the behavior of the FIFO with one word remaining.

Handshaking Flags

You can specify optional Read_ack, Write_ack, Overflow, and Underflow handshaking flags for the FIFO.

Copyright © 2013 Synopsys, Inc.

Read_ack	Asserted at the completion of each successful read operation. It indicates that the data on the Dout bus is valid. It is an optional port that is synchronous with Clock and can be configured as active high or active low. Read_ack is deasserted when the FIFO is underflowing, which indicates that the data on the Dout bus is invalid. Read_ack is asserted at the next rising clock edge after read enable. Read_enable is asserted when the FIFO is not empty.
Write_ack	Asserted at the completion of each successful write operation. It indicates that the data on the Din port has been stored in the FIFO. It is synchronous with the clock, and can be configured as active high or active low. Write_ack is deasserted for a write to a full FIFO, as illustrated in the figure. Write_ack is deasserted one clock cycle after Full is asserted to indicate that the last write operation was valid and no other write operations can be performed.
Overflow	Indicates that a write operation was unsuccessful because the FIFO was full. In the figure, Full is asserted to indicate that no more writes can be performed. Because the write enable is still asserted and the FIFO is full, the next cycle causes Overflow to be asserted. Note that Write_ack is not asserted when FIFO is overflowing. When the write enable is deasserted, Overflow deasserts on the next clock cycle.
Underflow	Indicates that a read operation was unsuccessful, because the read was attempted on an empty FIFO. In the figure, Empty is asserted to indicate that no more reads can be performed. As the read enable is still asserted and the FIFO is empty, the next cycle causes Underflow to be asserted. Note that Read_ack is not asserted when FIFO is underflowing. When the read enable is deasserted, the Underflow flag deasserts on the next clock cycle.

FIFO Programmable Flags

The FIFO supports completely programmable full and empty flags to indicate when the FIFO reaches a predetermined user-defined fill level. See the following:

Prog_full	Indicates that the FIFO has reached a user-defined full threshold. See *Programmable Full*, on page 1037 for more information.
Prog_empty	Indicates that the FIFO has reached a user-defined empty threshold. See *Programmable Empty*, on page 1040 for more information.

Copyright © 2013 Synopsys, Inc.

Both flags support various implementation options. You can do the following:

- Set a constant value
- Set dedicated input ports so that the thresholds can change dynamically in the circuit
- Use hysteresis, so that each flag has different assert and negative values

Programmable Full

The Prog_full flag (programmable full) is asserted when the number of entries in the FIFO is greater than or equal to a user-defined assert threshold. If the number of words in the FIFO is less than the negate threshold, the flag is de-asserted. The following is the valid range of threshold values:

Assert threshold value	Depth / 2 to Max of Depth For multiple threshold types, the assert value should always be larger than the negate value in multiple threshold types.
Negate threshold value	Depth / 2 to Max of Depth

Prog_full has four threshold types:

- Programmable Full with Single Threshold Constant, on page 1037
- Programmable Full with Multiple Threshold Constants, on page 1038
- Programmable Full with Single Threshold Input, on page 1038
- Programmable Full with Multiple Threshold Inputs, on page 1039

Programmable Full with Single Threshold Constant

PGM_FULL_TYPE = 1

This option lets you set a single constant value for the threshold. It requires significantly fewer resources when the FIFO is generated. This figure illustrates the behavior of Prog_full when configured as a single threshold constant with a value of 6.

Copyright © 2013 Synopsys, Inc.

Programmable Full with Multiple Threshold Constants

PGM_FULL_TYPE = 2

The programmable full flag is asserted when the number of words in the FIFO is greater than or equal to the full threshold assert value. If the number of FIFO words drops to less than the full threshold negate value, the programmable full flag is de-asserted. Note that the negate value must be set to a value less than the assert value. The following figure illustrates the behavior of Prog_full configured as multiple threshold constants with an assert value of 6 and a negate value of 4.

Programmable Full with Single Threshold Input

PGM_FULL_TYPE = 3

This option lets you specify the threshold value through an input port (Prog_full_thresh) during the reset state, instead of using constants. The following figure illustrates the behavior of Prog_full configured as a single threshold input with a value of 6.

Copyright © 2013 Synopsys, Inc.

Programmable Full with Multiple Threshold Inputs

PGM_FULL_TYPE = 4

This option lets you specify the assert and negate threshold values dynamically during the reset stage using the Prog_full_thresh_assert and Prog_full_thresh_negate input ports. You must set the negate value to a value less than the assert value.

The programmable full flag is asserted when the number of words in the FIFO is greater than or equal to the Prog_full_thresh_assert value. If the number of FIFO words goes below Prog_full_thresh_negate value, the programmable full flag is deasserted. The following figure illustrates the behavior of Prog_full configured as multiple threshold inputs with an assert value of 6 and a negate value of 4.

Copyright © 2013 Synopsys, Inc.

Programmable Empty

The programmable empty flag (Prog_empty) is asserted when the number of entries in the FIFO is less than or equal to a user-defined assert threshold. If the number of words in the FIFO is greater than the negate threshold, the flag is deasserted. The following is the valid range of threshold values:

Assert threshold value	1 to Max of Depth / 2 For multiple threshold types, the assert value should always be lower than the negate value in multiple threshold types.
Negate threshold value	1 to Max of Depth / 2

There are four threshold types you can specify:

- Programmable Empty with Single Threshold Constant, on page 1040
- Programmable Empty with Multiple Threshold Constants, on page 1041
- Programmable Empty with Single Threshold Input, on page 1041
- Programmable Empty with Multiple Threshold Inputs, on page 1042

Programmable Empty with Single Threshold Constant

PGM_EMPTY_TYPE = 1

This option lets you specify an empty threshold value with a single constant. This approach requires significantly fewer resources when the FIFO is generated. The following figure illustrates the behavior of Prog_empty configured as a single threshold constant with a value of 3.

Programmable Empty with Multiple Threshold Constants

PGM_EMPTY_TYPE = 2

This option lets you specify constants for the empty threshold assert value and empty threshold negate value. The programmable empty flag asserts and deasserts in the range set by the assert and negate values. The assert value must be set to a value less than the negate value. When the number of words in the FIFO is less than or equal to the empty threshold assert value, the Prog_empty flag is asserted. When the number of words in FIFO is greater than the empty threshold negate value, Prog_empty is deasserted.

The following figure illustrates the behavior of Prog_empty when configured as multiple threshold constants with an assert value of 3 and a negate value of 5.

Programmable Empty with Single Threshold Input

PGM_EMPTY_TYPE = 3

This option lets you specify the threshold value dynamically during the reset state with the Prog_empty_thresh input port, instead of with a constant. The Prog_empty flag asserts when the number of FIFO words is equal to or less than the Prog_empty_thresh value and deasserts when the number of FIFO words is more than the Prog_empty_thresh value. The following figure illustrates the behavior of Prog_empty when configured as a single threshold input with a value of 3.

Copyright © 2013 Synopsys, Inc.

Programmable Empty with Multiple Threshold Inputs

PGM_EMPTY_TYPE = 4

This option lets you specify the assert and negate threshold values dynamically during the reset stage using the Prog_empty_thresh_assert and Prog_empty_thresh_negate input ports instead of constants. The programmable empty flag asserts and deasserts according to the range set by the assert and negate values. The assert value must be set to a value less than the negate value.

When the number of FIFO words is less than or equal to the empty threshold assert value, Prog_empty is asserted. If the number of FIFO words is greater than the empty threshold negate value, the flag is deasserted. The following figure illustrates the behavior of Prog_empty configured as multiple threshold inputs, with an assert value of 3 and a negate value of 5.

Copyright © 2013 Synopsys, Inc.

SYNCore RAM Compiler

The SYNCore RAM Compiler generates Verilog code for your RAM implementation. This section describes the following:

- Single-Port Memories, on page 1043
- Dual-Port Memories, on page 1045
- Read/Write Timing Sequences, on page 1050

For further information, refer to the following:

- Specifying RAMs with SYNCore, on page 939 of the *User Guide*, for information about using the wizard to generate FIFOs
- Launch SYNCore Command, on page 245 and SYNCore FIFO Wizard, on page 247 for descriptions of the interface

Single-Port Memories

For single-port RAM, it is only necessary to configure Port A. The following diagrams show the read-write timing for single-port memories. See *Specifying RAMs with SYNCore*, on page 939 in the *User Guide* for a procedure.

Copyright © 2013 Synopsys, Inc.

Single-Port Read

Single-Port Write

Dual-Port Memories

SYNCore dual-port memory includes the following common configurations:

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

The following diagrams show the read-write timing for dual-port memories. See *Specifying RAMs with SYNCore*, on page 939 in the *User Guide* for a procedure to specify a dual-port RAM with SYNCore.

Copyright © 2013 Synopsys, Inc.

Dual-Port Single Read

Dual-Port Single Write

Copyright © 2013 Synopsys, Inc.

Dual-Port Read

Dual-Port Write

Copyright © 2013 Synopsys, Inc.

Read/Write Timing Sequences

The waveforms in this section describe the behavior of the RAM when both read and write are enabled and the address is the same operation. The waveforms show the behavior when each of the read-write sequences is enabled. The waveforms are merged with the simple waveforms shown in the previous sections. See the following:

- Read Before Write, on page 1050
- Write Before Read, on page 1051
- No Read on Write, on page 1052

Read Before Write

Copyright © 2013 Synopsys, Inc.

Write Before Read

Copyright © 2013 Synopsys, Inc.

No Read on Write

SYNCore Byte-Enable RAM Compiler

The SYNCore byte-enable RAM compiler generates SystemVerilog code describing byte-enabled RAMs. The data width of each byte is calculated by dividing the total data width by the write enable width. The byte-enable RAM compiler supports both single- and dual-port configurations.

This section describes the following:

- Functional Overview, on page
- Write Operation, on page
- Read Operation, on page
- Parameter List, on page

For further information, refer to the following:

- *Specifying Byte-Enable RAMs with SYNCore*, on page 946 of the user guide for information on using the wizard to generate single- or dual-port RAM configurations.
- *SYNCore Byte-Enable RAM Wizard*, on page 261 for descriptions of the interface.

Functional Overview

The SYNCore byte-enable RAM component supports bit/byte-enable RAM implementations using blockRAM and distributed memory. For each configuration, design optimizations are made for optimum use of core resources. The timing diagram that follow illustrate the supported signals for byte-enable RAM configurations.

Byte-enable RAM can be configured in both single- and dual-port configurations. In the dual-port configuration, each port is controlled by different clock, enable, and control signals. User configuration controls include selecting the enable level, reset type, and register type for the read data outputs and address inputs.

Reset applies only to the output read data registers; default value of read data on reset can be changed by user while generating core. Reset option is inactive when output read data is not registered.

Copyright © 2013 Synopsys, Inc.

Read/Write Timing Sequences

The waveforms in this section describe the behavior of the byte-enable RAM for both read and write operations.

Read Operation

On each active edge of the clock when there is a change in address, data is valid on the same clock or next clock (depending on latency parameter values for read address and read data ports). Active reset ignores any change in input address, and data and output data are initialized to user-defined values set by parameters RST_RDATA_A and RST_RDATA_B for port A and port B, respectively.

The following waveform shows the read sequence of the byte-enable RAM component with read data registered in single-port mode.

As shown in the above waveform, output read data changes on the same clock following the input address changed. When the address changes from 'h00 to 'h01, read data changes to 50 on the same clock, and data will be valid on the next clock edge.

The following waveform shows the read sequence with both the read data and address registered in single-port mode.

As shown in the above waveform, output read data changes on the next clock edge after the input address changes. When the address changes from 'h00 to 'h01, read data changes to 50 on the next clock, and data is valid on the next clock edge.

Note: The read sequence for dual-port mode is the same as single port; read/write conflicts occurring due to accessing the same location from both ports are the user's responsibility.

Write Operation

The following waveform shows a write sequence with read-after write in single-port mode.

Copyright © 2013 Synopsys, Inc.

On each active edge of the clock when there is a change in address with an active enable, data is written into memory on the same clock. When enable is not active, any change in address or data is ignored. Active reset ignores any change in input address and data.

The width of the write enable is controlled by the WE_WIDTH parameter. Input data is symmetrically divided and controlled by each write enable. For example, with a data width of 32 and a write enable width of 4, each bit of the write enable controls 8 bits of data (32/4=8). The byte-enable RAM compiler will error for wrong combination data width and write enable values.

The above waveform shows a write sequence with all possible values for write enable followed by a read:

- Value for parameter WE_WIDTH is 2 and DATA_WIDTH is 8 so each write enable controls 4 bits of input data.
- WenA value changes from 1 to 2, 2 to 0, and 0 to 3 which toggles all possible combinations of write enable.

The first sequence of address, write enable changes to perform a write sequence and the data patterns written to memory are 00, aa, ff. The read data pattern reflects the current content of memory before the write.

The second address sequence is a read (WenA is always zero). As shown in the read pattern, only the respective bits of data are written according to the write enable value.

Copyright © 2013 Synopsys, Inc.

Note: The write sequence for dual-port mode is the same as single port; conflicts occurring due to writing the same location from both ports are the user's responsibility.

Parameter List

The following table lists the file entries corresponding to the byte-enable RAM wizard parameters.

Name	Description	Default Value	Range
ADDR_WIDTH	Bit/byte enable RAM address width	2	multiples of 2
DATA_WIDTH	Data width for input and output data, common to both Port A and Port B	8	2 to 256
WE_WIDTH	Write enable width, common to both Port A and Port B	2	
CONFIG_PORT	Selects single/dual port configuration	1 (single port)	0 = dual-port 1 = single-port
RST_TYPE_A/B	Port A/B reset type selection	1 (synchronous)	0 = no reset 1 = synchronous
RST_RDATA_A/B	Default data value for Port A/B on active reset	All 1's	decimal value
WEN_SENSE_A/B	Port A/B write enable sense	1 (active high)	0 = active low 1 = active high
RADDR_LTNCY_A/B	Optional read address register select Port A/B	1	0 = no latency 1 = one cycle latency
RDATA_LTNCY_A/B	Optional read data register select Port A/B	1	0 = no latency 1 = one cycle latency

Copyright © 2013 Synopsys, Inc.

SYNCore ROM Compiler

The SYNCore ROM Compiler generates Verilog code for your ROM implementation. This section describes the following:

- Functional Overview, on page 1058
- Single-Port Read Operation, on page 1059
- Dual-Port Read Operation, on page 1060
- Parameter List, on page 1061
- Clock Latency, on page 1062

For further information, refer to the following:

- *Specifying ROMs with SYNCore*, on page 952 of the *User Guide*, for information about using the wizard to generate ROMs
- *Launch SYNCore Command*, on page 245 and *SYNCore ROM Wizard*, on page 264 for descriptions of the interface

Functional Overview

The SYNCore ROM component supports ROM implementations using block ROM or logic memory. For each configuration, design optimizations are made for optimum usage of core resources. Both single- and dual-port memory configurations are supported. Single-port ROM allows read access to memory through a single port, and dual-port ROM allows read access to memory through two ports. The following figure illustrates the supported signals for both configurations.

Copyright © 2013 Synopsys, Inc.

In the single-port (Port A) configuration, signals are synchronized to ClkA; ResetA can be synchronous or asynchronous depending on parameter selection. The read address (AddrA) and/or data output (DataA) can be registered to increase memory performance and improve timing. Both the read address and data output are subject to clock latency based on the ROM configuration (see *Clock Latency*, on page 1062). In the dual-port configuration, all Port A signals are synchronized to ClkA, and all PortB signals are synchronized to ClkB. ResetA and ResetB can be synchronous or asynchronous depending on parameter selection, and both data outputs can be registered and are subject to the same clock latencies. Registering the data output is recommended.

Note: When the data output is unregistered, the data is immediately set to its predefined reset value concurrent with an active reset signal.

Single-Port Read Operation

For single-port ROM, it is only necessary to configure Port A (see *Specifying ROMs with SYNCore*, on page 952 in the *User Guide*). The following diagram shows the read timing for a single-port ROM.

Copyright © 2013 Synopsys, Inc.

On every active edge of the clock when there is a change in address with an active enable, data will be valid on the same clock or next clock (depending on latency parameter values). When enable is inactive, any address change is ignored, and the data port maintains the last active read value. An active reset ignores any change in input address and forces the output data to its predefined initialization value. The following waveform shows the functional behavior of control signals in single-port mode.

When reset is active, the output data holds the initialization value (i.e., 255). When reset goes inactive (and enable is active), data is read form the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset are inactive, the output holds its previous read value.

Note: In the above timing diagram, reset is synchronous. Clock latency varies according to the implementation and parameters as described in *Clock Latency*, on page 1062.

Dual-Port Read Operation

Dual-port ROMs allow read access to memory through two ports. For dual-port ROM, both port A and port B must be configured (see *Specifying ROMs with SYNCore*, on page 952 in the *User Guide*). The following diagram shows the read timing for a dual-port ROM.

When either reset is active, the corresponding output data holds the initialization value (i.e., 255). When a reset goes inactive (and its enable is active), data is read form the addressed location of ROM. Reset has priority over enable and always sets the output to the predefined initialization value. When both enable and reset are inactive, the output holds its previous read value.

Note: In the above timing diagram, reset is synchronous. Clock latency varies according to the implementation and parameters as described in *Clock Latency*, on page 1062.

Parameter List

The following table lists the file entries corresponding to the ROM wizard parameters.

Name	Description	Default Value	Range
ADD_WIDTH	ROM address width value. Default value is 10	10	--
DATA_WIDTH	Read Data width, common to both Port A and Port B	8	2 to 256
CONFIG_PORT	Parameter to select Single/Dual configuration	dual (Dual Port)	dual (Dual), single (Single).

Copyright © 2013 Synopsys, Inc.

RST_TYPE_A	Port A reset type selection (synchronous, asynchronous)	1 - asynchronous	1(asyn), 0 (sync)
RST_TYPE_B	Port B reset type selection (synchronous, asynchronous)	1 - asynchronous	1 (asyn), 0 (sync)
RST_DATA_A	Default data value for Port A on active Reset	'1' for all data bits	0 – 2^DATA_WIDTH - 1
RST_DATA_B	Default data value for Port A on active Reset	'1' for all data bits	0 – 2^DATA_WIDTH - 1
EN_SENSE_A	Port A enable sense	1 – active high	0 - active low, 1- active high
EN_SENSE_B	Port B enable sense	1 – active high	0 - active low, 1- active high
ADDR_LTNCY_A	Optional address register select Port A	1- address registered	1(reg), 0(no reg)
ADDR_LTNCY_B	Optional address register select Port B	1- address registered	1(reg), 0(no reg)
DATA_LTNCY_A	Optional data register select Port A	1- data registered	1(reg), 0(no reg)
DATA_LTNCY_B	Optional data register select Port B	1- data registered	1(reg), 0(no reg)
INIT_FILE	Initial values file name	init.txt	--

Clock Latency

Clock latency varies with both the implementation and latency parameter values according to the following table. Note that the table reflects the values for Port A – the same values apply for Port B in dual-port configurations.

Implementation Type/Target	Parameter Value	Latency
block_rom (Xilinx) block_rom (Altera)	DATA_LTNCY_A = 0 ADDR_LTNCY_A = 1	1 ClkA cycle
	DATA_LTNCY_A = 1 ADDR_LTNCY_A = 0	1 ClkA cycle
	DATA_LTNCY_A = 1 ADDR_LTNCY_A = 1	2 ClkA cycles
logic (Xilinx) logic (Altera)	DATA_LTNCY_A = 0 ADDR_LTNCY_A = 0	0 ClkA cycles
	DATA_LTNCY_A = 0 ADDR_LTNCY_A = 1	1 ClkA cycle
	DATA_LTNCY_A = 1 ADDR_LTNCY_A = 0	1 ClkA cycle
	DATA_LTNCY_A = 1 ADDR_LTNCY_A = 1	2 ClkA cycles

Copyright © 2013 Synopsys, Inc.

SYNCore Adder/Subtractor Compiler

The SYNCore adder/subtractor compiler generates Verilog code for a parametrizable, pipelined adder/subtractor. This section describes the functionality of this block in detail.

Functional Description

The adder/subtractor has a single clock that controls the entire pipeline stages (if used) of the adder/subtractor.

As its name implies, this block just adds/subtracts the inputs and provides the output result. One of the inputs can be configured as a constant. The data inputs and outputs of the adder/subtractor can be pipelined; the pipeline stages can be 0 or 1, and can be configured individually. The individual pipeline stage registers include their own reset and enable ports.

The reset to all of the pipeline registers can be configured either as synchronous or asynchronous using the RESET_TYPE parameter. The reset type of the pipeline registers cannot be configured individually.

SYNCore adder/subtractor has ADD_N_SUB parameter, which can take three values ADD, SUB, or DYNAMIC. Based on this parameter value, the adder/subtractor can be configured as follows.

- Adder
- Subtractor
- Dynamic Adder and Subtractor

Adder

Based on the parameter CONSTANT_PORT, the adder can be configured in two ways.

- CONSTANT_PORT='0' – adder with two input ports (port A and port B)
- CONSTANT_PORT='1' – adder with one constant port

Adder with Two Input Ports (Port A and Port B)

In this mode, port A and port B values are added. Optional pipeline stages can also be inserted at port A, port B or at both port A and port B. Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, a number of the adder configurations are given below.

Adder with No Pipeline Stages – In this mode, the port A and port B inputs are added. The adder is purely combinational, and the output changes immediately with respect to the inputs.

Parameters: PORTA_PIPELINE_STAGE= ‘0’

PortA	0	4	9	13	5	1
PortB	0	1	3	5	2	5
PortOut	0	5	12	18	7	6

Adder with Pipeline Stages at Input Only – In this mode, the port A and port B inputs are pipelined and added. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

Copyright © 2013 Synopsys, Inc.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTB_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '0'

Adder with Pipeline Stages at Input and Output – In this mode, the port A and port B inputs are pipelined and added, and the result is pipelined. The result is valid only on the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTB_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '1'

Adder with a Port Constant

In this mode, port A is added with a constant value (the constant value can be passed though the parameter CONSTANT_VALUE). Optional pipeline stages can also be inserted at port A, Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the adder configurations are given below (here CONSTANT_VALUE='3')

Adder with No Pipeline Stages – In this mode, input port A is added with a constant value. The adder is purely combinational, and the output changes immediately with respect to the input.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’

Adder with Pipeline Stage at Input Only – In this mode, input port A is pipelined and added with a constant value. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’

Adder with Pipeline Stages at Input and Output – In this mode, input port A is pipelined and added with a constant value, and the result is pipelined. The result is valid only on the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’

Copyright © 2013 Synopsys, Inc.

Subtractor

Based on the parameter CONSTANT_PORT, the subtractor can be configure in two ways.

CONSTANT_PORT='0' – subtractor with two input ports (port A and port B)

CONSTANT_PORT='1' – subtractor with one constant port

Subtractor with Two Input Ports (Port A and Port B)

In this mode, port B is subtracted from port A. Optional pipeline stages can also be inserted at port A, port B, or both ports. Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the subtractor configurations are given below.

Subtractor with No Pipeline Stages – In this mode, input port B is subtracted from port A, and the subtractor is purely combinational. The output changes immediately with respect to the inputs.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTB_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’

Subtractor with Pipeline Stages at Input Only – In this mode, input port B and input PortA are pipelined and then subtracted. Because there is no pipeline stage at the output, the result is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’

Subtractor with Pipeline Stages at Input and Output – In this mode, input PortA and PortB are pipelined and then subtracted, and the result is pipelined. The result is valid only at the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’

Copyright © 2013 Synopsys, Inc.

Subtractor with a Port Constant

In this mode, a constant value is subtracted from port A (the constant value can be passed though the parameter CONSTANT_VALUE). Optional pipeline stages can also be inserted at port A, Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, a number of the subtractor configurations are given below (here CONSTANT_VALUE='1').

Subtractor with No Pipeline Stages – In this mode, a constant value is subtracted from port A. The subtractor is purely combinational, and the output changes immediately with respect to the input.

Parameters: PORTA_PIPELINE_STAGE= '0'
PORTOUT_PIPELINE_STAGE= '0'

Subtractor with Pipeline Stages at Input Only – In this mode, a constant value is subtracted from pipelined input port A. Because there is no pipeline stage at the output, the output is valid at each rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '0'

Subtractor with Pipeline Stages at Input and Output – In this mode, a constant value is subtracted from pipelined port A, and the output is pipelined. The result is valid only at the second rising edge of the clock.

Copyright © 2013 Synopsys, Inc.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’

Dynamic Adder/Subtractor

In dynamic adder/subtractor mode, port PortADDnSUB controls adder/subtractor operation.

PortADDnSUB='0' – adder operation

PortADDnSUB='1' – subtractor operation

Based on the parameter CONSTANT_PORT the dynamic adder/subtractor can be configured in one of two ways:

CONSTANT_PORT='0' – dynamic adder/subtractor with two input ports

CONSTANT_PORT='1' – dynamic adder/subtractor with one constant port

Dynamic Adder/Subtractor with Two Input Ports (Port A and Port B)

In this mode, the addition and subtraction is dynamic based on the value of input port PortADDnSUB. Optional pipeline stages can also be inserted at Port A, Port B, or both Port A and Port B. Optionally, pipeline stages can also be added at the output port. Depending on pipeline stages, some of the dynamic adder/subtractor configurations are given below.

Dynamic Adder/Subtractor with No Pipeline Registers – In this mode, the dynamic adder/subtractor is a purely combinational, and output changes immediately with respect to the inputs.

Copyright © 2013 Synopsys, Inc.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTB_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’

Dynamic Adder/Subtractor with Pipeline Stages at Input Only – In this mode, input port A and port B are pipelined and then added/subtracted based on the value of port PortADDnSUB. Because there is no pipeline stage at the output port, the result immediately changes with respect to the PortADDnSUB signal.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘0’

Dynamic Adder/Subtractor with Pipeline Stages at Input and Output – In this mode, input port A and port B are pipelined and then added/subtracted based on the value of port PortADDnSUB. Because the output port is pipelined, the result is valid only on the second rising edge of the clock.

Copyright © 2013 Synopsys, Inc.

Parameters: PORTA_PIPELINE_STAGE= ‘1’
PORTB_PIPELINE_STAGE= ‘1’
PORTOUT_PIPELINE_STAGE= ‘1’

Dynamic Adder/Subtractor with a Port Constant

In this mode, a constant value is either added or subtracted from port A based on input port value PortADDnSUB (the constant value can be passed though the parameter CONSTANT_VALUE). Optional pipeline stages can also be inserted at port A, Optionally, pipeline stages can also be added at the output port. Depending on the pipeline stages, a number of the dynamic adder/subtractor configurations are given below (here CONSTANT_VALUE=’1’).

Dynamic Adder/Subtractor with No Pipeline Registers – In this mode, dynamic adder/subtractor is a purely combinational, and the output change immediately with respect to the input.

Parameters: PORTA_PIPELINE_STAGE= ‘0’
PORTOUT_PIPELINE_STAGE= ‘0’

Copyright © 2013 Synopsys, Inc.

Dynamic Adder/Subtractor with Pipeline Stages at Input Only – In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because there is no pipeline stage on the output port, the result changes immediately with respect to the PortADDnSUB signal.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '0'

Dynamic Adder/Subtractor with Pipeline Stages at Input and Output – In this mode, a constant value is either added or subtracted from the pipelined version of port A based on the value of port PortADDnSUB. Because the output port is pipelined, the result is valid only on the second rising edge of the clock.

Parameters: PORTA_PIPELINE_STAGE= '1'
PORTOUT_PIPELINE_STAGE= '1'

Dynamic Adder/Subtractor with Carry Input

The following waveform shows the behavior of the dynamic adder/subtractor with a carry input (the carry input is assumed to be 0).

Dynamic Adder/Subtractor with Complete Control Signals

The following waveform shows the complete signal set for the dynamic adder/subtractor. The enable and reset signals are always present in all of the previous cases.

Copyright © 2013 Synopsys, Inc.

SYNCore Counter Compiler

The SYNCore counter compiler generates Verilog code for your up, down, and dynamic (up/down) counter implementation. This section describes the following:

- Functional Overview, on page 1076
- UP Counter Operation, on page 1077
- Down Counter Operation, on page 1078
- Dynamic Counter Operation, on page 1078

For further information, refer to the following:

- *Specifying Counters with SYNCore*, on page 964 of the *User Guide*, for information about using the wizard to generate a counter core.
- *Launch SYNCore Command*, on page 245 and *SYNCore Counter Wizard*, on page 272 for descriptions of the interface and generating the core.

Functional Overview

The SYNCore counter component supports up, down, and dynamic (up/down) counter implementations using DSP blocks or logic elements. For each configuration, design optimizations are made for optimum use of core resources.

As its name implies, the COUNTER block counts up (increments) or down (decrements) by a step value and provides an output result. You can load a constant or a variable as an intermediate value or base for the counter. Reset to the counter on the PortRST input is active high and can be can be configured either as synchronous or asynchronous using the RESET_TYPE parameter. Count enable on the PortCE input must be value high to enable the counter to increment or decrement.

Copyright © 2013 Synopsys, Inc.

Note: Altera Stratix II and Stratix III DSP blocks have registers with asynchronous resets only. Xilinx Virtex4 and Virtex5 DSP blocks have registers with synchronous resets only; Xilinx Spartan3A DSP block registers can be configured to have either synchronous or asynchronous resets. Please configure the multiplier accordingly while targeting to these technologies for effective utilization of the DSP blocks.

UP Counter Operation

In this mode, the counter is incremented by the step value defined by the STEP parameter. When reset is asserted (when PostRST is active high), the counter output is reset to 0. After the assertion of PortCE, the counter starts counting upwards coincident with the rising edge of the clock. The following waveform is with a constant STEP value of 5 and no load value.

Parameters: MODE= 'Up'
LOAD= '0'

Note: Counter core can be configured to use a constant or dynamic load value in Up Counter mode (for the counter to load the PortLoadValue, PortCE must be active). This functionality is explained in *Dynamic Counter Operation*, on page 1078.

Copyright © 2013 Synopsys, Inc.

Down Counter Operation

In this mode, the counter is decremented by the step value defined by the STEP parameter. When reset is asserted (when PostRST is active high), the counter output is reset to 0. After the assertion of PortCE, the counter starts counting downwards coincident with the rising edge of the clock. The following waveform is with a constant STEP value of 5 and no load value.

Parameters: MODE= 'Down'
LOAD= '0'

Note: Counter core can be configured to use a constant or dynamic load value in Down Counter mode (for the counter to load the PortLoadValue, PortCE must be active). This functionality is explained in *Dynamic Counter Operation*, on page 1078.

Dynamic Counter Operation

In this mode, the counter is incremented or decremented by the step value defined by the STEP parameter; the count direction (up or down) is controlled by the PortUp_nDown input (1 = up, 0 = down).

Dynamic Up/Down Counters with Constant Load Value*

On de-assertion of PortRST, the counter starts counting up or down based on the PortUp_nDown input value. The following waveform is with STEP value of 5 and a LOAD_VALUE of 80. When PortLoad is asserted, the counter loads the constant load value on the next active edge of clock and resumes counting in the specified direction.

Copyright © 2013 Synopsys, Inc.

Parameters: MODE= ‘Dynamic’
LOAD= ‘1’

Note: *For counter to load the PortLoadValue, PortCE must be active.

Dynamic Up/Down Counters with Dynamic Load Value*

On de-assertion of PortRST, the counter starts counting up or down based on the PortUp_nDown input value. The following waveform is with STEP value of 5 and a LOAD_VALUE of 80. When PortLoad is asserted, the counter loads the constant load value on the next active edge of clock and resumes counting in the specified direction.

In this mode, the counter counts up or down based on the PortUp_nDown input value. On the assertion of PortLoad, the counter loads a new PortLoadValue and resumes up/down counting on the next active clock edge. In this example, a variable PortLoadValue of 8 is used with a counter STEP value of 5.

Parameters: MODE= ‘Dynamic’
LOAD= ‘2’

Copyright © 2013 Synopsys, Inc.

Note: * For counter to load the PortLoadValue, PortCE should be active.

Encryption Scripts

Two Synopsys FPGA IP encryption methods are available to Synplify Pro and Synplify Premier users:

- P1735
- OpenIP

With either of these encryption methods, the IP vendor is able to encrypt and control distribution of their IP from their own website. The synthesis user has access from the synthesis tool to the IP that the vendor makes available for download and evaluation within a synthesis design.

The two encryption methods are supported by two scripts:

- The encryptP1735 Script, on page 1082
- The encryptIP Script, on page 1090

Encryption and Decryption Methodologies

This section describes common encryption schemes. There are two major classes of encryption/decryption algorithms: symmetric, and asymmetric.

- Symmetric Encryption
  With this kind of encryption, a special number is used as a key to encrypt the files. The same key is used to decrypt the file, so you must have access to the same key.

  Examples of this kind of algorithm are

  – Data Encryption Standard (DES)

  – Triple DES or TDES or TDEA (Triple Data Encryption Algorithm), which uses DES three times

  – Advanced Encryption Standard (AES)

Copyright © 2013 Synopsys, Inc.

- Asymmetric Encryption
  With this encryption scheme, you use different keys to encode and decode data. The end user generates the keys and makes a public key to everyone who needs it for encryption. The public key cannot be used for decryption. The end user uses the private key to decrypt the data. Examples of asymmetric encryption are DH (Diffie-Hellman) and RSA (Rivest, Shamir, and Adelman).

The encryptP1735 Script

The encryptP1735 script is one of the Synopsys FPGA IP scripts that is available to IP vendors who wish to provide evaluation IP to their synthesis users. The script is a Perl script that allows IP vendors to encrypt modules or components that can then be downloaded for evaluation or use by a Synopsys FPGA user.

Syntax for Running encryptP1735

This is the syntax for running the encryptP1735 script:

encryptP1735
  -in | **input** *inputFileName*
  -out | **output** *outputFileName*
  -pk | **public_keys** *keyFileName*
  -k | key *sessionKeyInTextFormat*
  -kv | **key_version** *minKeyVersion*
  -kx | **key_in_hex** *symmetricEncryptionKeyInHexadecimalFormat*
  -om | **output_method "{plaintext** | **blackbox** | **encrypted}"**
  -av | **allowed_vendor** *vendorString*

-tier "{pro | premier}"
-sk | showkey
-v | verbose 0 | 1
-verilog
-vhdl

Script Command-Line Arguments

The following table provides additional detail on the encryptP1735 script command-line arguments.

-in | input	Specifies the input file; *inputFileName* is a non-encrypted HDL file containing one or more encryption envelopes.
-out | output	Specifies the encrypted file; *outputFileName* is the encrypted HDL file from running the encryption script.
-pk | public_keys	Specifies the public keys repository file. This file contains public keys for various tools. If the encryption envelope contains a key block with a particular keyowner and keyname, the public keys file is searched by the script to find a public key to use during key-block generation.
-k | key	Specifies the session key in text format.
-kv | key_version	Specifies the minimum encryption version required from the Synplify software used to run the IP.
-kx | key_in_hex	Optional parameter. Specifies the symmetric encryption key in hexadecimal format. If omitted, keys are randomly generated.
-om | output_method	Determines how the IP is treated in the output netlist after synthesis: • plaintext specifies that the IP is unencrypted in the synthesis netlist. • blackbox specifies that the IP is treated as a black box (the IP is not written), and only interface information is present in the output netlist. • encrypted specifies that the output netlist for the IP will be encrypted using the same session key as input. See *Output Methods for encryptIP*, on page 1092 for more information.
-av | allowed_vendor	Adds a requirement that IP be synthesized only for given a specific FPGA vendor. Value is a vendor string such as altera or microsemi. (To be supported in a future release.)

Copyright © 2013 Synopsys, Inc.

-tier	Specifies the minimum tool tier to be used for the encryption. Acceptable values are pro and premier (a file encrypted with premier can only be decrypted in Synplify Premier).
-sk | showkey	When used, the encryption script displays the session key in use (useful when random keys are used and the user wants to make a note of the key being used).
-v | verbose	Specifies that the encryption script is to be run in verbose mode (displays more detailed messages) when set to 1.

Input Files

The encryptP1735 encryption script reads an HDL file containing one or more encryption envelopes. Additionally it reads the keys repository file that contains the public keys for the IP consumer tools. This section provides examples and some details on these file formats.

Encryption Envelope Verilog Example

```
module secret (a, b, clk);
input a, clk;
output b;

`pragma protect version=1
`pragma protect encoding=(enctype="base64")
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl",
   encrypt_agent_info="Synplify encryption scripts"
`pragma protect key_keyowner="Synplicity",
   key_keyname="SYNP05_001", key_method="rsa", key_block
`pragma protect key_keyowner="ACME", key_keyname="ACME_2012",
   key_method="rsa", key_block
`pragma protect data_keyowner="ip-vendor-a",
   data_keyname="fpga-ip", data_method="des-cbc"
`pragma protect begin
reg b=0;

always @(posedge clk) begin
   b = a;
end
`pragma protect end
endmodule
```

Encryption Envelope VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity secret is
   port (clk : in std_logic;
         a : in std_logic;
         b : out std_logic );
end entity;

architecture rtl of secret is
`protect version=1
`protect author="author-a", author_info="author-a-details"
`protect encrypt_agent="encryptP1735.pl",
   encrypt_agent_info="Synplify encryption scripts"
`protect encoding=(enctype="base64")
`protect key_keyowner="Synplicity", key_keyname="SYNP05_001",
   key_method="rsa", key_block
`protect key_keyowner="ACME", key_keyname="ACME_2012",
   key_method="rsa", key_block
`protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
   data_method="des-cbc"
`protect begin
   signal b_reg: std_logic;
   begin
   process (clk) is
      begin
      if rising_edge(clk) then
         b_reg <= a;
      end if;
   end process;
   b <= b_reg;
`protect end
end architecture;
```

Public Keys Repository File

The encryptP1735 encryption script requires public keys from the file specified by the –public_keys option. This file includes public keys for each of the tools that require a key block in the encrypted file. The public keys file includes a Synopsys synthesis tool public key; the file can be expanded by the user to include public keys for other tools. The following is an example of a public keys file.

```
`pragma protect key_keyowner="Synplicity",
   key_keyname="SYNP05_001", key_public_key
```


Copyright © 2013 Synopsys, Inc.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAybsQaMidiCHZyh14wbXn
UpP8lK+jJY5oLpGqDfSW5PMXBVp0WFd1d32onXEpRkwxEJLlK4RgS43d0FG2ZQ1l
irdimRKNnUtPxsrJzbMr74MQkwmG/X7SEe/lEqwK9Uk77cMEncLycI5yX4f/K9Q9
WS5nLD+Nh6BL7kwR0vSevfePC1fkOa1uC7b7Mwb1mcqCLBBRP9/eF0wUIoxVRzjA
+pJvORwhYtZEhnwvTblBJsnyneT1LfDi/D5WZoikTP/0KBiP87QHMSuVBydMA7J7
g6sxKB92hx2Dpv1ojds1Y5ywjxFxOAA93nFjmLsJq3i/P0lv5TmtnCYX3Wkryw4B
eQIDAQAB
-----END PUBLIC KEY-----

`pragma protect key_keyowner="ACME", key_keyname="ACME_2012",
   key_public_key

-----BEGIN PUBLIC KEY-----
irdimRKNnUtPxsrJzbMr74MQkwmG/X7SEe/lEqwK9Uk77cMEncLycI5yX4f/K9Q9
+pJvORwhYtZEhnwvTblBJsnyneT1LfDi/D5WZoikTP/0KBiP87QHMSuVBydMA7J7
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAybsQaMidiCHZyh14wbXn
UpP8lK+jJY5oLpGqDfSW5PMXBVp0WFd1d32onXEpRkwxEJLlK4RgS43d0FG2ZQ1l
WS5nLD+Nh6BL7kwR0vSevfePC1fkOa1uC7b7Mwb1mcqCLBBRP9/eF0wUIoxVRzjA
g6sxKB92hx2Dpv1ojds1Y5ywjxFxOAA93nFjmLsJq3i/P0lv5TmtnCYX3Wkryw4B
eQIDAQAB
-----END PUBLIC KEY-----
```

The two pragma attributes at the top of each public key are required for the encryption script to correctly identify the key. Values for these attributes must match the values in the key-block section of the encryption envelope.

Output Files

The following examples show encrypted HDL files in both Verilog and VHDL formats. As shown in the examples, the primary difference between the two formats is the protect directive syntax.

Decryption Envelope Verilog Example

```
module secret (a, b, clk);
input a, clk;
output b;

`pragma protect begin_protected
`pragma protect version=1
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl",
   encrypt_agent_info="Synplify encryption scripts"
`pragma protect key_keyowner="Synplicity",
   key_keyname="SYNP05_001", key_method="rsa"
```

```
`pragma protect encoding=(enctype="base64", line_length=76,
   bytes=256)
`pragma protect key_block
   GcPSRCuP+M1OEr5GO+V9eN/oofJ2UUsIo+HL4TGYpjBLsZKpfSw4kEyWfzx
   GmIKFH1BgyV5f6eEyZAq8Gvb7cgnZODDIVtULwS+2wN2nOoLpcitq0QfNx8
   3Z6ofbcyksSNwjmeZlcJfz3WFJA3soauFuz3vDEqsAO+kjk6Sb5mAd2zVhj
   oMriVNEkDZJvUvIGxa2onHHkzPrg6jX1uFD7dUWLkL9Re/wfcF7cqps5Zj/
   p8Pci/xKixWjj5ZEiXuEpax9k8YfL1/p+4jS7WeQsUd36Ateus4p6QyFS2T
   wixHURtYMhSFEWkLKLOkhAr+6zieXYekm0QkEjzatXKUULg==
`pragma protect key_keyowner="ACME", key_keyname="ACME_2012",
   key_method="rsa"
`pragma protect encoding=(enctype="base64", line_length=76,
   bytes=256)
`pragma protect key_block
   cqps5Zj/p8Pci/xKixWjj5ZEiXuEpax9k8YfL1/p+4jS7WeQsUd36Ateus4
   p6QyFS2TwixHURtYMZAq8Gvb7cgnZODDIVtULwS+2wN2nOoLpcitq0QfNx8
   3Z6ofbcyksSNwjmeZlcJfz3WFJA3soauFuz3vDEqsAO+kjk6Sb5mAd2zVhj
   oMriVNEkDZJvUvIGxa2onHHkzPrg6jX1uFD7dUWLkL9Re/wfcF7GcPSRCuP
   +M1OEr5GO+V9eN/oofJ2UUsIo+HL4TGYpjBLsZKpfSw4kEyWfzxGmIKFH1B
   gyV5f6eEyhSFEWkLKLOkhAr+6zieXYekm0QkEjzatXKUULg==
`pragma protect data_keyowner="ip-vendor-a",
   data_keyname="fpga-ip", data_method="des-cbc"
`pragma protect encoding=(enctype="base64",
   line_length=76, bytes=64)
`pragma protect data_block
   MDEyMzQ1NjcC5VWXO6Vh6o+3A3yw8KvVF5Zq4KeMauCbrw4GGNHx3rLBKH/
   fp5PuY1apN0B708D6aYE8oUtAWg==
`pragma protect end_protected
endmodule
```

Decryption Envelope VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity secret is
   port (clk : in std_logic;
         a : in std_logic;
         b : out std_logic );
end entity;

architecture rtl of secret is
`protect begin_protected
`pragma protect version=1
`protect author="author-a", author_info="author-a-details"
`protect encrypt_agent="encryptP1735.pl",
   encrypt_agent_info="Synplify encryption scripts"
```


Copyright © 2013 Synopsys, Inc.

```
`protect key_keyowner="Synplicity", key_keyname="SYNP05_001",
   key_method="rsa"
`protect encoding=(enctype="base64", line_length=76, bytes=256)
`protect key_block
   HVAvdUaZelTLSES2j5o+Eb61UTZb09sF82LSLJU2Qr4SxNDTz5gZZWoNq93
   /ux4+oVZdSYIPRHO1MX5h1oJ3pG0VPoyMQxX7n7YRcFETjH4cV42ajhqNIL
   x+NhBOnFJiJnZYPMHVwvuWj85pd8wkk75kxUouRZ2TW4ra21KeIsOcJ9Lz9
   ZVR2g01jkSlZOzqv2k4kMEDLTt47cBvxKFsAqSxrXnrk8aMtwHiBHsfdEtz
   El38OdvMAVfbe4uiMgkPAoixcZyCeyTUnn7xPJdVg06PYWKeWju26iI4LYl
   oeoI5+q9tagNYLHF3Ui3uPx4p48WGm6TgVwywptbHgZcstg==
`protect key_keyowner="ACME", key_keyname="ACME_2012",
   key_method="rsa"
`protect encoding=(enctype="base64", line_length=76, bytes=256)
`protect key_block
   xUouRZ2TW4ra21KeIsOcJ9Lz9ZVR2g01jkSlZOzqv2k4kMEDLTt47cBvxKF
   sAqSxrXnrk8aMtwHiBHsfdEtzEl38OdvMAVfbe4uiMgkPAoixcZyCeyTUnn
   7xPJdVg06PYWKeWju26iI4LYloeoI5+q9tHVAvdUaZelTLSES2j5o+Eb61U
   TZb09sF82LSLJU2Qr4SxNDTz5gZZWoNq93/ux4+oVZdSYIPRHO1MX5h1oJ3
   pG0VPoyMQxX7n7YRcFETjH4cV42ajhqNILx+NhBOnFJiJnZYPMHVwvuWj85
   pd8wkk75kagNYLHF3Ui3uPx4p48WGm6TgVwywptbHgZcstg==
`protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
   data_method="des-cbc"
`protect encoding=(enctype="base64", line_length=76, bytes=176)
`protect data_block
   MDEyMzQ1NjeNrSh8zBeceFMAyESDTaD3I3dwKZpS1EQEBIV9iMGnYGxjeaH
   DhHzq861/g8ULF7xLpsOYt1MbNVaxNfrXntS5JxqNQT/kGqLF5QuywGavdR
   Y9TjWw/DOaAVIeI35WUoIcNwsbCGjnJHXtGJM/QQL4Qb1YzU4EmxnrVlpoz
   z4bJAjgCjFG614O1B2ef/5BQTpZJ1YXLV51Yx6eB9XyM7VN4cCFlySlHCw=
`protect end_protected
end architecture;
```

Output Methods for encryptP1735

You can add encrypted HDL files to a Synplify project, like any other HDL files, and you can control the level of IP protection in the synthesis output netlist by specifying an output method when you encrypt your IP with the encryptP1735 script. The encrypted module in the Synplify output netlist can be in any of the three states: clear-text, encrypted, or black-box. The state of the output netlist depends on the value of output_method attribute applied during encryption. The output method is included in the encrypted key block of your encrypted RTL. The table below shows the values for the output_method attribute. See *Specifying the Script Output Method*, on page 979 in the *User Guide* for guidelines on how to use these methods effectively.

Output Method	Description
blackbox	The output netlist contains only the IP interface and no IP content (the netlist includes only ports and connections, and there are no nets or instances shown inside the IP). With this method, you cannot run gate-level simulation on the output netlist nor can you run physical synthesis or place and route the IP.
plaintext	With this method, you can run synthesis, gate-level simulation, place and route, and implement an FPGA on the board that includes your IP. The output netlist contains your unencrypted IP, which is completely readable.
encrypted	The output netlist for the IP is encrypted using the same session key that was used to encrypt your IP for the synthesis tools.

Pragmas Used in the encryptP1735 Script

The header blocks in the encryptP1735 script use the pragmas described in the following table.

Pragma Keyword	Description
begin	Opens a new encryption envelope
end	Closes an encryption envelope
begin_protected	Opens a new decryption envelope
end_protected	Closes a decryption envelope
author	Identifies the author of an envelope
author_info	Specifies additional author information
encoding	Specifies the coding scheme for the encrypted data
data_keyowner	Identifies the owner of the data encryption key
data_method	Identifies the data encryption algorithm
data_keyname	Specifies the name of the data encryption key
data_public_key	Specifies the public key for data encryption
data_decrypt_key	Specifies the data session key

Copyright © 2013 Synopsys, Inc.

Pragma Keyword	Description
data_block	Begins an encoded block of encrypted data
digest_method	Specifies the digest computation algorithm
digest_block	Specifies a digest for data integrity
key_keyowner	Identifies the owner of the key encryption key
key_method	Specifies the key encryption algorithm
key_keyname	Specifies the name of the key encryption key
key_public_key	Specifies the public key for key encryption
key_block	Begins an encoded block of key data
comment	Uninterpreted documentation string
reset	Restores all pragma keyword values to their defaults
version	P1735 encryption version

The encryptIP Script

The encryptIP script is a Synopsys FPGA IP script that is provided to IP vendors who wish to provide evaluation IP to synthesis users. The script is a Perl script and lets the IP vendor encrypt modules or components that can be downloaded for evaluation or use by the Synopsys FPGA user. You can download the encryptIP Perl script from SolvNet. See the article published at:

https://solvnet.synopsys.com/retrieve/032343.html

For details, see the following:

- Syntax for Running encryptIP, on page 1091
- Output Methods for encryptIP, on page 1092
- The encryptIP Script Run, on page 1095
- Pragmas Used in the encryptIP Script, on page 1098 in the *User Guide*

Syntax for Running encryptIP

This is the syntax for running the encryptIP script:

encryptIP
-in | **input** *inputFile*
-out | **output** *outputFileName*
-c | **cipher "{des-cbc** | **3des-cbc** |**aes128-cbc** |**blowfish-cbc}"**
-k | **key** *symmetricEncryptionKeyInTextFormat*
-kx | **keyx** *symmetricEncryptionKeyInHexadecimalFormat*
-bd | **build_date** *ddmmmyyyy*
-om | **outputmethod "{plaintext** | **blackbox** | **persistent_key}"**
-incv | **includevendor** *vendorKeyBlock*
-dkn | **datakeyname** *sessionKeyName*
-dko | **datakeyowner** *sessionKeyOwner*
-a | **author** *dataAuthor*
-v | **verbose**

The following table describes the parameters. You must specify all the parameters, except for the ones that are optional.

-in | input	Names the input RTL file to be encrypted.
-out | output	Names the output file generated after encryption.
-c | cipher	Specifies the symmetric encryption cipher. See *Encryption and Decryption Methodologies*, on page 1081 for details. The keylength must match the algorithm being used, with each character using 8 bits. • des-cbc specifies the Data Encryption Standard (DES); uses a 64-bit key. • 3des-cbc specifies the Triple Data Encryption Standard (Triple DES); uses a 192-bit key. • aes128-cbc specifies the Advanced Encryption Standard (AES Rijndael); uses a 128-bit key. • blowfish-cbc specifies the Blowfish standard, and is used for Lattice designs only. For information about how symmetric encryption fits into the encryption flow, see *Encryption and Decryption*, on page 971 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

-k | key	Specifies the symmetric data decryption key used to encode your RTL data block. The key is in text format, and can be any string (e.g. ABCDEFG).The exact length of the key depends on the data method you use. In a future release, this key will be automatically generated if you do not specify one.
-kx | keyx*	Optional parameter. Specifies the symmetric encryption key in hexadecimal format.
-bd | build_date	Specifies a date (ddmmmyyyy). The IP only works in Synplicity software released after the specified date. It is recommended that you use a date in January 2008 or later. For example:16FEB2008. This option lets you force users to use newer Synopsys FPGA releases that contain more security features. Contact Synplicity if you need help in deciding what build date to use.
-om | outputmethod	Determines how the IP is treated in the output after synthesis: • plaintext specifies that the IP is unencrypted in the synthesis netlist. • blackbox specifies that the IP is treated as a black box, and only interface information is in the output. • persistent_key is the default setting. It is used for Lattice designs, and re-encrypts the output after synthesis in accordance with the OpenIP standard. See *Output Methods for encryptIP*, on page 1092 for more information.
-incv | includevendor	Optional parameter that specifies a key block for an EDA vendor, so that IP can be read by the vendor tools. C
-dkn | datakeyname	Specifies a string that denotes your session key, that was used to encrypt your IP.
-dko | datakeyowner	Optional parameter that names the owner of the session key. The value can be any string.
-a | author	Optional parameter that names the author of the session key. The value can be any string.
-v | verbose	Specifies that the script run in verbose mode.

Output Methods for encryptIP

You can control the level of IP protection in the synthesis output netlist by specifying an output method when you encrypt your IP by running the encryptIP script. For example, `-om blackbox`. The output method is included

in the encrypted key block of your encrypted RTL. The table below shows the values for `-om`. See *Specifying the Script Output Method*, on page 979 in the *User Guide* for guidelines on how to use these methods effectively.

Output Method	Description
blackbox	With this method, you cannot run gate-level simulation on the output netlist nor can you run physical synthesis or place and route the IP. This is because the output netlist contains the IP interface only and no IP contents. It only includes ports and connections; there are no nets or instances shown inside the IP.
plaintext	With this method, you can synthesize, run gate-level simulation, place, route, and implement an FPGA on the board that includes your IP. The output netlist contains your unencrypted IP, which is completely readable.
persistent_key	By default, the output is encrypted using the same session key and cipher you used to encrypt your IP for the synthesis tools.

Copyright © 2013 Synopsys, Inc.

The following figure shows how the synthesis tools work with each of the three encryptIP output methods:

Effect of Output Method on Viewing IP

In the synthesis tools, the contents of the IP are always shown as black boxes, and you cannot view the contents. In the Technology view, you cannot view the initialization values for the LUT.

Copyright © 2013 Synopsys, Inc.

Effect of Output Method on Output Constraints

After synthesis, the output constraints generated for the IP are not encrypted, regardless of the output method. They are always readable.

Effect of Output Method on Output Netlist

The following table summarizes how different output methods affect what is output:

Method (-om)	Output Netlist After Synthesis
blackbox	The output netlist contains the IP interface only and no IP content, and 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 content applies to all netlist formats generated for different vendors, whether it is HDL (`vm` or `vhm`), EDIF (`edf` or `edn`), or `vqm`. For Lattice technologies, the netlist includes one output EDIF file with IP blocks encrypted with the remainder of the netlist readable; for Xilinx technologies, the netlist includes a top-level unencrypted EDIF file plus a hierarchy of EDIF files where the internals of the IP blocks are empty.
plaintext	The output netlist contains your unencrypted IP, which is completely readable (nothing is encrypted).
persistent_key	The output netlist includes encrypted versions of the IP. For Lattice designs, the tool generates one output netlist (EDIF or HDL) that includes the encrypted IP blocks. The netlist is readable, except for the IP block sections, which are encrypted. For Xilinx designs, the netlist includes a top-level unencrypted EDIF file plus a hierarchy of EDIF files where the internals of the IP blocks are empty

In future releases, you will be able to specify IP rights at the time of encryption, which will determine how your IP can be viewed in the RTL and Technology views.

The encryptIP Script Run

The following example explains the encryptIP script, in the order in which it executes. For descriptions of the pragmas used in the encryptIP script, see *Pragmas Used in the encryptIP Script*, on page 1098.

Copyright © 2013 Synopsys, Inc.

1. For each RTL file, the script creates a data block using symmetric algorithm and your own session key.

   You can use any of the CBC encryption modes listed in *Syntax for Running encryptIP*, on page 1091. The initialization vector is a constant, and the block is encoded in base 64. The following excerpt uses the data encryption key ABCDEFG:

   ```
   %%% protect data_block
   UWhcm3CPmGz27DXAWQZF8rY7hSsvLwedXiP59HYZHJfoMIMkJ0W+6H7vmJEXZ/
      dTxnAgGB2YJCF7lsaZ6x6kRisdtBtIo8+0Glskcykt7FtpAjpz24cJ9hoSYMu
      HCmG70dHDNzTHWjkwBs2fxo5S6559d3pW+SDutrvrsHntyvHYiqxUZPsGce
      ZZQJpqQIpqo24uFCVHNSX/URvL47CUBWoKB2XEpyRv5Zgd1F52YjBLIpET
      +kBEzutAorF5rD9eZSALSo0kvVb7MPXFxmfCF8wHwTnRtkPthNCMq0t3iCgf9EH
   ```

2. The script precedes the data block with a small data block header that describes the data method:

   ```
   %%% protect data_method=aes128-cbc
   %%% protect data_block
   UWhcm3CPmGz27DXAWQZF8rY7hSsvLwedXiP59HYZHJfoMIMkJ0W+6H7vmJEXZ/
      dTxnAgGB2YJCF7lsaZ6x6kRisdtBtIo8+0Glskcykt7FtpAjpz24cJ9hoSYMu
      HCmG70dHDNzTHWjkwBs2fxo5S6559d3pW+SDutrvrsHntyvHYiqxUZPsGce
      ZZQJpqQIpqo24uFCVHNSX/URvL47CUBWoKB2XEpyRv5Zgd1F52YjBLIpET
      +kBEzutAorF5rD9eZSALSo0kvVb7MPXFxmfCF8wHwTnRtkPthNCMq0t3iCgf9EH
   ```

3. The script then prepares a key block for Synplify, which contains your session key and Synplicity-specific directives. Note that the data_decrypt_key is required.

   `output_method=blackbox` (Your IP will be a black box in the output netlist)
   `data_decrypt_key=ABCDEFG` (Session key you used to encrypt your data block)

   See *Output Methods for encryptIP*, on page 1092 for information about output methods.

4. Use Synplify Public key (Synplicity has an executable that returns this key) and RSA2048 assymmetic encryption to create a key block. Encode it in base64.

   ```
   %%% protect key_block
   U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJDfonqfqzjzEQ+xQ4
      wyh65wo6X56Jm+ClaVuZjgQKK0c4y47nyA1iWcuq1Nh6KeuUscxp+nL6yT9Am
      +nv+c57jSCMG0QsFbRBAIhdlohQAbYbSIuFxdLFEFxW4znF3+YDAsMHeIs
      1tqxKqhQzYQ2fGJdQz0NVRi1hFjx/RpGmoXmzvSTX2xsre+ZNDh3r9qvj37
      QGwLH2erPt/iXcUVnlnPCOaV5z8M1YLrKY8ui7KBs/HhyP7L2mAMPQAFY3i
      DhycUcJ5sirBgKZycpkhP8jQ02yjTZMb7z9KyYTHrzDdA==
   ```

```
%%% protect key_block
+700tKshqQgTmb8UdRU9n263KwF7RWb8GSz7Cwo6X56Jm+ClxISekIJDfon
   qfqzjzEQ+xQ4wyh657nyA1iWcuq1Nh6KeuUscxp+nL6yT9AaVuZjG0QsFb
   RBAIhdlohQAbYbgQKK0c4y4m+nv+c57jSCMxW4znF3+YDAsMHeIs1tqx
   KqhQzYQ2fGJdQz0NXmzvSTX2xsre+ZNVRi1hFjSIuFxdLFEFx/RpGmo9qvj
   37QGwLH2erPt/iXcUVnlnPCO7KBs/HhyP7L2mAMPQAFY3iDhycUcJaV5z8MD
   h3r1YLrKY8ui8jQ02yjTZMb7z9pkhPYTHrzDdAKy5sirBgKZyc==
```

5. The script adds a small key block header to each key block.

```
%%% protect key_keyowner=Synplicity
%%% protect key_keyname=SYNP05_001
%%% protect key_block
U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJDfonqfqzjzEQ+xQ4
```

6. Your final encrypted IP key and data blocks looks like this:

```
%%% protect protected_file 1.0
<optional unencrypted HDL>
%%% protect begin_protected
%%% protect key_keyowner=Synplicity
%%% protect key_keyname=SYNP05_001

%%% protect key_block
U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJDfonqfqzjzEQ+xQ4
   wyh65wo6X56Jm+ClaVuZjgQKK0c4y47nyA1iWcuq1Nh6KeuUscxp+nL6yT9Am
   +nv+c57jSCMG0QsFbRBAIhdlohQAbYbSIuFxdLFEFxW4znF3+YDAsMHeIs
   1tqxKqhQzYQ2fGJdQz0NVRi1hFjx/RpGmoXmzvSTX2xsre+ZNDh3r9qvj37
   QGwLH2erPt/iXcUVnlnPCOaV5z8M1YLrKY8ui7KBs/HhyP7L2mAMPQAFY3i
   DhycUcJ5sirBgKZycpkhP8jQ02yjTZMb7z9KyYTHrzDdA==

<other key blocks>

%%% protect data_method=aes128-cbc
%%% protect data_block
UWhcm3CPmGz27DXAWQZF8rY7hSsvLwedXiP59HYZHJfoMIMkJ0W+6H7vmJEXZ/
   dTxnAgGB2YJCF7lsaZ6x6kRisdtBtIo8+0Gls/kcykt7FtpAjpz24cJ9ho
   SYMuHCmG70dHDNzTHWjkwBs2fxo5S6559d3pW+SDutrvrsHntyv
   HYiqxUZPsGceZZQJpqQIpqo24uFCVHNSX/URvL47CUBWoKB2XEpyRv5Zg
   …

%%% protect end_protected
<optional unencrypted HDL>
```


Copyright © 2013 Synopsys, Inc.

Pragmas Used in the encryptIP Script

The header blocks in the encryptIP script use the pragmas described in the following tables. Note the following:

- The %%% protect directive must be placed at the exact beginning of a line.
- Exactly one white-space character must separate the %%% from the command that follows

The following table describes the general pragmas used:

%%% protect protected_file 1.0	Line 1 of protected file
%%% protect begin_protected	Begin protected section
%%% protect end_protected	Ends protected section
%%% protect comment *comment*	Single line plain-text comment
%%% protect begin_comment	Begin block of plain-text comments
%%% protect end_comment	End block of plain-text block comments

The following table describes the data block pragmas:

%%% protect author=*string*	Arbitrary string
%%% protect data_method=*string*	For all supported FPGA vendors, one of the DES encryption methods described in *Syntax for Running encryptIP*, on page 1091.
%%% protect data_block	Immediately precedes encrypted data block

The following table describes the key block pragmas:

%%% protect key_keyowner=*string*	Arbitrary string
%%% protect key_keyname=*string*	Name recognized by Synplify to select key block
%%% protect key_method=*string*	Encryption algorithm. Currently we support “rsa”
w %%% protect key_block	Immediately precedes encrypted data block

Copyright © 2013 Synopsys, Inc.

CHAPTER 14

Timing Constraint Syntax

The following describe Tcl equivalents for the timing constraints you specify in the SCOPE editor or manually in a constraint file.

- FPGA Timing Constraints, on page 1100
- Synplify-Style Timing Constraints (Legacy), on page 1135

Copyright © 2013 Synopsys, Inc.

FPGA Timing Constraints

The FPGA synthesis tools support FPGA timing constraints for a subset of the clock definition, I/O delay, and timing exception constraints.

For more information about using FPGA timing constraints with your project, see *Using the SCOPE Editor*, on page 130 in the *User Guide*.

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.

The remainder of this section describes the constraint file syntax for the following FPGA timing constraints in the FPGA synthesis tools. For constraint file syntax for the legacy timing constraints, see *Synplify-Style Timing Constraints (Legacy)*, on page 1135.

- create_clock
- create_generated_clock
- reset_path
- set_clock_groups
- set_clock_latency
- set_clock_route_delay
- set_clock_uncertainty
- set_false_path
- set_input_delay
- set_max_delay
- set_multicycle_path
- set_output_delay
- set_reg_input_delay
- set_reg_output_delay

Copyright © 2013 Synopsys, Inc.

For information on the supported design constraints, see Chapter 15, *FPGA Design Constraint Syntax*.

Copyright © 2013 Synopsys, Inc.

create_clock

Creates a clock object and defines its waveform in the current design.

Syntax

The supported syntax for the create_clock constraint is:

```
create_clock
    -name clockName [-add] {objectList} |
        -name clockName [-add] [{objectList}] |
        [-name clockName [-add]] {objectList}
    -period value
    [-waveform {riseValue fallValue}]
    [-disable]
    [-comment commentString]
```

Arguments

-name *clockName*	Specifies the name for the clock being created, enclosed in quotation marks or curly braces. If this option is not used, the clock gets the name of the first clock source specified in the *objectList* option. If you do not specify the *objectList* option, you must use the -name option, which creates a virtual clock not associated with a port, pin, or net. You can use both the -name and *objectList* options to give the clock a more descriptive name than the first source pin, port, or net. If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you specify this option, you must also use the -name option.
-period *value*	Specifies the clock period in nanoseconds. This is the minimum time over which the clock waveform repeats. The *value* type must be greater than zero.

Copyright © 2013 Synopsys, Inc.

-waveform *riseValue fallValue*	Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise followed by fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of *periodValue*/2.
objectList	Clocks can be defined on the following objects: pins, ports, and nets The FPGA synthesis tools support nets and instances, where instances have only one output (for example, BUFGs).
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

create_generated_clock

Creates a generated clock object.

Syntax

The supported syntax for the create_generated_clock constraint is:

create_generated_clock
-name *clockName* [**-add**]] | **{***clockObject***}**
-source *masterPinName*
[**-master_clock** *clockName*]
[**-divide_by** *integer* | **-multiply_by** *integer* [**-duty_cycle** *value*]]
[**-invert**]
[**-edges {***edgeList***}**]
[**-edge_shift {***edgeShiftList***}**]
[**-combinational**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-name *clockName*	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the -source option (*clockObject*). If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also use the -name and -master_clock options.
clockObject	The first clock source specified in the -source option in the absence of *clockName*. Clocks can be defined on pins, ports, and nets. The FPGA synthesis tools support nets and instances, where instances have only one output (for example, BUFGs).
-source *masterPinName*	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.

Copyright © 2013 Synopsys, Inc.

-master_clock *clockName*	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.
-divide_by *integer*	Specifies the frequency division factor. If the *divideFactor* value is *2*, the generated clock period is twice as long as the master clock period.
-multiply_by *integer*	Specifies the frequency multiplication factor. If the *multiplyFactor* value is *3*, the generated clock period is one-third as long as the master clock period.
-duty_cycle *percent*	Specifies the duty cycle, as a percentage, if frequency multiplication is used. Duty cycle is the high pulse width.
-invert	Inverts the generated clock signal (in the case of frequency multiplication and division).
-edges *edgeList*	Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.
-edge_shift *edgeShiftList*	Specifies a list of floating point numbers that represents the amount of shift, in nanoseconds, that the specified edges are to undergo to yield the final generated clock waveform. The number of edge shifts specified must be equal to the number of edges specified. The values can be positive or negative; positive indicating a shift later in time, while negative indicates a shift earlier in time. For example, 1 indicates that the corresponding edge is to be shifted by one library time unit.
-combinational	The source latency paths for this type of generated clock only includes the logic where the master clock propagates. The source latency paths do not flow through sequential element clock pins, transparent latch data pins, or source pins of other generated clocks.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

reset_path

Resets the specified paths to single-cycle timing.

Syntax

The supported syntax for the reset_path constraint is:

reset_path [**-setup**]
[**-from {***objectList***}**]
[-**through {***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-setup	Specifies that setup checking (maximum delay) is reset to single-cycle behavior.
-from	Specifies the names of objects to use to find path start points. The -from *objectList* includes: • Clocks • Registers • Top-level input or bi-directional ports) • Black box outputs When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points

Copyright © 2013 Synopsys, Inc.

-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, reset_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, reset_path applies only if the -from or -to and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during timing analysis.

Syntax

The supported syntax for the set_clock_groups constraint is:

set_clock_groups
-asynchronous
-group {*clockList***}** [**-group {***clockList***}** …]
-derive
[**-disable**]
[**-comment** *commentString*]

Arguments

-asynchronous	Specifies that the clock groups are asynchronous to each other (the FPGA synthesis tools assume all clock groups are synchronous). Two clocks are asynchronous with respect to each other if they have no phase relationship at all.
-group *clockList*	Specifies the clocks in a group (*clockList*). You can use the -group option more than once in a single command execution. However, you can include a clock only in one group in a single command execution. To include a clock in multiple groups, you must execute the set_clock_groups command multiple times. By default, a generated clock and its master clock are not in the same group when the exclusive or asynchronous clock groups are defined. The -derive option can be used to override this behavior and allow generated or derived clocks to inherit the clock group of their parent source clock. Each -group instance specifies a group of clocks, which are exclusive or asynchronous with the clocks in all other groups. If you specify only one group, it means that the clocks in that group are exclusive or asynchronous with all other clocks in the design. A default other group is created for this single group. Whenever a new clock is created, it is automatically included in this group.
-derive	Specifies that generated and derived clocks inherit the clock group of the parent clock.

Copyright © 2013 Synopsys, Inc.

-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Limitations

Clock grouping in the FPGA synthesis environment is inclusionary or exclusionary (for example, clk2 and clk3 can each be related to clk1 without being related to each other).

The following set_clock_groups constraint specifies that clk1 and clk2 are asynchronous, but clk1 and clk2 are synchronous to clk3.

```
create_clock [get_ports {c1}] -name clk1 -period 10
create_clock [get_ports {c2}] -name clk2 -period 16
create_clock [get_ports {c3}] -name clk3 -period 5
set_clock_groups -asynchronous -group [get_clocks {clk1}]
   -group [get_clocks {clk2}]
```

This set_clock_groups constraint specifies that clock clk1, clk2, and clk3 are synchronous to one another, but asynchronous to all other clocks in the design.

```
set_clock_groups -asynchronous -group {clk1 clk2 clk3}
```

This set_clock_groups constraint specifies that clk1 and clk2 are synchronous to one another, but are asynchronous to clk3 and clk4. However, clk3 and clk4 are synchronous to each other.

```
set_clock_groups -asynchronous -group {clk1 clk2}
   -group {clk3 clk4}
```

This set_clock_groups constraint specifies that clk4 is asynchronous to all other clocks in the design.

```
set_clock_groups -asynchronous -group {clk4}
```


Copyright © 2013 Synopsys, Inc.

Note: The -group option contains a list of clocks that are separated by a space. Do not add a comma after the clock name.

For example, if you specify the following:

```
set_clock_groups -asynchronous -group {clk1, clk2}
```

The tool generates a warning that clk1 cannot be found, because the comma is treated as part of the clock name. This is true for all constraints that contain lists.

Copyright © 2013 Synopsys, Inc.

set_clock_latency

Specifies clock network latency.

Syntax

The supported syntax for the set_clock_latency constraint is:

set_clock_latency
-source
[**-clock {***clockList***}**]
delayValue
{*objectList***}**
[**-disable**]

Arguments

-source	Indicates that the specified delay is applied to the clock source latency.
-clock *clockList*	Indicates that the specified delay is applied with respect to the specified clocks. By default, the specified delay is applied to all specified objects.
delayValue	Specifies the clock latency value.
objectList	Specifies the input ports for which clock latency is to be set

Description

In the FPGA synthesis tools, the set_clock_latency constraint accepts both clock objects and clock aliases. Applying a set_clock_latency constraint on a port can be used to model the off-chip clock delays in a multi-chip environment. Clock latency is forward annotated in the top-level constraint file as part of the time budgeting that takes place in the Certify/HAPS flow. The annotated values represent the arrival times for clocks on specific ports of any particular FPGA in a HAPS design.

In the above syntax, *objectList* references either input ports with defined clocks or clock aliases defined on the input ports. When more than one clock is defined for an input port, the -clock option can be used to apply different latency values to each alias.

Copyright © 2013 Synopsys, Inc.

Restrictions

The following limitations are present in the FPGA synthesis environment:

- Clock latency can only be applied to clocks defined on input ports.
- The set_clock_latency constraint is only used for source latency.
- The constraint only applies to port clock objects.
- Latency on clocks defined with create_generated_clock is not supported.

Copyright © 2013 Synopsys, Inc.

set_clock_route_delay

Translates the -route option for the legacy define_clock constraint.

Syntax

The supported syntax for the set_clock_route_delay constraint is:

set_clock_route_delay {*clockAliasList***} {***delayValue***}**

Arguments

clockAliasList	Lists the clock aliases to include the route delay.
delayValue	Specifies the route delay value.

Description

The sdc2fdc translator performs a translation of the -route option for the legacy define_clock constraint and places a set_clock_route_delay constraint in the *_translated.fdc file using the following format:

```
set_clock_route_delay [get_clocks {clk_alias_1 clk_alias_2 ...}]
   {delay_in_ns}
```


Copyright © 2013 Synopsys, Inc.

set_clock_uncertainty

Specifies the uncertainty (skew) of the specified clock networks.

Syntax

The supported syntax for the set_clock_uncertainty constraint is:

set_clock_uncertainty
{*objectList***}**
-from *fromClock* |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*
-to *toClock* |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*
value

Arguments

objectList	Specifies the clocks for simple uncertainty. The uncertainty is applied to the capturing latches clocked by one of the specified clocks. You must specify either this argument or a clock pair with the -from/-rise_from/-fall_from and -to/-rise_to/-fall_to options; you cannot specify both an object list and a clock pair.
-from *fromClock*	Specifies the source clocks for interclock uncertainty. You can use only one of the -from, -rise_from, and -fall_from options and you must specify a destination clock with one of the -to, -rise_to, and -fall_to options.
-rise_from *riseFromClock*	Specifies that the uncertainty applies only to the rising edge of the source clock. You can use only one of the -from, -rise_from, and -fall_from options and you must specify a destination clock with one of the -to, -rise_to, and -fall_to options.
-fall_from *fallFromClock*	Specifies that the uncertainty applies only to the falling edge of the source clock. You can use only one of the -from, -rise_from, and -fall_from options and you must specify a destination clock with one of the -to, -rise_to, and -fall_to options.
-to *toClock*	Specifies the destination clocks for interclock uncertainty. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.
-rise_to *riseToClock*	Specifies that the uncertainty applies only to the rising edge of the destination clock. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.

Copyright © 2013 Synopsys, Inc.

-fall_to *fallToClock*	Specifies that the uncertainty applies only to the falling edge of the destination clock. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.
value	Specifies a floating-point number that indicates the uncertainty value. Typically, clock uncertainty should be positive. Negative uncertainty values are supported for constraining designs with complex clock relationships. Setting the uncertainty value to a negative number could lead to optimistic timing analysis and should be used with extreme care.

Copyright © 2013 Synopsys, Inc.

set_datapathonly_delay

Specifies a point-to-point path delay that excludes any applicable I/O delays or clock delays at the start and end points.

Syntax

set_datapathonly_delay *option* **{***value***}**

In the above syntax, *option* is one or more of the following:

-disable
-from {*port***|***inst***|***clock***}**
-rise_from *clock*
-fall_from *clock*
-to {*port***|***inst***|***clock***}**
-rise_to *clock*
-fall_to *clock*
-comment *textsString*

In the above options, the variable *clock* is a clock alias.

Description

The set_datapathonly_delay constraint specifies a point-to-point path delay that excludes any applicable I/O delays or clock delays at the start and end points. The constraint includes clock to out (Tco) and setup time (Tsu) as part of the data path delay.

Use this constraint in Xilinx designs, as the tool only forward-annotates the constraint for Xilinx designs. The set_datapathonly_delay constraint appears in the UCF file as a Xilinx TIMESPEC constraint with the DATAPATHONLY option. The original `sdc` command is also forward-annotated as a comment. For example:

```
# set_datapathonly_delay -from {p:din[*]} -to {i:int[*]} {4}
...
NET "din[7]" TNM = "from_1005_0";
INST "int[*]" TNM = "to_1005_0"; # int[7:0]
TIMESPEC "TS_1005_0" = FROM "from_1005_0" TO "to_1005_0"
   DATAPATHONLY 4.000;
```

Copyright © 2013 Synopsys, Inc.

When you specify this constraint, the tool ignores any clock skew or phase information. It constrains, analyzes, and forward-annotates just the data path between the specified points.

Copyright © 2013 Synopsys, Inc.

set_false_path

Removes timing constraints from particular paths.

Syntax

The supported syntax for the set_false_path constraint is:

set_false_path
[**-setup**]
[**-from {***objectList***}**]
[-**through {***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-setup	Specifies that setup checking (maximum delay) is reset to single-cycle behavior.
-from	Specifies the names of objects to use to find path start points. The -from *objectList* includes: • Clocks • Registers • Top-level input or bi-directional ports • Black box outputs When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points.

Copyright © 2013 Synopsys, Inc.

-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, set_false_path applies only if the -from or -to and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

set_input_delay

Sets input delay on pins or input ports relative to a clock signal.

Syntax

The supported syntax for the set_input_delay constraint is:

set_input_delay
[**-clock** *clockName* [**-clock_fall**]]
[**-rise**|**-fall**]
[**-min**|**-max**]
[**-add_delay**]
delayValue
{*portPinList***}**
[**-disable**]
[**-comment** *commentString*]

Argument

-clock *clockName*	Specifies the clock to which the specified delay is related. If -clock_fall is used, -clock *clockName* must be specified. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.
-clock_fall	Specifies that the delay is relative to the falling edge of the clock. The default is the rising edge.
-rise	Specifies that *delayValue* refers to a rising transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.

Copyright © 2013 Synopsys, Inc.

-fall	Specifies that *delayValue* refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
-min	Specifies that *delayValue* refers to the shortest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal. **Note:** The synthesis tool does not optimize for hold time violations and only reports -min delay values in the `synlog/topLevel_fpga_mapper.srr_Min` timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.
-max	Specifies that *delayValue* refers to the longest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal. **Note:** The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
-add_delay	Specifies if delay information is to be added to the existing input delay or if is to be overwritten. The -add_delay option enables you to capture information about multiple paths leading to an input port that are relative to different clocks or clock edges.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
delayValue	Specifies the path delay. The *delayValue* must be in units consistent with the technology library used during optimization. The *delayValue* represents the amount of time the signal is available after a clock edge. This represents a combinational path delay from the clock pin of a register.
portPinList	Specifies a list of input port names in the current design to which *delayValue* is assigned. If more than one object is specified, the objects are enclosed in quotes ("") or in braces ({}).

Copyright © 2013 Synopsys, Inc.

set_max_delay

Specifies a maximum delay target for paths in the current design.

Syntax

The supported syntax for the set_max_delay constraint is:

set_max_delay
[**-from {***objectList***}**]
[-**through{***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
delayValue
[**-disable**]
[**-comment** *commentString*]

Arguments

-from

Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bi-directional ports
- Black box outputs

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. All paths from these start points to the end points in the -from *objectList* are constrained to *delayValue*. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({}).

-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. The max delay value applies only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_max_delay applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, set_max_delay applies only if the -from or -to and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. All paths to the end points in the -to *objectList* are constrained to *delayValue*. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({}).
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

delayValue	Specifies the value of the desired maximum delay for paths between start and end points. You must express *delayValue* in the same units as the technology library used during optimization. If a path start point is on a sequential device, clock skew is included in the computed delay. If a path start point has an input delay specified, that delay value is added to the path delay. If a path end point is on a sequential device, clock skew and library setup time are included in the computed delay. If the end point has an output delay specified, that delay is added into the path delay.

Copyright © 2013 Synopsys, Inc.

set_multicycle_path

Modifies the single-cycle timing relationship of a constrained path.

Syntax

The supported syntax for the set_multicycle_path constraint is:

set_multicycle_path
[**-start** |**-end**]
[**-from {***objectList***}**]
[-**through {***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
pathMultiplier
[**-disable**]
[**-comment** *commentString*]

Arguments

-start | **-end**

Specifies if the multi-cycle information is relative to the period of either the start clock or the end clock. These options are only needed for multi-frequency designs; otherwise start and end are equivalent. The start clock is the clock source related to the register or primary input at the path start point. The end clock is the clock source related to the register or primary output at the path endpoint. The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with -end moves the relation forward one cycle of the end clock. A setup multiplier of 2 with -start moves the relation back one cycle of the start clock. A hold multiplier of 1 with -start moves the relation forward one cycle of the start clock. A hold multiplier of 1 with -end moves the relation back one cycle of the end clock.

Copyright © 2013 Synopsys, Inc.

-from	Specifies the names of objects to use to find path start points. The -from *objectList* includes: • Clocks • Registers • Top-level input or bi-directional ports • Black box outputs When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({}).
-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells The multi-cycle values apply only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, the objects must be enclosed either in double quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_multicycle_delay applies to the paths that pass through a member of each *objectList*. If the -through option is used in combination with the -from or -to options, the multi-cycle values apply only if the -from or -to conditions and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({})..
-disable	Disables the constraint.

Copyright © 2013 Synopsys, Inc.

-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
pathMultiplier	Specifies the number of cycles that the data path must have for setup or hold relative to the start point or end point clock before data is required at the end point. When used with -setup, this value is applied to setup path calculations. When used with -hold, this value is applied to hold path calculations. If neither -hold nor -setup are specified, *pathMultiplier* is used for setup, and 0 is used for hold. Changing the *pathMultiplier* for setup also affects the hold check.

Copyright © 2013 Synopsys, Inc.

set_output_delay

Sets output delay on pins or output ports relative to a clock signal.

Syntax

The supported syntax for the set_output_delay constraint is:

set_output_delay
[**-clock** *clockName* [**-clock_fall**]]
[**-rise**|[**-fall**]
[**-min**|**-max**]
[**-add_delay**]
delayValue
{*portPinList***}**
[**-disable**]
[**-comment** *commentString*]

Arguments

-clock *clockName*	Specifies the clock to which the specified delay is related. If -clock_fall is used, -clock *clockName* must be specified. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.
-clock_fall	Specifies that the delay is relative to the falling edge of the clock. If -clock is specified, the default is the rising edge.
-rise	Specifies that *delayValue* refers to a rising transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.

Copyright © 2013 Synopsys, Inc.

-fall	Specifies that *delayValue* refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
-min	Specifies that *delayValue* refers to the shortest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal. **Note:** The synthesis tool does not optimize for hold time violations and only reports -min delay values in the `synlog/topLevel_fpga_mapper.srr_Min` timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.
-max	Specifies that *delayValue* refers to the longest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal. **Note:** The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
-add_delay	Specifies whether to add delay information to the existing output delay or to overwrite. The -add_delay option enables you to capture information about multiple paths leading to an output port that are relative to different clocks or clock edges.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
delayValue	Specifies the path delay. The *delayValue* must be in units consistent with the technology library used during optimization. The *delayValue* represents the amount of time that the signal is required before a clock edge. For maximum output delay, this usually represents a combinational path delay to a register plus the library setup time of that register. For minimum output delay, this value is usually the shortest path delay to a register minus the library hold time

Copyright © 2013 Synopsys, Inc.

portPinList	A list of output port names in the current design to which *delayValue* is assigned. If more than one object is specified, the objects are enclosed in double quotation marks ("") or in braces ({}).

Copyright © 2013 Synopsys, Inc.

set_reg_input_delay

Speeds up paths feeding a register by a given number of nanoseconds.

Syntax

set_reg_input_delay {*registerName***}** [**-route** *ns*] [**-disable**] [**-comment** *textString*]

Arguments

registerName	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The set_reg_input_delay timing constraint speeds up paths feeding a register by a given number of nanoseconds. The Synopsys FPGA synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with create_clock). Use this constraint to speed up the paths feeding a register. For information about the equivalent SCOPE spreadsheet interface, see *Registers*, on page 484.

Use this constraint instead of the legacy constraint, define_reg_input_delay.

Copyright © 2013 Synopsys, Inc.

set_reg_output_delay

Speeds up paths coming from a register by a given number of nanoseconds.

Syntax

set_reg_output_delay {*registerName***}** [**-route** *ns*] [**-disable**] [**-comment** *textString*]

Arguments

registerName	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths from the register.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The set_reg_output_delay constraint speeds up paths coming from a register by a given number of nanoseconds. The synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with create_clock). Use this constraint to speed up the paths coming from a register. For information about the equivalent SCOPE spreadsheet interface, see *Registers*, on page 484.

Use this constraint instead of the legacy constraint, define_reg_output_delay.

Naming Rule Syntax Commands

The FPGA synthesis environment uses a set of naming conventions for design objects in the RTL when your project contains constraint files. The following naming rule commands are added to the constraint file to change the expected default values. These commands must appear at the beginning of

the constraint file before any other constraints. Similarly, when multiple constraint files are included in the project, the naming rule commands must be in the first constraint file read.

set_hierarchy_separator Command

The set_hierarchy_separator command redefines the hierarchy separator character (the default separator character is the period in the FPGA synthesis environment). For example, the following command changes the separator character to a forward slash:

```
set_hierarchy_separator {/}
```

set_rtl_ff_names Command

The set_rtl_ff_names command controls the stripping of register suffixes in the object strings of delay-path constraints (for example, set_false_path, set_multicycle_path). Generally, it is only necessary to change this value from its default when constraints that target ASIC designs are being imported from the Design Compiler (in the Design Compiler, inferred registers are given a _reg suffix during the elaboration phase; constraints targeting these registers must include this suffix). When importing constraints from the Design Compiler, include the following command to change the value of this naming rule to {_reg} to automatically recognize the added suffix.

```
set_rtl_ff_names {_reg}
```

For example, using the above value allows the DC exception

```
set_false_path -to [get_cells {register_bus_reg[0]}]
```

to apply to the following object without having to manually modify the constraint:

```
[get_cells {register_bus[0]}]
```

bus_naming_style Command

The bus_naming_style command redefines the format for identifying bits of a bus (by default, individual bits of a bus are identified by the bus name followed by the bus bit enclosed in square brackets). For example, the following command changes the bus-bit identification from the default *busName*[*busBit*] format to the *busName_busBit* format:

```
bus_naming_style {%s_%d}
```


Copyright © 2013 Synopsys, Inc.

bus_dimension_separator_style Command

The bus_dimension_separator_style command redefines the format for identifying multi-dimensional arrays (by default, multidimensional arrays such as row 2, bit 3 of array ABC[*n* x *m*] are identified as ABC[2][3]). For example, the following command changes the bus-dimension separator from individual square bracket sets to an underscore:

```
bus_dimension_separator_style {_}
```

The resulting format for the above example is:

```
ABC[2_3]
```

read_sdc Command

Reads in a script in Synopsys FPGA constraint format. The supported syntax for the read_sdc constraint is:

read_sdc *fileName*

Synplify-Style Timing Constraints (Legacy)

This section describes the constraint file syntax for the Synplify-style legacy timing constraints. For tool versions 2012.09 and later, it is recommended that you use the Synopsys FDC format instead of the Synplify-style legacy format. Use the sdc2fdc command to convert your constraints. See *FPGA Timing Constraints*, on page 1100 for more information.

The legacy constraints are included in your project through an SDC file. This section describes the constraint file syntax for the legacy FPGA timing constraints, which include:

- define_clock, on page 1136
- define_clock_delay, on page 1139
- define_false_path, on page 1140
- define_input_delay, on page 1143
- define_multicycle_path, on page 1145
- define_output_delay, on page 1149
- define_path_delay, on page 1151
- define_reg_input_delay, on page 1154
- define_reg_output_delay, on page 1155
- set_datapathonly_delay, on page 1156

For information on the supported design constraints, see Chapter 15, *FPGA Design Constraint Syntax*.

Copyright © 2013 Synopsys, Inc.

define_clock

Defines a clock with a specific duty cycle and frequency or clock period goal.

Syntax

define_clock [**-disable**] [**-virtual**] {*clockObject*}
[**-freq** *MHz* |**-period** *ns*] [**-clockgroup** *domain*]
[**-rise** *value* |**-fall** *value*] [**-route** *ns*]
[**-name** *clockName*] [**-comment** *textString*]

Arguments

-disable	Disables a previous clock constraint.
-virtual	Specifies arrival and required times on top level ports that are enabled by clocks external to the chip (or block) that you are synthesizing. When specifying -name for the virtual clock, the field can contain a unique name not associated with any port or instance in the design.
clockObject	A required parameter that specifies the clock object name. Clocks can be defined on the following objects: • Top-level input ports (p:) • Nets (n:) • Instances (i:) For Xilinx technologies, see *Forward-annotation of Port Names in EDIF*, on page 2117 for details about specifying the define_clock constraint on an instance. • Output pins of instantiated cells (t:) Clocks defined on any of the following objects *WILL NOT* be honored: • Top-level output ports • Input pins of instantiated gates • Pins of inferred instances
-name	Specifies a name for the clock if you want to use a name other than the clock object name. This alias name is used in the timing reports.
-freq	Defines the frequency of the clock in MHz. You can specify either this or -period, but not both.
-period	Specifies the period of the clock in ns. Specify either this or -freq, but not both.

-clockgroup	Allows you to specify clock relationships. You assign related (synchronized) clocks to the same clock group and unrelated clocks in different groups. The synthesis tool calculates the relationship between clocks in the same clock group, and analyzes all paths between them. Paths between clocks in different groups are ignored (false paths). See *Clock Groups*, on page 522 for more information.
-rise|-fall	Specifies a non-default duty cycle. By default, the synthesis tool assumes that the clock is a 50% duty cycle clock, with the rising edge at 0 and the falling edge at period/2. If you have another duty clock cycle, specify the appropriate Rise At and Fall At values. For more information, see *Rise and Fall Constraints*, on page 523.
-route	An advanced user option that improves the path delays of all registers controlled by this clock. The value is the difference between the synthesis timing report path delays and the value in the place-and-route timing report. The -route constraint applies globally to the clock domain, and can over constrain registers where constraints are not needed. For details, see *Route Option*, on page 524. Before you use this option, evaluate the path delays on individual registers in the optimization timing report and try to improve the delays by applying the constraints define_reg_input_delay and define_reg_output_delay only on the registers that need them.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Description

The define_clock timing constraint defines a clock with a specific duty cycle and frequency or clock period goal. You can have multiple clocks with different clock frequencies. Set the default frequency for all clocks with the set_option -frequency Tcl command in the project file. If you do not specify a global frequency, the timing analyzer uses a default. Use the define_clock timing constraint to override the default and specify unique clock frequency goals for specific clock signals. Additionally, you can use the define_clock timing constraint to set the clock frequency for a clock signal output of clock divider logic. The clock name is the output signal name for the register instance. If you are constraining a differential clock, you only need to constrain the positive input.

Copyright © 2013 Synopsys, Inc.

For the equivalent SCOPE spreadsheet interface and descriptions of the options, see *Clocks (Legacy)*, on page 519 and *Clocks Panel Description*, on page 520.

Examples

define_clock Syntax

```
define_clock {CLK1} -period 10.0 -clockgroup default_clkgroup

define_clock {CLK3} -period 5.0 -clockgroup default_clkgroup
   -name INT_REF3

define_clock -virtual {CLK2} -period 20.0 -clockgroup g2

define_clock {CLK4} -period 20.000 -clockgroup g3
   -rise 1.000 -fall 11.000 -ref_rise 0.000 -ref_fall 10.000
```

define_clock Pin-Level Constraint

```
define_clock {i:myInst1.Q} -period 10.000
   -clockgroup default -rise 0.200 –fall 5.200 -name myff1

define_clock {i:myInst2.Q} -period 12.000
   -clockgroup default -rise 0.400 –fall 5.400 -name myff2
```

In the example above, a clock is defined on the Q pins of instances myInst1 and myInst2.

define_clock_delay

Defines the delay between the clocks in the design.

Syntax

define_clock_delay [**-disable**]
-rise|-fall {*clockName1***} -rise|-fall {***clockName2***}** *delayValue*

Arguments

-rise|fall	**Specifies the clock edge**
clockName	Specifies the clocks to constrain The clock must be already defined with define_clock.
delayValue	Specifies the delay, in nanoseconds, between the two clocks. You can also specify a value false which defines the path as a false path.
-disable	Disables the constraint.

Description

The define_clock_delay timing constraint defines the delay between the clocks in the design. By default, the software automatically calculates clock delay based on the clock parameters you define through the define_clock command. However, if you use define_clock_delay, the specified delay value overrides any calculations made by the software. The results shown in the Clock Relationships section of the Timing Report are based on calculations made using this constraint.

Note: Maintenance for the define_clock_delay command is limited. It is recommended that you use the define_false_path and define_path_delay commands instead, since these commands support clock aliases.

Copyright © 2013 Synopsys, Inc.

define_false_path

Defines paths to ignore (remove) during timing analysis.

Syntax

define_false_path [**-disable**] {**-from** *startPoint* |**-to** *endPoint* |**-through** *throughPoint*} [**-comment** *textString*]

Arguments

-from *startPoint*	Specifies the starting point for the false path. The from point defines a timing start point and can be any of the following: • Clocks (c:) (See *Clocks as From/To Points*, on page 511.) • Registers (i:) • Top-level input or bi-directional ports (p:) • Black box outputs (i:) For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*.
-to *endPoint*	Specifies the ending point for the false path. The to point defines a timing end point and can be any of the following objects: • Clocks (c:) (See *Clocks as From/To Points*, on page 511.) • Registers (i:) • Top-level output or bi-directional ports (p:) • Black box inputs (i:) For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*

Copyright © 2013 Synopsys, Inc.

-through *throughPoint*	Specifies the intermediate points for the timing exception. Intermediate points can be any of the following objects: • Combinational nets (n:) • Hierarchical ports (t:) • Pins on instantiated cells (t:) By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in the list. For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide* To keep the signal name intact through synthesis, set the syn_keep directive (Verilog or VHDL) on the signal.
-disable	Disables the constraint.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Description

The define_false_path timing constraint defines paths to ignore (remove) during timing analysis and gives lower (or no) priority during optimization. The false paths are also passed on to supported place-and-route tools. For information about the equivalent SCOPE spreadsheet interface and the options, see *False Paths*, on page 503. See *False Path Constraint Examples*, on page 504, and *Priority of False Path Constraints*, on page 504, for additional information.

Example

The following example shows the syntax for setting a define_false_path constraint between registers:

```
define_false_path -from {i:myInst1_reg} -through {n:myInst2_net}
   -to {i:myInst3_reg}
```


Copyright © 2013 Synopsys, Inc.

The constraint is defined from the output pin of myInst1_reg, through net myInst2_net, to the input of myInst3_reg. If an instance is instantiated, a pin-level constraint applies on the pin, as defined. However, if an instance is inferred, the pin-level constraint is transferred to the instance.

For through points specified on pins, the constraint is transferred to the connecting net. You cannot define a through point on a pin of an instance that has multiple outputs. When specifying a pin on a vector of instances, you cannot refer to more than one bit of that vector.

Copyright © 2013 Synopsys, Inc.

define_input_delay

Specifies the external input delays on top-level ports in the design.

Syntax

define_input_delay [**-disable**] {*inputportName* | **-default** *ns* [**-route** *ns*] [**-ref** *clockName*:*edge*] [**-comment** *textString*]

Arguments

-disable	Disables the constraint.
inputportName	The name of the input port.
-default	Sets a default input delay for all inputs. Use this option to set an input delay for all inputs. You can then set define_input_delay on individual inputs to override the default constraint. This example sets a default input delay of 3.0 ns: define_input_delay -default 3.0 This constraint overrides the default and sets the delay on input_a to 10.0 ns: define_input_delay {input_a} 10.0
-route	An advanced option, which includes route delay when the synthesis tool tries to meet the clock frequency goal. Use the -route option on an input port when the place-and-route timing report shows that the timing goal is not met because of long paths through the input port. See *Route Option*, on page 529 for an example of its use.
-ref	(Recommended.) Clock name and edge that triggers the event. The value must include either the rising edge or falling edge. **r** - rising edge **f** - falling edge For example: define_input_delay {portb[7:0]} 10.00 -ref clock2:f
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

Description

The define_input_delay timing constraint specifies the external input delays on top-level ports in the design. This external delay is the delay outside the chip before the signal arrives at the input pin. This constraint is used to model the interface of the inputs of the FPGA with the outside environment. The tool has no way of knowing what the input delay is unless you specify it in a timing constraint. For information about the equivalent SCOPE interface, see *Inputs/Outputs (Legacy)*, on page 527.

Example

Here are some examples of the define_input_delay constraint:

```
define_input_delay {porta[7:0]} 7.8 -ref clk1:r
define_input_delay -default 8.0
define_input_delay -disable {resetn}
```

The following example shows how all the bits of the datain input bus are constrained with a delay of 2 ns, except for bit 16:

```
define_input_delay {datain[0:15]} 2.00 -ref clk:r
define_input_delay {datain[17:99]} 2.00 -ref clk:r
```

Copyright © 2013 Synopsys, Inc.

define_multicycle_path

Specifies a path that uses multiple clock cycles as a timing exception.

Syntax

define_multicycle_path [**-disable**] [**-start** |**-end**]
{**-from** *startPoint* |**-to** *endPoint* |**-through** *throughPoint*}
clockCycles [**-comment** *textString*]

Arguments

-start | **-end**	Specifies the clock cycles to use for paths with different start and end clocks. This option determines the clock period to use as the multiplicand in the calculation for clock distance. If you do not specify a start or end option, the end clock is the default. See *Multi-cycle Path with Different Start and End Clocks*, on page 500 for more information.
-from *startPoint*	Specifies the start point for the multi-cycle timing exception. The From point in the constraint defines a timing start point and can be any of the following: • Clocks (c:) (See *Clocks as From/To Points*, on page 511.) • Registers (i:) • Top-level input or bi-directional ports (p:) • Black box outputs (i:) For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

-to *endPoint*	Specifies the end point for the multi-cycle timing exception. The to point defines a timing end point and can be any of the following objects: • Clocks (c:) (See Clocks as From/To Points, on page 511.) • Registers (i:) • Top-level output or bi-directional ports (p:) • Black box outputs (i:) For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*.
-through *throughPoint*	Specifies the intermediate points for the timing exception. Intermediate points can be: • Combinational nets (n:) • Hierarchical ports (t:) • Pins on instantiated cells (t:) By default, the intermediate points are treated as an OR list, the exception is applied if the path crosses any points in the list. For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*. You can combine this option with -to or -from to get a specific path. To keep the signal name intact throughout synthesis when you use this option, set the syn_keep directive (Verilog or VHDL) on the signal.
clockCycles	The number of clock cycles to use for the path constraint.
-disable	Disables the constraint.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Copyright © 2013 Synopsys, Inc.

Description

The define_multicycle_path timing constraint specifies a path that uses multiple clock cycles as a timing exception. This constraint provides extra clock cycles to the designated paths for timing analysis and optimization. For information about the equivalent SCOPE interface, see *Multicycle Paths*, on page 500.

Timing Exceptions Object Types

Timing exception constraints must contain object types in the specification. Timing exceptions, such as multi-cycle path and false path constraints, require that you explicitly specify the object type (n: or i:) in the instance name parameter. For example:

```
define_multicycle_path -from {i:inst2.lowreg_output[7]}
   -to {i:inst1.DATA0[7]} 2
```

If you use the SCOPE UI to specify timing exceptions, it automatically attaches object type qualifiers to the object names. For more information, see *Specifying From, To, and Through Points*, on page 506.

Example

Here are some examples of the define_multicycle_path constraint syntax:

```
define_multicycle_path -from{i:regs.addr[4:0]}
   -to {i:special_regs.w[7:0]} 2

define_multicycle_path -to {i:special_regs.inst[11:0]} 2

define_multicycle_path -from {p:porta[7:0]}
   -through {n:prgmcntr.pc_sel44[0]} -to {p:portc[7:0]} 2

define_multicycle_path -from {i:special_regs.trisc[7:0]}
   -through {t:uc_alu.aluz.Q} -through {t:special_net.Q} 2
```

The following example shows the syntax for setting a multi-cycle path constraint between registers:

```
define_multicycle_path -from {i:myInst1_reg}
   -through {n:myInst2_net} -to {i:myInst3_reg} 2
```

The constraint is defined from the output of myInst1_reg, through net myInst2_net, to the input pin myInst3_reg. If the instance is instantiated, a pin-level constraint applies on the pin, as defined. However, if the instance is inferred, the pin-level constraint is transferred to the instance. For through

Copyright © 2013 Synopsys, Inc.

points specified on pins, the constraint is transferred to the connecting net. You cannot define a through point on a pin of an instance that has multiple outputs. When specifying a pin on a vector of instances, you cannot refer to more than one bit of that vector.

For additional examples of multi-cycle paths, see *Multicycle Path Examples*, on page 501.

define_output_delay

Specifies the delay of the logic outside the FPGA driven by the top-level outputs.

Syntax

define_output_delay [**-disable**] **{***outputportName***}** |**-default** *ns* [**-route** *ns*] [**-ref** *clockName*:*edge*] [**-comment** *textString*]

Arguments

-disable	Disables the constraint.
outputportName	The name of the output port.
-default	Sets a default output delay for all outputs. Use this option to set a delay for all outputs. You can then set define_output_delay on individual outputs to override the default constraint. This example sets a default output delay of 8.0 ns. The delay is outside the FPGA. define_output_delay -default 8.0 The following constraint overrides the default and sets the output delay on output_a to 10.0 ns. This means that output_a drives 10 ns of combinational logic before the relevant clock edge. define_output_delay {output_a} 10.0

Copyright © 2013 Synopsys, Inc.

-route	An advanced option, which includes route delay when the synthesis tool tries to meet the clock frequency goal. See *Route Option*, on page 529 for an example of its use with an input register.
-ref	Defines the clock name and edge that controls the event. Value must be one of the following: **r** - rising edge **f** - falling edge For example: define_output_delay {portb[7:0]} 10.00 -ref clock2:f See *Output Pad Clock Domain Default*, below for more information on this option.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Description

The define_output_delay constraint models the interface of the outputs of the FPGA with the outside environment. The default delay outside the FPGA is 0.0 ns. Output signals typically drive logic that exists outside the FPGA, but the tool has no way of knowing the delay for that logic unless you specify it using a timing constraint. For information about the equivalent SCOPE spreadsheet interface, see *Inputs/Outputs (Legacy)*, on page 527.

Output Pad Clock Domain Default

By default, define_output_delay constraints with no reference clock are constrained against the global frequency, instead of the start clock for the path to the port. The tool assumes the register and pad are not in the same clock domain. This change affects the timing report and timing driven optimizations on any logic between the register and the pad.

You must specify the clock domain for all output pads on which you have set output delay constraints. For the pads for which you do not specify a clock, add the -ref option to the define_output_delay constraint. For example:

```
define_output_delay {LDCOMP} 0.50
   -route 0.25 -ref {CLK1:r}
```

define_path_delay

Specifies point-to-point delay for maximum delay constraints.

Syntax

define_path_delay [**-disable**]
-from {*startPoint*} | **-to** {*endPoint*} | **-through** {*throughPoint*}
-max *delayValue* [**-comment** *textString*]

Arguments

-disable	Disables the constraint.
-from *startPoint*	Specifies the starting point of the path. The From point defines a timing start point and can be any of the following objects: • Clocks (c:) (See *Clocks as From/To Points*, on page 511.) • Registers (i:) • Top-level input or bi-directional ports (p:) • Black box outputs (i:) For more information, see the following: • syn_black_box, on page 1235 • Specifying From, To, and Through Points, on page 506 • Defining From/To/Through Points for Timing Exceptions, on page 146 in the *User Guide*.
-to *endPoint*	Specifies the ending point of the path. The to point in the constraint must be a timing end point and can be any of the following objects: • clocks (c:) (See *Clocks as From/To Points*, on page 511.) • registers (i:) • top-level output or bi-directional ports (p:) • black box inputs (i:) For more information, see: • *Defining From/To/Through Points for Timing Exceptions*, on page 146 in the *User Guide*. You can combine this option with -from or -through to get a specific path.

Copyright © 2013 Synopsys, Inc.

-through *throughPoint*	Specifies the intermediate points for the timing exception. Intermediate points can be: • combinational nets (n:) • hierarchical ports (t:) • pins on instantiated cells (t:) By default, the intermediate points are treated as an OR list, the exception is applied if the path crosses any points in the list. For more information, see: • *Defining From/To/Through Points for Timing Exceptions*, on page 146 in the *User Guide* You can combine this option with -to or -from to get a specific path. To keep the signal name intact throughout synthesis when you use this option, set the syn_keep directive (Verilog or VHDL) on the signal.
-max *delayValue*	Sets the maximum allowable delay for the specified path. This is an absolute value in nanoseconds and is shown as max analysis in the timing report.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

The define_path_delay timing constraint specifies point-to-point delay, in nanoseconds, for maximum delay constraints. You can specify the start, end, or through points using the -from, -to, or -through options or any combination of these options.

When you specify define_path_delay and you also have input or output delays defined, the synthesis tool adds the input or output delays to the path delay. The timing constraint that is forward-annotated includes the I/O delay with the path delay. This could result in discrepancies with the Xilinx P&R tool, which ignores the I/O delays and only reports the path delay.

Example

Here are examples of the path delay constraint.

```
define_path_delay -from {i:dmux.alua[5]}
   -to {i:regs.mem_regfile_15[0]} -max 0.800
```

The following constraint sets a max delay of 2 ns on all paths to the falling edge of the flip-flops clocked by clk1.

```
define_path_delay -to {c:clk1:f} -max 2
```

Here is an example of setting the path delay constraint on the pins between registers:

```
define_path_delay -from {i:myInst1_reg} -through {t:myInst2_net.Y}
   -to {i:myInst3_reg} -max 0.123
```

The constraint is defined from the output pin of myInst1, through pin Y of net myInst2, to the input pin of myInst3. If the instance is instantiated, a pin-level constraint applies on the pin, as defined. If the instance is inferred, the pin-level constraint is transferred to the instance.

Limitations

The following limitations are present:

- For through points specified on pins, the constraint is transferred to the connecting net. You cannot define a through point on a pin of an instance that has multiple outputs.
- When specifying a pin on a vector of instances, you cannot refer to more than one bit of that vector.

Copyright © 2013 Synopsys, Inc.

define_reg_input_delay

Speeds up paths feeding a register by a given number of nanoseconds. It is recommended that you use the FDC equivalent of this legacy command, which is described in *set_reg_input_delay*, on page 1131.

Syntax

define_reg_input_delay {*registerName***}** [**-route** *ns*] [**-disable**] [**-comment** *textString*]

Arguments

registerName	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register. See *Route Option*, on page 529 for an example of its use.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The legacy define_reg_input_delay timing constraint speeds up paths feeding a register by a given number of nanoseconds.

The Synopsys FPGA synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with define_clock). Use this constraint to speed up the paths feeding a register. For information about the equivalent SCOPE spreadsheet interface, see *Registers (Legacy)*, on page 531.

Copyright © 2013 Synopsys, Inc.

define_reg_output_delay

Speeds up paths coming from a register by a given number of nanoseconds. It is recommended that you use the FDC equivalent of this legacy command, which is described in *set_reg_output_delay*, on page 1132.

Syntax

define_reg_output_delay {*registerName***}** [**-route** *ns*] [**-disable**] [**-comment** *textString*]

Arguments

registerName	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths from the register. For an example of its use (on an input register), see *Route Option*, on page 529.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

The define_reg_output_delay constraint speeds up paths coming from a register by a given number of nanoseconds. The synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with define_clock). Use this constraint to speed up the paths coming from a register. For information about the equivalent SCOPE spreadsheet interface, see *Registers (Legacy)*, on page 531.

Copyright © 2013 Synopsys, Inc.

set_datapathonly_delay

Specifies a point-to-point path delay that excludes any applicable I/O delays or clock delays at the start and end points.

Syntax

set_datapathonly_delay *option* **{***value***}**

In the above syntax, *option* is one or more of the following:

-disable
-from {*port|inst|clock***}**
-rise_from *clock*
-fall_from *clock*
-to {*port|inst|clock***}**
-rise_to *clock*
-fall_to *clock*
-comment *textsString*

In the above options, the variable *clock* is a clock alias.

Description

The set_datapathonly_delay constraint specifies a point-to-point path delay that excludes any applicable I/O delays or clock delays at the start and end points. The constraint includes clock to out (Tco) and setup time (Tsu) as part of the data path delay.

Use this constraint in Xilinx designs, as the tool only forward-annotates the constraint for Xilinx designs. The set_datapathonly_delay constraint appears in the UCF file as a Xilinx TIMESPEC constraint with the DATAPATHONLY option. The original `sdc` command is also forward-annotated as a comment. For example:

```
# set_datapathonly_delay -from {p:din[*]} -to {i:int[*]} {4}
...
NET "din[7]" TNM = "from_1005_0";
INST "int[*]" TNM = "to_1005_0"; # int[7:0]
TIMESPEC "TS_1005_0" = FROM "from_1005_0" TO "to_1005_0"
   DATAPATHONLY 4.000;
```

Copyright © 2013 Synopsys, Inc.

When you specify this constraint, the tool ignores any clock skew or phase information. It constrains, analyzes, and forward-annotates just the data path between the specified points.

Object Naming Syntax

This section describes the prefix syntax for identifying objects when different design objects share the same name. Objects like Verilog modules, VHDL design units, component instances, ports, and internal nets can have shared names. The prefix identifiers are used in constraint files to assign timing constraints and synthesis attributes to the correct object. The syntax varies slightly, depending on the language used to define the module or component.

The objects in the SCOPE pull-down menus automatically use the correct prefix, but if you drag and drop an object from an HDL Analyst or other view, make sure you use the appropriate prefix for the language as described below:

- Verilog Naming Syntax, on page 1157
- VHDL Naming Syntax, on page 1158
- Object Naming Examples, on page 1159

Verilog Naming Syntax

This section provides syntax for object names in Verilog. No spaces are allowed in names. You can use the ***** and **?** characters as wildcards in names. These characters do not match the dot (**.**) used as a hierarchy separator. Examples of using wildcards in object names include the following: `mybus*`, `mybus[??]`, `mybus[*]`.

Verilog uses the following syntax for module names:

v: *cell*

v:	Identifies a view object
cell	The name of the Verilog module

Instance, port, and net names have the following syntax:

cell typespec object_path

Copyright © 2013 Synopsys, Inc.

cell	The name of the Verilog module
typespec	A single letter followed by a colon. The letter can be one of the following, and is used to determine the kind of object when objects have the same names: **i:** identifies the name of an instance. **p:** identifies the name of an entire port (the port itself). **b:** identifies the name of a slice of a port (bit-slice of the port). **n:** identifies the name of an internal net. This designation is required for all internal nets.
object_path	An instance path with a dot (.) used as a hierarchy separator in the name. The object path ends with the name of the desired object.

See *Object Naming Examples*, on page 1159.

VHDL Naming Syntax

This section provides syntax for object names in VHDL. No spaces are allowed in names. You can use the ***** and **?** characters as wildcards in names. These characters do not match the dot (**.**) used as a hierarchy separator. Examples of using wildcards in object names include the following: `mybus*`, `mybus[??]`, `mybus[*]`.

VHDL has the following syntax for design unit names:

v:[*lib*.]*cell* [.*view*]

v:	Identifies a view object
lib	The name of a library that contains the design unit. The default is the library containing the top-level design unit.
cell	The name of the design unit entity.
view	The name of the design unit architecture. The use of *view* with VHDL designs is optional; it is required only when the design unit has more than one architecture.

Copyright © 2013 Synopsys, Inc.

Instance, port, and net names have the following syntax:

[[*lib*.] *cell* [.*view*]] | [*typespec*] *objpath*

lib	The name of a library that contains the design unit. The default is the library containing the top-level design unit.
cell	The name of the VHDL entity.
view	The name of the design unit architecture. The use of *view* with VHDL designs is optional; it is required only when the design unit has more than one architecture.
typespec	A single letter followed by a colon that is used (if needed) to resolve the ambiguity of two or more objects in the design that have the same name. *typespec* can have the following values: **i:** identifies the name of an instance. **p:** identifies the name of an entire port (the port itself). **b:** identifies the name of a slice of a port (bit-slice of the port). **n:** identifies the name of an internal net. This designation is required for all internal nets.
objpath	An instance path using dot (.) as a hierarchy separator. The object path ends with the name of the desired object.

See *Object Naming Examples*, on page 1159.

Object Naming Examples

To identify all bits of instance statereg in module statemod:

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

To identify instances one level of hierarchy from the top with names that begin with state:

```
i:*.state*
```

To identify port mybus[19:0] instead of a driving register that also has the name mybus[19:0]:

```
p:mybus[19:0]
```

To identify top-level port mybus bit 1 instead of a driving register that also has the name mybus[1]:

```
b:mybus[1]
```


Copyright © 2013 Synopsys, Inc.

The following shows the constraint file entries that correspond to the previous source code examples:

```
define_attribute {statemod|i:statereg[*]} syn_encoding sequential
define_multicycle_path -to {*.*slowreg[*]} 2
```

CHAPTER 15

FPGA Design Constraint Syntax

The following describe the Tcl syntax for the design constraints specified with a SCOPE editor or entered manually into a constraint file. These constraints are common between the two supported constraint file formats.

The remainder of this section describes the constraint file syntax for the following FPGA design constraints:

- define_compile_point, on page 1162
- define_current_design, on page 1163
- define_io_standard, on page 1164

For information on timing constraints, see either *FPGA Timing Constraints*, on page 1100 or *Synplify-Style Timing Constraints (Legacy)*, on page 1135.

Copyright © 2013 Synopsys, Inc.

define_compile_point

The define_compile_point command, available only for the Synplify Pro and Synplify Premier tools, defines a compile point in a top-level constraint file. You use one define_compile_point command for each compile point you define. For the equivalent SCOPE spreadsheet interface, see *Compile Points*, on page 491. (Compile points are only available for certain technologies.)

This is the syntax:

define_compile_point [**-disable**] **{***moduleName***}**
-type {black_box|soft|hard|locked|locked, partition} [**-comment** *textString*]

-disable	Disables a previous compile point definition.
-type	Specifies the type of compile point. This can be black_box, soft, hard, locked, or locked, partition. See *Compile Point Types*, on page 1092 for more information.

Refer to *Object Naming Syntax*, on page 1157 for details about the syntax and prefixes for naming objects.

Here is a syntax example:

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

define_current_design

The define_current_design command, available only for the Synplify Pro and Synplify Premier tools, specifies the module to which the constraints that follow it apply. It must be the first command in a block-level or compile-point constraint file. The specified module becomes the top level for objects defined in this hierarchy and the constraints applied in the respective block-level or compile-point constraint file.

This is the syntax:

define_current_design {*regionName* | *libraryName*.*moduleName* **}**

Refer to *Object Naming Syntax*, on page 1157 for details about the syntax and prefixes for naming objects.

Here is an example:

```
define_current_design {lib1.prgm_cntr}
```

Objects in all constraints that follow this command relate to prgm_cntr.

Copyright © 2013 Synopsys, Inc.

define_io_standard

Specifies a standard I/O pad type to use for various Microsemi, Altera, and Xilinx families. See *I/O Standards, on page 489* for details of the SCOPE equivalent.

define_io_standard [**-disable**] **{p:***portName***} -delay_type input|output|bidir syn_pad_type {***IO_standard***}** [*parameter* **{***value***}**...]

In the above syntax:

portName is the name of the input, output, or bidirectional port.

-delay_type identifies the port direction which must be input, output, or bidir.

syn_pad_type is the I/O pad type (I/O standard) to be assigned to *portName*.

parameter is one or more of the parameters defined in the following table. Note that these parameters are device-family dependent.

Parameter	Function
syn_io_termination	The termination type; typical values are pullup and pulldown.
syn_io_drive	The output drive strength; values include low and high or numerical values in mA.
syn_io_dv2	Switch to use a 2x impedance value.
syn_io_dci	Switch for digitally-controlled impedance (DCI).
syn_io_slew	The slew rate for single-ended output buffers; values include slow and fast or low and high.

Example:

```
define_io_standard {p:DATA1[7:0]} -delay_type input
   syn_pad_type {LVCMOS_33} syn_io_slew {high}
   syn_io_drive {12} syn_io_termination {pulldown}
```

Copyright © 2013 Synopsys, Inc.

CHAPTER 16

Attribute and Directive Syntax

Synthesis attributes and directives let you direct the way a design is analyzed, optimized, and mapped during synthesis. This chapter shows you how to specify attributes and directives. It contains individual attribute and directive descriptions, syntax definitions, and examples. It contains the following information:

- How Attributes and Directives are Specified, on page 1166
- Summary of Attributes and Directives, on page 1169
- Attribute/Directive Descriptions, on page 1182
- Summary of Global Attributes, on page 1632

Copyright © 2013 Synopsys, Inc.

How Attributes and Directives are Specified

By definition, *attributes* control mapping optimizations and *directives* control compiler optimizations. Because of this difference, directives must be entered directly in the HDL source code or through a compiler design constraint file. Attributes can be entered either in the source code, in the SCOPE Attributes tab, or manually in a constraint file. For detailed procedures on different ways to specify attributes and directives, see *Specifying Attributes and Directives*, on page 245 in the *User Guide*.

Verilog files are case sensitive, so attributes and directives must be entered exactly as presented in the syntax descriptions. For more information about specifying attributes and directives using C-style and Verilog 2001 syntax, see *Verilog Attribute and Directive Syntax*, on page 758.

The SCOPE Attributes Tab

This section describes how to enter attributes using the SCOPE Attributes tab. To use the SCOPE spreadsheet, use this procedure:

1. Start with a compiled design, then open the SCOPE window.
2. Scroll if needed and click the Attributes tab.

	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. Click in the Attribute cell and use the pull-down menus to enter the appropriate attributes and their values.

The Attributes panel includes the following columns.

Column	Description
Enabled	(Required) Turn this on to enable the constraint.
Object Type	Specifies the type of object to which the attribute is assigned. Choose from the pull-down list, to filter the available choices in the Object field.
Object	(Required) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field. You can also drag and drop an object from the RTL or Technology view into this column.
Attribute	(Required) Specifies the attribute name. You can choose from a pull-down list that includes all available attributes for the specified technology. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the synthesis tool filters the available choices in the Object field. You must select an attribute before entering a value.
Value	(Required) Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate.
Val Type	Specifies the kind of value for the attribute. For example, string or boolean.
Description	Contains a one-line description of the attribute.
Comment	Contains any comments you want to add about the attributes.

For more details on how to use the Attributes panel of the SCOPE spreadsheet, see *Specifying Attributes Using the SCOPE Editor*, on page 250 in the *User Guide*.

When you use the SCOPE spreadsheet to create and modify a constraint file, the proper define_attribute or define_global_attribute statement is automatically generated for the constraint file. The following shows the syntax for these statements as they appear in the constraint file.

define_attribute {*object***}** *attributeName* **{***value***}**

define_global_attribute *attributeName* **{***value***}**

Copyright © 2013 Synopsys, Inc.

object	The design object, such as module, signal, input, instance, port, or wire name. The object naming syntax varies, depending on whether your source code is in Verilog or VHDL format. See *syn_black_box*, on page 1235 for details about the syntax conventions. If you have mixed input files (*Synplify Pro and Synplify Premier tools*), use the object naming syntax appropriate for the format in which the object is defined. Global attributes, since they apply to an entire design, do not use an *object* argument.
attributeName	The name of the synthesis attribute. This must be an attribute, not a directive, as directives are not supported in constraint files.
value	String, integer, or boolean value.

See *Summary of Global Attributes*, on page 1632 for more details on specifying global attributes in the synthesis environment.

Compiler Directives File Examples

This feature is available in the Synplify Premier tools.

You can use a compiler directives file to specify directives to be added to the source code. During compilation, the tool passes all active compiler constraint files to the compiler. The compiler references the object names in these files with the original RTL objects to assign the corresponding directive. For details about specifying directives with this file, see *Specifying Directives in a CDC File*, on page 248 in the *User Guide*.

The remainder of this section consists of code examples.

Example – Verilog Compiled into Default Library

Example – Verilog Compiled into a Defined Library

Example – VHDL Compiled into Default Library

Example – VHDL Compiled into a Defined Library

Summary of Attributes and Directives

The following sections summarize the synthesis attributes and directives:

- Attribute and Directive Summary Table (by Vendor), on page 1169
- Attribute and Directive Summary (Alphabetical), on page 1170

For detailed descriptions of individual attributes and directives, see the individual attributes and directives, which are listed in alphabetical order.

Attribute and Directive Summary Table (by Vendor)

For a list of vendor-specific attributes and directives, see the appropriate vendor documentation, as shown in the following table. For alphabetical summaries of attributes and directives, see *Attribute and Directive Summary (Alphabetical)*, on page 1170.

Vendor	Attributes and Directives	Synplify	Synplify Pro	Synplify Premier
Altera	Altera Attribute and Directive Summary, on page 1944	X	X	X
Lattice	Lattice Attribute and Directive Summary, on page 2021	X	X	
Microsemi	Microsemi Attribute and Directive Summary, on page 2059	X	X	
Xilinx	Xilinx Attribute and Directive Summary, on page 2136	X	X	X

Copyright © 2013 Synopsys, Inc.

Attribute and Directive Summary (Alphabetical)

The following table summarizes the synthesis attributes and directives. For detailed descriptions of each one, you can find them listed in alphabetical order.

Attribute/Directive	Description	Default
alsloc	Preserves relative placement in Microsemi designs.	
alspin	Assigns scalar or bus ports to I/O pin numbers in Microsemi designs.	
alspreserve	Specifies nets that must be preserved by the Microsemi place-and-route tool.	
altera_io_powerup	Controls the power up mode for Altera I/O registers.	low
altera_logiclock_location	Assigns a compile point to an Altera LogicLock region, specifying the location of the region.	floating
altera_logiclock_size	Assigns a compile point to an Altera LogicLock region, specifying the size of the region.	auto
black_box_pad_pin	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.	
black_box_tri_pins	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.	
CLOCK_DEDICATED_ROUTE	*Xilinx* Determines if the software enforces placement rules for the specified architecture.	1
diff_term	Specifies Differential Termination for true differential input I/O standards.	

Attribute/Directive	Description	Default
full_case	Specifies that a Verilog case statement has covered all possible cases.	
loc	Specifies I/O pin locations for Lattice designs.	
loop_limit	Specifies a loop iteration limit for for loops.	
orca_padtype	Specifies pad type for Lattice ORCA I/O.	
orca_props	Specifies Lattice ORCA I/O properties for forward annotation.	
parallel_case	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.	
pragma translate_off/pragma translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.	
syn_allow_retiming	*Synplify Pro and Synplify Premier* Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.	0
syn_allowed_resources	*Synplify Pro and Synplify Premier* Specifies the number of resources available globally or for compile points.	
syn_assign_to_region	*Synplify Premier* *Xilinx* Assigns logic to regions created on the device.	
syn_assign_to_slr	*Xilinx* Lets you assign logic to a specific Super Logic Region (SLR) for a Stacked Silicon Interconnect (SSI) technology that allows multiple die to be combined in a single device.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_auto_insert_bufg	*Synplify Premier* *Xilinx* Automatically inserts the BUFG primitive on high fanout nets to help reduce congestion for a physical synthesis design.	
syn_auto_partition	*Xilinx* Allows automatic partitioning to occur on Stacked Silicon Interconnect (SSI) devices.	
syn_black_box	Defines a black box for synthesis.	
syn_clean_reset	*Xilinx* Changes asynchronous reset registers, which cannot go into DSP48 blocks, into synchronous reset logic.	
syn_clock_priority	Establishes clock priority to be forward-annotated to the Xilinx UCF file.	
syn_cp_use_fast_synthesis	Specifies Fast Synthesis mode be used for selected manual compile points during automatic compile point synthesis.	
syn_diff_io	Controls inference of I/O buffers in Xilinx designs	0
syn_direct_enable	*Synplify Pro and Synplify Premier* Assigns clock enable nets to dedicated flip-flop enable pins. It can also be used as a compiler directive that marks flip-flops with clock enables for inference.	1
syn_direct_reset	Controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the reset when the design has a conditional reset on multiple candidates.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_direct_set	Controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the set when the design has a conditional set on multiple candidates.	
syn_disable_purifyclock	*Synplify Premier* *Xilinx* Disables purification of clocks. Clock purification duplicates nets that drive clock and data pins simultaneously, to ensure no clock net drives a data pin. Clock purification is on by default.	
syn_dspstyle	*Achronix* Determines whether the multiplier logical structures are mapped into BMACC56 or to logic. *Xilinx* Determines if an operator, register, or module/architecture is placed in the DSP48 component.	
syn_edif_bit_format	Formats bus names and delimiters in a Xilinx output EDIF file.	
syn_edif_name_length	Determines the length of port names in an Altera output EDIF file.	restricted
syn_edif_scalar_format	Formats scalar names in a Xilinx output EDIF file.	
syn_encoding	Specifies the encoding style for state machines.	Based on number of states.
syn_enum_encoding	Specifies the encoding style for enumerated types (VHDL only).	
syn_fast_auto	Speeds up signal transitions on output ports.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_force_pads	*Lattice* Enables/disables I/O insertion on a port level or global level.	
syn_force_seq_prim	*Synplify Pro and Synplify Premier* Indicates that the fix gated clocks algorithm can be applied to the associated primitive.	
syn_forward_io_constraints	Controls forward annotation of I/O timing constraints.	1
syn_fsm_correction	Specifies a Hamming 3 encoding used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers.	
syn_gatedclk_clock_en	*Synplify Pro and Synplify Premier* Specifies the name of the enable pin within a black box to support fix gated clocks.	
syn_gatedclk_clock_en_polarity	*Synplify Pro and Synplify Premier* Indicates the polarity of the clock enable port on a black box, so that the software can apply the algorithm to fix gated clocks.	
syn_global_buffers	*Synplify Pro and Synplify Premier* *Microsemi, Lattice, Xilinx* Determines the number of global buffers available.	
syn_hier	Determines hierarchical control across module or component boundaries.	soft
syn_highrel_ioconnector	Adds I/O connectors that interface to modules with distributed TMR or DWC, which allows you to access the signals within its boundaries ensuring their inputs and outputs are not a single-point of failure.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_insert_buffer (Microsemi) syn_insert_buffer (Lattice) syn_insert_buffer (Xilinx)	*Microsemi, Lattice, Xilinx* Inserts a clock buffer according to the vendor-specified value.	
syn_insert_pad	*Lattice, Xilinx* Inserts I/O buffers to either ports or nets.	
syn_isclock	Specifies that a black box input port is a clock, even if the name does not indicate it is one.	
syn_keep	Prevents the internal signal from being removed during synthesis.	
syn_loc	Specifies pin locations for I/O pins and cores, and forward-annotates this information to the place-and-route tool.	
syn_looplimit	Specifies a loop iteration limit for while loops.	
syn_macro	Prevents optimizations on instantiated macros.	0 (OFF)
syn_map_dffrs	*Xilinx* Allows the synthesis software to handle registers with asynchronous set and reset.	1
syn_maxfan	Overrides the default fanout guide for an individual input port, net, or register output.	
syn_max_memsize_reg	Allows you to specify the maximum number of registers that can be mapped to an inferred RAM above the limit for which the tool errors out.	
syn_multstyle	Determines how multipliers are implemented.	Varies
syn_netlist_hierarchy	Controls hierarchy generation in EDIF and VQM output files	1

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_noarrayports	Specifies ports as individual signals or bus arrays.	1
syn_noclockbuf	Disables automatic clock buffer insertion.	0
syn_noclockpad	*Lattice* Infers SB_GB and SB_IO instead of SB_GB_IO on clock nets.	
syn_no_compile_point	Ignores modules as compile points in the Automatic Compile Point flow.	
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.	
syn_packer_effort_level	*Synplify Premier* Determines the effort level for packing during Synplify Premier global placement.	Design-dependent
syn_pad_type	Specifies an I/O buffer standard for certain technology families.	
syn_pipeline	*Synplify Pro and Synplify Premier* Specifies that registers be moved into multipliers and ROMs in order to improve frequency.	1
syn_placer_effort_level	*Synplify Premier* Set this attribute to only place the FIFO, RAM, and DSP blocks in the design and forward-annotate them for P&R or optimize the design for routing at the expense of timing.	
syn_preserve	Preserves registers that can be optimized due to redundancy or constraint propagation.	
syn_preserve_sr_priority	*Synplify Pro and Synplify Microsemi* Forces set/resets to observe priority.	1

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_probe	*Synplify Pro and Synplify Premier* Adds probe points for testing and debugging.	
syn_radhardlevel	*Synplify and Synplify Pro: Microsemi* *Synplify Premier: Altera, Xilinx* For Microsemi designs, specifies the radiation-resistant design technique to use. For Altera and Xilinx designs, enables the triple modular redundancy and voting logic in the design.	
syn_ramstyle	Determines how RAMs are implemented.	registers
syn_reduce_controlset_size (Xilinx) syn_reduce_controlset_size	*Xilinx, Lattice* Controls the minimum size of the unique control-set on which control-set optimizations can occur.	
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.	
syn_rename_module	Rename a module without uniquification.	
syn_replicate	*Altera, Microsemi, Lattice, Xilinx* Controls replication, either globally or on registers.	0
syn_resources	Specifies resources used in black boxes.	
syn_romstyle (Altera) syn_romstyle (Lattice) syn_romstyle Attribute (Lattice iCE65/iCE40) syn_romstyle Attribute (Xilinx)	Specifies how inferred ROMs are implemented.	logic

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_rw_conflict_logic	Prevents the synthesis tool from automatically inferring a block RAM, and ensures that the tool does NOT insert bypass logic around the RAM to prevent a simulation mismatch.	1
syn_safe_case	Enables/disables the safe case option. When enabled, the high reliability safe case option turns off sequential optimizations for counters, FSM, and sequential logic to increase the circuit's reliability.	
syn_sharing	Enables/disables resource sharing of operators inside a module.	
syn_slow	Increases transition time of a specific output port or ports.	
syn_srlstyle	*Altera, Lattice, Xilinx* Determines how sequential shift components are implemented.	select_sr 1
syn_state_machine	Determines if the FSM Compiler extracts a structure as a state machine.	
syn_tco<n>	Defines timing clock to output delay through a black box. The *n* indicates a value between 1 and 10.	
syn_tpd<n>	Specifies timing propagation for combinational delay through a black box. The *n* indicates a value between 1 and 10.	
syn_tristate	Specifies that a black-box pin is a tristate pin.	
syn_tristatetomux	Controls conversion of tristate buses to multiplexers.	
syn_tsu<n>	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
syn_unconnected_inputs	You can specify that input pins of an instance be left floating (unassigned in the RTL).	
syn_unique_inst_module	Uniquifies a module of the instance while leaving other instances of the same module intact.	
syn_use_carry_chain (Lattice)	*Lattice* Enables/disables carry chain implementation for counters or adders.	
syn_use_carry_chain (Lattice)	*Lattice* Use this attribute to turn on or off carry chain implementation for arithmetic and combinational operators.	
syn_use_xilinx_placement	*Synplify Premier* *Xilinx* Specifies that Xilinx global placement is used for the Physical Plus flow.	
syn_useenables	Generates clock enable pins for registers.	1
syn_useioff (Altera) syn_useioff (Lattice) syn_useioff (Xilinx)	Controls register packing in I/Os. By default, the synthesis software automatically determines when register packing occurs based on timing. If you want to control when register packing occurs, use the syn_useioff attribute.	
syn_useioff (Lattice iCE65/iCE40)	*Lattice* Controls register packing in I/Os.	
syn_user_instance	Prevents user-instantiated instances from being merged or otherwise optimized away.	0
syn_vote_loops	Specifies whether or not to add a synchronous voter to a sequential feedback path for a distributed TMR module.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.	
xc_area_group	*Synplify Pro* Assigns a compile point to a Xilinx area group.	
xc_clockbuftype	Specifies clock buffer type for clock ports.	
xc_fast	*Synplify and Synplify Pro* Speeds up transition times for output drivers.	1
xc_fast_auto	Determines whether fast or slow output buffers are inferred.	1
xc_global_buffers	Determines the number of global buffers available.	
xc_loc	Specifies port locations.	
xc_map	Specifies primitive to use for relative location.	
xc_padtype	Specifies I/O buffer standard.	
xc_props	Specifies Xilinx attributes for forward annotation.	
xc_pullup/xc_pulldown	Defines pullup/pulldown ports.	1
xc_rloc	Specifies the relative location of instances.	
xc_slow	*Synplify and Synplify Pro* Slows down transition times for output drivers.	1
xc_use_keep_hierarchy	Preserves the hierarchy of the marked module for diagnostics and timing simulation.	1

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description	Default
xc_use_rpms	Specifies how to handle relationally placed macros (RPMs) during global placement and physical synthesis.	1
xc_use_timespec_for_io	Uses the TIMESPEC FROM ... TO command for writing flop-to-out and in-to-flop I/O constraints to the Xilinx constraint (`ncf`) file.	0
xc_uset	Specifies a group of instances for relative placement.	

Copyright © 2013 Synopsys, Inc.

Attribute/Directive Descriptions

Attribute and directive descriptions are provided in the remaining sections. Each description contains a definition, and for the constraint, Verilog and VHDL files, syntax and examples. Verilog files are case-sensitive, so the syntax must be entered exactly as shown.

alsloc

Attribute; Microsemi (except 500K and PA). Synplify and Synplify Pro.
Preserves relative placements of macros and IP blocks in the Microsemi Designer place-and-route tool. The alsloc attribute has no effect on synthesis, but is passed directly to Microsemi Designer.

Constraint File Syntax and Example

define_attribute {*object*} alsloc {*location*}

In the attribute syntax, *object* is the name of a macro or IP block and *location* is the row-column address of the macro or IP block.

Following is an example of setting alsloc on a macro (u1).

```
define_attribute {u1} alsloc {R15C6}
```

Verilog Syntax and Example

object /* **synthesis alsloc = "***location***"** */ **;**

Where *object* is a macro or IP block and *location* is the row-column string. For example:

```
module test(in1, in2, in3, clk, q);
input in1, in2, in3, clk;
output q;
wire out1 /* synthesis syn_keep = 1 */, out2;
and2a u1 (.A (in1), .B (in2), .Y (out1))
            /* synthesis alsloc="R15C6" */;
assign out2 = out1 & in3;
df1 u2 (.D (out2), .CLK (clk), .Q (q))
            /* synthesis alsloc="R35C6" */;
endmodule
```

```
module and2a(A, B, Y); // synthesis syn_black_box
input A, B;
output Y;
endmodule

module df1(D, CLK, Q); // synthesis syn_black_box
input D, CLK;
output Q;
endmodule
```

VHDL Syntax and Example

attribute alsloc of *object* : label is "*location*" ;

Where *object* is a macro or IP block and *location* is the row-column string. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test is
port (in1, in2, in3, clk : in std_logic;
      q : out std_logic);
end test;

architecture rtl of test is
signal out1, out2 : std_logic;

component AND2A
port (A, B : in std_logic;
      Y : out std_logic);
end component;

component df1
port (D, CLK : in std_logic;
      Q : out std_logic);
end component;

attribute syn_keep : boolean;
attribute syn_keep of out1 : signal is true;
attribute alsloc: string;
attribute alsloc of U1: label is "R15C6";
attribute alsloc of U2: label is "R35C6";
attribute syn_black_box : boolean;
attribute syn_black_box of AND2A, df1 : component is true;
```


Copyright © 2013 Synopsys, Inc.

```
begin
U1: AND2A port map (A => in1, B => in2, Y => out1);
out2 <= in3 and out1;
U2: df1 port map (D => out2, CLK => clk, Q => q);
end rtl;
```

alspin

Attribute; Microsemi. Available only in the *Synplify* and *Synplify Pro* tools. The alspin attribute assigns the scalar or bus ports of the design to Microsemi I/O pin numbers (pad locations). Refer to the Microsemi databook for valid pin numbers. If you want to use alspin for bus ports or for slices of bus ports, you must also use the syn_noarrayports attribute. See *Specifying Locations for Microsemi Bus Ports*, on page 831 of the *User Guide* for information on assigning pin numbers to buses and slices.

Constraint File Syntax and Example

define_attribute {*port_name***} alspin {***pin_number***}**

In the attribute syntax, *port_name* is the name of the port and *pin_number* is the Microsemi I/O pin.

```
define_attribute {DATAOUT} alspin {48}
```

Verilog Syntax and Example

object **/* synthesis alspin = "***pin_number***" */ ;**

Where *object* is the port and *pin_number* is the Microsemi I/O pin. For example:

```
module comparator (datain, clk, dataout);
output dataout /* synthesis alspin="48" */;
input [7:0] datain;
input clk;

// Other code
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

attribute alspin of *object* **:** *objectType* **is "***pin_number***" ;**

Where *object* is the port, *objectType* is signal, and *pin_number* is the Microsemi I/O pin. For example:

```
entity comparator is
   port (datain : in bit_vector(7 downto 0);
      clk : in bit;
      dataout : out bit);
attribute alspin : string;
attribute alspin of dataout : signal is "48";

-- Other code
```

alspreserve

Attribute; Microsemi (except 500K and PA). Available only in the *Synplify* and *Synplify Pro* tools. Specifies a net that you do not want removed (optimized away) by the Microsemi Designer place-and-route tool. The alspreserve attribute has no effect on synthesis, but is passed directly to the Microsemi Designer place-and-route software. However, to prevent the net from being removed during the synthesis process, you must also use the syn_keep directive.

Constraint File Syntax and Example

define_attribute {n:*net_name*} alspreserve {1}

In the attribute syntax, *net_name* is the name of the net to preserve.

```
define_attribute {n:and_out3} alspreserve {1};
define_attribute {n:or_out1} alspreserve {1};
```

Verilog Syntax and Example

object **/* synthesis alspreserve = 1 */ ;**

Where *object* is the name of the net to preserve. For example:

```
module complex (in1, out1);
input [6:1] in1;oh
output out1;
wire out1;
wire or_oosut1 /* synthesis syn_keep=1 alspreserve=1 */;
wire and_out1;
wire and_out2;
wire and_out3 /* synthesis syn_keep=1 alspreserve=1 */;
assign and_out1 = in1[1] & in1[2];
assign and_out2 = in1[3] & in1[4];
assign and_out3 = in1[5] & in1[6];
assign or_out1 = and_out1 | and_out2;
assign out1 = or_out1 & and_out3;
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

attribute alspreserve of *object* **: signal is true ;**

Where *object* is the name of the net to preserve.

For example:

```
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;

entity complex is
port (input : in std_logic_vector (6 downto 1);
      output : out std_logic);
end complex;

architecture RTL of complex is
signal and_out1 : std_logic;
signal and_out2 : std_logic;
signal and_out3 : std_logic;
signal or_out1 : std_logic;
attribute syn_keep of and_out3 : signal is true;
attribute syn_keep of or_out1 : signal is true;
attribute alspreserve of and_out3 : signal is true;
attribute alspreserve of or_out1 : signal is true;

begin
   and_out1 <= input(1) and input(2);
   and_out2 <= input(3) and input(4);
   and_out3 <= input(5) and input(6);
   or_out1 <= and_out1 or and_out2;
   output <= or_out1 and and_out3;
end;
```

Copyright © 2013 Synopsys, Inc.

altera_io_powerup

Attribute; Altera Stratix. Controls the power-up mode of registers in the I/O port (bit or bus). However, you can only specify this attribute if the I/O register does not have a defined preset or clear condition.

If the I/O register has a preset, powerup is set to high. If the I/O register has a clear, powerup is set to low. You can only specify the altera_io_powerup attribute to control the powerup mode if the I/O register is not defined.

By default, Quartus II sets the power-up mode of the I/O to low.

If the register cannot be packed into an I/O, the synthesis tool issues a warning saying it has ignored the attribute.

Constraint File Syntax and Examples

define_attribute {*port*} altera_io_powerup {high|low}

You can apply the altera_io_powerup attribute to a port, bus, or slice of a bus in the SCOPE spreadsheet, whereas in the source code this attribute can only be applied to a port definition.

1. In the SCOPE spreadsheet, click the port in the Object column.
2. In the Attribute column pulldown menu, choose altera_io_powerup.
3. Enter the appropriate string in the Value column: high or low.

Following is an example:

```
define_attribute {seg [31:0]} altera_io_powerup {high}
```


Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Example

object /***synthesis altera_io_powerup = "high | low" */ ;**

The following example demonstrates how to apply the altera_io_powerup attribute to a single-bit port in Verilog.

```
module dff_or(q, a, b, clk);
output q /* synthesis altera_io_powerup="low" */;
input a, b, clk;
reg q;

// Other code
```

VHDL Syntax and Example

attribute altera_io_powerup of *object* **:** *objectType* **is "high | low" ;**

The following demonstrates how to apply the altera_io_powerup attribute to a 32-bit bus port in VHDL.

```
entity eight_bit_uc is
port (clock : in std_logic;
      resetn : in std_logic;
      com : out std_logic_vector(3 downto 0);
      seg : out std_logic_vector(31 downto 0);
      porta : inout std_logic_vector(7 downto 0);
      portb : inout std_logic_vector(7 downto 0);
      portc : inout std_logic_vector(7 downto 0) );

attribute altera_io_powerup : string;
attribute altera_io_powerup of seg : signal is "high";
end EIGHT_BIT_UC;
```

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

altera_logiclock_location

Attribute; Altera (Stratix and Cyclone families) (Synplify Pro tool only).

The Synplify Premier tool *does not use* this attribute; it is used by other Synopsys FPGA products to assign a compile point to an Altera LogicLock™ region, specifying the location. In the Synplify Premier tool, this assignment is done during design planning (see below).

You apply this attribute to a compile point. The attribute assigns the compile point to an Altera LogicLock™ region, specifying the location. The synthesis tool provides a separate VQM file (`vqm`) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to an Altera LogicLock region within the region.

Note: Do *not* use this attribute with the Synplify Premier tool. Physical assignments can only be made in the Synplify Premier Design Planner view; not by assigning attributes. The Synplify Premier tool *ignores* applications of this attribute in constraint files: it does not forward-annotate them to the Quartus II place-and-route tool.

Specifying a Compile Point Region Using the Synplify Premier Tool

Instead of applying this attribute, to specify a compile-point region as an Altera LogicLock region you use the Synplify Premier Design Planner view popup menu command LogicLock -> Enabled – see *Design Planner Popup Menus, on page 558*. The Synplify Premier tool then forward-annotates the region location and size in a `.tcl` file for treatment by the Quartus II place-and-route tool as an Altera LogicLock region.

You can only designate a physical region as an Altera LogicLock region if you have defined the region as a *compile point*. For more information on compile points and the compile-point synthesis flow, see and *Synthesizing Compile Points, on page 1109* of the *User Guide*.

When you specify a compile point as an Altera LogicLock region, the Synplify Premier tool provides a separate VQM file (`vqm`) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to an Altera LogicLock region within the region.

Copyright © 2013 Synopsys, Inc.

Applying altera_logiclock_location to a Compile Point

Available only in the Synplify Pro tool, when you assign this attribute to a compile point, you typically also assign the companion attribute altera_logiclock_size to it – see *altera_logiclock_size*, on page 1193. If you assign only one of these two attributes, a default value is used for the unassigned attribute.

The attribute value floating means that Quartus II determines the actual LogicLock region location during its optimization process. The *default* value is floating.

Constraint File Syntax and Examples

define_attribute {v:*module*|*architectureName***}**
altera_logiclock_location {floating}

In the attribute syntax, *module* or *architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a *compile point*.

Following is an example of specifying a floating LogicLock region location:

```
define_attribute {v.work.a1} altera_logiclock_location {floating}
```

Note: You apply altera_logiclock_location only in a constraint file; you cannot use this attribute in source code

altera_logiclock_size

Attribute; Altera (Stratix and Cyclone families) (Synplify Pro tool only).

The Synplify Premier tool *does not use* this attribute; it is used by other Synopsys FPGA products to assign a compile point to an Altera LogicLock™ region, specifying the size. In the Synplify Premier tool, this assignment is done during design planning (see below).

You apply this attribute to a compile point. The attribute assigns the compile point to an Altera LogicLock™ region, specifying the size. The synthesis tool provides a separate VQM file (`vqm`) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to an Altera LogicLock region within the region.

Note: Do *not* use this attribute with the Synplify Premier tool. Physical assignments can only be made in the Synplify Premier Design Planner view; not by assigning attributes. The Synplify Premier tool *ignores* applications of this attribute in constraint files: it does not forward-annotate them to the Quartus II place-and-route tool.

Specifying a Compile Point Region Using the Synplify Premier Tool

Instead of applying this attribute, to specify a compile-point region as an Altera LogicLock region you use the Synplify Premier Design Planner view popup menu command LogicLock -> Enabled – see *Design Planner Popup Menus, on page 558*. The Synplify Premier tool then forward-annotates the region location and size in a `.tcl` file for treatment by the Quartus II place-and-route tool as an Altera LogicLock region.

You can only designate a physical region as an Altera LogicLock region if you have defined the region as a *compile point*. For more information on compile points and the compile-point synthesis flow, see *Synthesizing Compile Points, on page 1109* of the *User Guide*.

When you specify a compile point as an Altera LogicLock region, the synthesis tool provides a separate VQM file (`vqm`) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to an Altera LogicLock region within the region.

Copyright © 2013 Synopsys, Inc.

Applying altera_logiclock_size to a Compile Point

Available only in the Synplify Pro tool, when you assign this attribute to a compile point, you typically also assign the companion attribute altera_logiclock_location to it – see *altera_logiclock_location*, on page 1191. If you assign only one of these two attributes, a default value is used for the unassigned attribute.

The attribute value auto means that Quartus II determines the actual LogicLock region size during its optimization process. The *default* value is auto. If the value is a comma-separated pair of numbers, it explicitly specifies the size of the region.

Constraint File Syntax and Examples

define_attribute {v:*module*|*architectureName***}**
altera_logiclock_size {auto|*width*,*height***}**

In the attribute syntax:

- *module* or *architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a *compile point*
- *width* and *height* are the width and height, respectively, of the Altera LogicLock region

Following is an example of specifying an explicit LogicLock region size:

```
define_attribute {v.work.a1} altera_logiclock_size {30,20}
```

Note: You apply altera_logiclock_size only in a constraint file; you cannot use this attribute in source code.

black_box_pad_pin

Directive. Used with the syn_black_box directive and specifies that the pins on black box are I/O pads visible to the outside environment. To specify more than one port as an I/O pad, list the ports inside double-quotes ("), separated by commas, and without enclosed spaces.

To instantiate an I/O from your programmable logic vendor, you usually do not need to define a black box or this directive. The synthesis tool provides predefined black boxes for vendor I/Os. For more information, refer to your vendor section under FPGA and CPLD Support.

The black_box_pad_pin directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Verilog Syntax and Example

object **/* synthesis syn_black_box black_box_pad_pin = "***portList***" */ ;**

where *portList* is a spaceless, comma-separated list of the names of the ports on black boxes that are I/O pads. For example:

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
   /* synthesis syn_black_box black_box_pad_pin="GIN[2:0],Q" */;
```

VHDL Syntax and Example

attribute black_box_pad_pin of *object* **:** *objectType* **is "***portList***" ;**

where *object* is an architecture or component declaration of a black box. Data type is string; *portList* is a spaceless, comma-separated list of the black-box port names that are I/O pads.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;

Entity top is
generic ( width : integer := 4);
   port (in1,in2 : in std_logic_vector(width downto 0);
   clk : in std_logic;
```


Copyright © 2013 Synopsys, Inc.

```
   q : out std_logic_vector (width downto 0)
   );
end top;

architecture top1_arch of top is
component test is
   generic (width1 : integer := 2);
      port (in1,in2 : in std_logic_vector(width1 downto 0);
      clk : in std_logic;
      q : out std_logic_vector (width1 downto 0)
   );
end component;

attribute syn_black_box : boolean;
attribute black_box_pad_pin : string;
attribute syn_black_box of test : component is true;
attribute black_box_pad_pin of test : component is "in1(4:0),
in2[4:0], q(4:0)";

begin
   test123 : test generic map (width) port map (in1,in2,clk,q);
end top1_arch;
```

black_box_tri_pins

Directive. Used with the syn_black_box directive and specifies that an output port on a black box component is a tristate. This directive eliminates multiple driver errors when the output of a black box has more than one driver. To specify more than one tristate port, list the ports inside double-quotes ("), separated by commas (,), and without enclosed spaces.

The black_box_tri_pins directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Verilog Syntax and Examples

object /* **synthesis syn_black_box black_box_tri_pins = "***portList***"** */ **;**

where *portList* is a spaceless, comma-separated list of multiple pins.

Here is an example with a single port name:

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
   /* synthesis syn_black_box black_box_tri_pins="PAD" */;
```

Here is an example with a list of multiple pins:

```
module bb1(D,E,tri1,tri2,tri3,Q)
/* synthesis syn_black_box black_box_tri_pins="tri1,tri2,tri3" */;
```

For a bus, you specify the port name followed by all the bits on the bus:

```
module bb1(D,bus1,E,GIN,GOUT,Q)
   /* synthesis syn_black_box black_box_tri_pins="bus1[7:0]" */;
```

VHDL Syntax and Examples

attribute black_box_tri_pins of *object* **:** *objectType* **is "***portList***" ;**

where *object* is a component declaration or architecture. Data type is string, and *portList* is a spaceless, comma-separated list of the tristate output port names.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Copyright © 2013 Synopsys, Inc.

```
library ieee;
use ieee.std_logic_1164.all;

package my_components is
component BBDLHS
   port (D: in std_logic;
         E: in std_logic;
         GIN : in std_logic;
         GOUT : in std_logic;
         PAD : inout std_logic;
         Q: out std_logic );
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS : component is true;
attribute black_box_tri_pins : string;
attribute black_box_tri_pins of BBDLHS : component is "PAD";
end package my_components;
```

Multiple pins on the same component can be specified as a list:

```
attribute black_box_tri_pins of bb1 : component is
   "tri,tri2,tri3";
```

To apply this directive to a port that is a bus, specify all the bits on the bus:

```
attribute black_box_tri_pins of bb1 : component is "bus1[7:0]";
```

CLOCK_DEDICATED_ROUTE

Attribute; *Xilinx only*. This attribute supports *Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, Spartan-6, Virtex-4 and newer* devices for the Synplify Pro tool and *Spartan-3, Virtex-4 and newer* devices for the Synplify Premier tool.

Use this attribute with the Xilinx ISE 10.1 or later version of the place-and-route tool. The CLOCK_DEDICATED_ROUTE attribute is an advanced constraint that determines if the software enforces the placement rules for the architecture you specify. By default, clock placement rules must be followed, effective when this attribute is not used or is set to 1. When clock components are not optimally placed, an error can occur. However, if you set this attribute to 0, the synthesis software ignores the specific clock placement rule and continues through placement and routing. If possible, you should fix all clock placement rule violations in the design to ensure the best possible QoR. It is recommended that this constraint be used only in limited situations and only when absolutely necessary.

Set CLOCK_DEDICATED_ROUTE in the constraint file.

Constraint File Syntax

define_attribute {*object_name*} CLOCK_DEDICATED_ROUTE {1|0}

Where:

- *object_name* can be specified for any of the following:
  - net of an instance that is the driver
  - I/O pin that is the driver
  - input or output pins of the following primitives: BUFG, BUFR, DCM, PLLPMCD, GT11, GT11 DUAL, GT11 CLK, or GTP DUAL
- 1 enforces clock placement rules. This is the default.
- 0 allows you to override the clock placement rules.

For example:

```
define_attribute {b:clk} CLOCK_DEDICATED_ROUTE {0}
```

In this example, this attribute allows you to override the fast path between the I/O and the clock buffer or global clock buffer if the IOB is placed in the master clock IOB site. When a clock IOB component is not placed at an

Copyright © 2013 Synopsys, Inc.

optimal clock IOB site, normally an error is generated. However, if the CLOCK_DEDICATED_ROUTE attribute is applied on the clock pin to this clock buffer, the design is still synthesized and allowed to continue through placement and routing. This constraint disables all the clock placement rules associated with the specified clock pin. This attribute can potentially result in a significant degradation in QoR. When possible, you should correct the error in your design.

Copyright © 2013 Synopsys, Inc.

diff_term

Attribute; *Xilinx only*.

Use this attribute to specify Differential Termination for true differential input I/O standards. You can use this attribute with inputs for the buffer, such as IBUFDS and OBUFDS. Set diff_term in the constraint file or HDL source code.

Constraint File Syntax

define_attribute {*input*} diff_term {true|false}

Where:

- *input* can be specified for any of the following inferred primitives: IBUFDS, IBUFGDS, OBUFDS, OBUFGDS, and IOBUFGDS.
- The diff_term data type is Boolean.
  - A value of true enables differential termination on the pin.
  - A value of false disables differential termination on the pin.

For example:

```
define_attribute {p:in1_p} {diff_term} {true}

define_attribute {p:in1_n} {diff_term} {true}
```

Verilog Syntax and Example

object **/* synthesis diff_term=true|false*/**

```
module syn_diff_io (in1_p, in1_n, d, out1);
   input in1_p /*synthesis diff_term = true*/,
      in1_n /*synthesis diff_term = true*/,
      d;
   output reg out1;

reg in1;

always@(in1_p, in1_n)
   if (in1_p != in1_n)
      in1 = in1_p;
   else
      in1 = in1;
```


Copyright © 2013 Synopsys, Inc.

```
always@(posedge in1)
   out1 <= d;

endmodule
```

VHDL Syntax and Example

attribute diff_term of *object* **:** *objectType* **is true | false;**

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test is
   port  in1_p : in std_logic;
         in1_n : in std_logic;
         d     : in std_logic;
         out1  : out std_logic
         );
attribute diff_term: boolean;
attribute diff_term of in1_p,in1_n: signal is true;
end test;

architecture arc of test is
signal in1 : std_logic;

begin

process(in1_p,in1_n)
begin
   if(in1_p /= in1_n)then
      in1 <= in1_p;
   else
      in1 <= in1;
   end if;
end process;

process(in1)
begin
   if (in1'event and in1='1') then
      out1 <= d;
   end if;
end process;

end arc;
```

full_case

Directive. For Verilog designs only. When used with a case, casex, or casez statement, this directive indicates that all possible values have been given, and that no additional hardware is needed to preserve signal values.

Verilog Syntax and Example

object **/* synthesis full_case */**

where *object* is case, casex, or casez statement declarations.

The following casez statement creates a 4-input multiplexer with a pre-decoded select bus (a decoded select bus has exactly one bit enabled at a time):

```
module muxnew1 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or
c or d)

begin
   casez (select)
      4'b???1: out = a;
      4'b??1?: out = b;
      4'b?1??: out = c;
      4'b1???: out = d;
   endcase
end
endmodule
```

This code does not specify what to do if the select bus has all zeros. If the select bus is being driven from outside the current module, the current module has no information about the legal values of select, and the synthesis tool must preserve the value of the output out when all bits of select are zero. Preserving the value of out requires the tool to add extraneous level-sensitive latches if out is not assigned elsewhere through every path of the always block. A warning message like the following is issued:

Copyright © 2013 Synopsys, Inc.

```
"Latch generated from always block for signal out, probably missing
assignment in branch of if or case."
```

If you add the full_case directive, it instructs the synthesis tool not to preserve the value of out when all bits of select are zero.

```
module muxnew3 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)

begin
   casez (select) /* synthesis full_case */
      4'b???1: out = a;
      4'b??1?: out = b;
      4'b?1??: out = c;
      4'b1???: out = d;
   endcase
end
endmodule
```

If the select bus is decoded in the same module as the case statement, the synthesis tool automatically determines that all possible values are specified, so the full_case directive is unnecessary.

Assigned Default and full_case

As an alternative to full_case, you can assign a default in the case statement. The default is assigned a value of 'bx (a 'bx in an assignment is treated as a “don't care”). The software assigns the default at each pass through the casez statement in which the select bus does not match one of the explicitly given values; this ensures that the value of out is not preserved and no extraneous level-sensitive latches are generated.

The following code shows a default assignment in Verilog:

```
module muxnew2 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;
```

```
always @(select or a or b or c or d)
begin
   casez (select)
      4'b???1: out = a;
      4'b??1?: out = b;
      4'b?1??: out = c;
      4'b1???: out = d;
      default: out = 'bx;
   endcase
end
endmodule
```

Both techniques help keep the code concise because you do not need to declare all the conditions of the statement. The following table compares them:

Default Assignment	full_case
Stays within Verilog to get the desired hardware	Must use a synthesis directive to get the desired hardware
Helps simulation debugging because you can easily find that the invalid select is assigned a 'bx	Can cause mismatches between pre- and post-synthesis simulation because the simulator does not use full_case

Copyright © 2013 Synopsys, Inc.

loc

Attribute; The loc attribute specifies pin locations for Lattice I/Os, instances, and registers, and forward-annotates them to the place-and-route tool. If the attribute is on a bus, the software writes out bit-blasted constraints for forward-annotation. This attribute can only be specified in a constraint file.

Refer to the Lattice databook for valid pin location values.

Constraint File Syntax and Example

define_attribute {*portName*} loc {*pinLocations*}

In the syntax, *pinLocations* is a comma-separated list of pin locations.

The following example assigns a pad location to all bits of a bus:

```
define_attribute {DATA0[3:0]} loc {P14,P12,P11,P5}
```

loop_limit

Directive. For Verilog designs only. Specifies a loop iteration limit for for loops in the design when the loop index is a variable, not a constant. The compiler uses the default iteration limit of 1999 when the exit or terminating condition does not compute a constant value, or to avoid infinite loops. The default limit ensures the effective use of runtime and memory resources.

If your design requires a variable loop index or if the number of loops is greater than the default limit, use the loop_limit directive to specify a new limit for the compiler. If you do not, you get a compiler error. You must hard code the limit at the beginning of the loop statement. The limit cannot be an expression. The higher the value you set, the longer the runtime.

Note: VHDL applications use the syn_looplimit directive (see *syn_looplimit*, on page 1390).

Verilog Syntax and Example

*beginning_of_loop_statement /**** **synthesis loop_limit** *integer ***/*

The following is an example where the loop limit is set to 2000:

```
module test(din,dout,clk);
input[1999 : 0] din;
input clk;
output[1999 : 0] dout;
reg[1999 : 0] dout;
integer i;

always @(posedge clk)
begin
   /* synthesis loop_limit 2000 */
   for(i=0;i<=1999;i=i+1)
   begin
      dout[i] <= din[i];
   end
end

endmodule
```


Copyright © 2013 Synopsys, Inc.

orca_padtype

Attribute; Lattice ORCA FPSC, Series 2, Series 3, and Series 4. The orca_padtype attribute specifies the pad type for a Lattice ORCA I/O.

Constraint File Syntax and Example

define_attribute {*portName*} orca_padtype {*padType*}

Values for *padType* are defined in the Lattice FPGA databook. The default for inputs is IBM, which is a CMOS pad type. The default for outputs is OB6.

This example defines bit 3 of AIN as an ORCA TTL input pad.

```
define_attribute {AIN[3]} orca_padtype {IBT}
```

Verilog Syntax and Example

*object /** **synthesis orca_padtype = "***padType***"** **/* **;**

This is an example of the orca_padtype attribute in a Verilog source file:

```
module counter4 (cout, out, in, ce, load, clk, rst);

// Choose IBT padtype to select a TTL input pad for "load"
input load /* synthesis orca_padtype="IBT" */;
endmodule
```

VHDL Syntax and Example

attribute orca_padtype of *object* **:** *objectType* **is "***padType***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives. This is an example of the orca_padtype attribute in a VHDL source file:

```
entity counter4 is
   port (cout: out bit;
         output_vector: out bit_vector (3 downto 0);
         input_vector: in bit_vector (3 downto 0);
         ce, load, clk, rst: in bit );
-- Choose OB12 padtype for "output_vector".
-- The OB12 padtype is being set for all four
-- bits in "output_vector".
attribute orca_padtype : string;
attribute orca_padtype of output_vector: signal is "OB12";
end counter4;
```


Copyright © 2013 Synopsys, Inc.

orca_props

Attribute; *Lattice ORCA FPSC, Series 2, Series 3, and Series 4*. Specifies properties for forward-annotation. Add the attribute to ports or I/O pads. For more information about the properties and their values, see *I/O Properties and Values*, on page 1211.

Constraint File Syntax and Example

define_attribute {p:*port*} orca_props {*property*=*value*}

The following is an example of the syntax:

```
define_attribute {p:data_in} orca_props {LEVELMODE=LVDS}
```

Verilog Syntax and Example

object **/* synthesis orca_props = "*property*=*value*" */;**

where *object* can be a port (input, output, bidirectional, or tristate) or an I/O pad. For more information about the properties and their values, see *I/O Properties and Values*, on page 1211. The following are examples of the syntax:

```
output data_out /* synthesis orca_props = "LEVELMODE=LVDS" */;

output clk_out_ddr /* synthesis orca_props = "CLKMODE = SCLK" */;
```

If multiple properties are required, separate them with commas. This is an example of an object with more than one property:

```
output data_out
   /* synthesis orca_props = "BUFMODE=FAST,LEVELMODE=LVTTL" */;
```

VHDL Syntax and Example

attribute orca_props : string;
attribute orca_props of *object* **:** *objectType* **is "***property***=***value***" ;**

For more information about the properties and their values, see *I/O Properties and Values*, on page 1211. The following is an example of the VHDL syntax. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Copyright © 2013 Synopsys, Inc.

```
Entity test is port (a : in std_logic; b : out std_logic;);
attribute orca_props : string;
attribute orca_props of a :signal is "LEVELMODE=LVDS";
```

This is an example of the syntax when you have multiple properties:

```
Entity test is port (a : in std_logic; b : out std_logic;);
attribute orca_props : string;
attribute orca_props of a :signal is "LEVELMODE=LVTTL, AMPSMODE=6";
```

I/O Properties and Values

The following table lists the properties and values currently supported.

Property	Valid Values	Attached to	Notes
AMPSMODE	6 | 12 | 24	Bidirectional or output ports	Must be set in conjunction with levelmode set to LVTTL or LVCMOS2.
BUFMODE	SLOW | FAST	Bidirectional or output ports	Must be set in conjunction with levelmode set to LVTTL or LVCMOS2.
CLKMODE	SCLK | ECLK	Bidirectional or output ports, and I/O pads	
DELAYMODE	0 | 1	Input ports	
KEEPERMODE	off | on	Bidirectional or tristate ports	
LEVELMODE	LVTTL | LVCMOS2 | LVCMOS18 | PCI | SSTL2 | SSTL3 | HSTL1 | HSTL3 | GTL | GTLPLUS | PECL | LVPECL | LVDS	Any port	
RESISTOR	off | on	Input or output ports	Must be set in conjunction with levelmode set to LVDS or LVPECL.

Copyright © 2013 Synopsys, Inc.

parallel_case

Directive. For Verilog designs only. Forces a parallel-multiplexed structure rather than a priority-encoded structure. This is useful because case statements are defined to work in priority order, executing (only) the first statement with a tag that matches the select value.

If the select bus is driven from outside the current module, the current module has no information about the legal values of select, and the software must create a chain of disabling logic so that a match on a statement tag disables all following statements. However, if you know the legal values of select, you can eliminate extra priority-encoding logic with the parallel_case directive. In the following example, the only legal values of select are 4'b1000, 4'b0100, 4'b0010, and 4'b0001, and only one of the tags can be matched at a time. Specify the parallel_case directive so that tag-matching logic can be parallel and independent, instead of chained.

Verilog Syntax and Example

You specify the directive as a comment immediately following the select value of the case statement.

object **/* synthesis parallel_case */**

where *object* is a case, casex or casez statement declaration.

```
module muxnew4 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)

begin
   casez (select) /* synthesis parallel_case */
      4'b???1: out = a;
      4'b??1?: out = b;
      4'b?1??: out = c;
      4'b1???: out = d;
      default: out = 'bx;
   endcase
end
endmodule
```

If the select bus is decoded within the same module as the case statement, the parallelism of the tag matching is determined automatically, and the parallel_case directive is unnecessary.

Copyright © 2013 Synopsys, Inc.

pragma translate_off/pragma translate_on

Directive. Allows you to synthesize designs originally written for use with other synthesis tools without needing to modify source code. All source code that is between these two directives is ignored during synthesis.

Another use of these directives is to prevent the synthesis of stimulus source code that only has meaning for logic simulation. You can use pragma translate_off/translate_on to skip over simulation-specific lines of code that are not synthesizable.

When you use pragma translate_off in a module, synthesis of all source code that follows is halted until pragma translate_on is encountered. Every pragma translate_off must have a corresponding pragma translate_on. These directives cannot be nested, therefore, the pragma translate_off directive can only be followed by a pragma translate_on directive.

Note: See also, *translate_off/translate_on*, on page 1593. These directives are implemented the same in the source code.

Verilog Syntax and Example

The Verilog syntax for these directives is as follows:

/* pragma translate_off */

/* pragma translate_on */

For example:

```
module real_time (ina, inb, out);
input ina, inb;
output out;

/* pragma translate_off */
realtime cur_time;
/* pragma translate_on */

assign out = ina & inb;
endmodule
```

VHDL Syntax and Example

The following is the VHDL syntax for these directives:

pragma translate_off

pragma translate_on

For example:

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
   port (a, b, cin:in std_logic;
         sum, cout:out std_logic );
end adder;

architecture behave of adder is
signal a1:std_logic;

--pragma translate_off

constant a1:std_logic:='0';

--pragma translate_on

begin
   sum <= (a xor b xor cin);
   cout <= (a and b) or (a and cin) or (b and cin); end behave;
```


Copyright © 2013 Synopsys, Inc.

syn_allow_retiming

Attribute

Determines if registers can be moved across combinational logic to improve performance.

Vendor	Technology	Synthesis Tool
Achronix	:Speedster22iHD	Synplify Pro
Altera	Stratix and Cyclone families FLEX10K, APEX	Synplify Pro, Synplify Premier
Lattice	Lattice iCE65/iCE40	Synplify Pro
Microsemi	Axcelerator, Fusion IGLOO, IGLOOe, IGLOO+ ProASIC, ProASICPLUS, ProASIC3/3E/3L	Synplify Pro
Xilinx	All Virtex technologies Spartan-II/IIE, Spartan-3/3E, Spartan-6	Synplify Pro, Synplify Premier

syn_allow_retiming values

1 | true	Allows registers to be moved during retiming.
0 | false	Does not allow retimed registers to be moved.

Description

The syn_allow_retiming attribute determines if registers can be moved across combinational logic to improve performance.

The attribute can be applied either globally or to specific registers. Typically, you enable the global Retiming option in the UI (or the set_option -retiming 1 switch in Tcl) and use the syn_allow_retiming attribute to disable retiming for specific objects that you do not want moved. Do not use the syn_allow_retiming attribute with the Fast Synthesis flow.

Copyright © 2013 Synopsys, Inc.

syn_allow_retiming Syntax

Global	Object
Yes	Register

You can specify the attribute in the following files:

FDC	**define_attribute {*register*} syn_allow_retiming {1|0}** **define_global_attribute syn_allow_retiming {1|0}**	FDC Example
Verilog	*object* **/* synthesis syn_allow_retiming = 0 | 1 */ ;**	Verilog Example
VHDL	**attribute syn_allow_retiming of** *object* **:** *objectType* **is true | false ;**	VHDL Example

FDC Example

define_attribute {*register*} syn_allow_retiming {1|0}

define_global_attribute syn_allow_retiming {1|0}

Enable	Object Type	Object	Attribute	Value	Value Type	Description
☑	<any>	<Global>	syn_allow_retiming	1	boolean	Controls retiming of reg...

Verilog Example

object **/* synthesis syn_allow_retiming = 0 | 1 */ ;**

Here is an example of applying it to a register:

```
module parity_check (clk,data,count_one);
input clk;
input [20:0]data ;
output reg [3:0]count_one /* synthesis syn_allow_retiming=1*/;

integer i;
reg parity= 1'b1;
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge clk)
begin
   for (i=0; i<21; i=i+1)
      if (data[i] == parity)
         count_one<=count_one+1;


end
endmodule
```

VHDL Example

attribute syn_allow_retiming of *object* **:** *objectType* **is true | false ;**

The data type is Boolean. Here is an example of applying it to a register:

```
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.std_logic_unsigned.ALL;

ENTITY ones_cnt IS
   PORT ( vin  : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
      vout : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
      clk  : IN STD_LOGIC );
END ones_cnt;

ARCHITECTURE lan OF ones_cnt IS
signal vout_reg : STD_LOGIC_VECTOR (3 DOWNTO 0);
attribute syn_allow_retiming : boolean;
attribute syn_allow_retiming of vout_reg : signal is true;

BEGIN
   gen_vout: PROCESS(clk,vin)
      VARIABLE count : STD_LOGIC_VECTOR(vout'RANGE);
   BEGIN
      if rising_edge(clk) then
      count := (OTHERS => '0');
      FOR I IN vin'RANGE LOOP
         count := count + vin(i);
      END LOOP;
      vout_reg <= count;
   end if;
vout <= vout_reg;
END PROCESS gen_vout;
END lan;
```

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Effect of using syn_allow_retiming

Before applying syn_allow_retiming.

Verilog	output reg [3:0]count_one /* synthesis syn_allow_retiming=0*/;
VHDL	attribute syn_allow_retiming of vout_reg : signal is false;

Copyright © 2013 Synopsys, Inc.

The critical path and the worst slack for this scenario are given below along with the original count_one [3] register (before being retimed) as found in the design.

After applying syn_allow_retiming.

Verilog	output reg [3:0]count_one /* synthesis syn_allow_retiming=1*/;
VHDL	attribute syn_allow_retiming of vout_reg : signal is true;

The critical path and the worst slack for this scenario are shown along with the four '*_ret' retimed registers.

syn_allowed_resources

Attribute; Xilinx (all Virtex and Spartan,and all Altera Cyclone and Stratix families).

Synplify Pro and Synplify Premier

This attribute limits the available resources that a compile point can use. It is designed to be used in conjunction with compile points; do not use it on a non-compile point design, unless it is the topmost level of the design.

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 lets you account for this usage by limiting the resources a compile point can use.

If you do not set this attribute for a given compile point, the default maximum values for the region or chip are used. This can result in exhausting all region or chip resources for a single compile point.

The attribute value assigned to a given compile point includes the resources used by its children (at all levels). For example, if a compile point is limited by this attribute to a maximum of four block RAMs and it contains a compile point that uses two block RAMs, there are, at most, two block RAMs remaining for the parent compile point itself.

- For RAM resources, you can use multiple values to specifically define how many of each type of RAM can be used.

For information on compile points and the compile-point synthesis flow, see *Synthesizing Compile Points*, on page 1109 of the *User Guide*.

Constraint File Syntax

define_attribute {v:*module*|*architectureName***}**
syn_allowed_resources {*spec***=***n* [**,** *spec***=***n* ...]**}**

define_global_attribute syn_allowed_resources {*spec***=***n* [**,** *spec***=***n* ...]**}**

- *module* or *architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a *compile point*.
- *spec* can be any of the keywords in the following table depending on the technology you select.

Copyright © 2013 Synopsys, Inc.

- *n* is an integer that specifies the maximum number of resources of the specified type.

Keyword	Devices	Description
blockrams	All devices	block RAMs
M4Ks	Altera Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Cyclone, CycloneII	M4K RAM blocks (128 x 36 bits)
M512s	Altera Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Arria GX, Arria II GX	M512 RAM blocks (32 x 18 bits)
M9Ks	Altera Stratix III, Stratix IV, Stratix IV GT, Cyclone III	M9K RAM blocks (256 x 36 bits)
M144Ks	Altera Stratix III, Stratix IV, Stratix IV GT	M144K RAM blocks (4K x 36 bits)
M-RAM	Altera Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Arria GX, Arria II GX	Mega RAM blocks (4K x 144 bits)
MLAB	Altera Stratix III, Stratix IV, Stratix IV GT, Stratix V).	Memory LAB resources are configured as LUTRAMS.
M20Ks	Altera Arria GZ and Stratix V	M20K RAM blocks
M10Ks	Altera Arria V and Cyclone V	M10K RAM blocks
dsp_blocks	Altera all Stratix and Arria families	Digital signal processing blocks
dsps	Xilinx families with DSP and DSP48 blocks	Digital signal processing blocks. For newer technologies, from Virtex 4 on, use this value instead of blockmults.
blockmults	Xilinx families with DSP48 blocks	Block mults for DSP48 blocks. This is for backwards compatibility; use dsps.

Copyright © 2013 Synopsys, Inc.

If you make multiple assignments with different types of resources, separate them with commas. See *Application Examples*, on page 1224.

Verilog Syntax and Example

object /* **synthesis syn_allowed_resources = "***spec*=*N*" */ ;

object must be register definition (reg) signals. For descriptions of the *spec* values, see the table above.

Here is an example:

```
module ram4 (datain,dataout,clk);
output[31:0] dataout;
input clk;
input[31:0] datain;
reg [7:0] dataout[31:0] /* synthesis
syn_allowed_resources="M4Ks=0,dsp_blocks=1" */;

// Other code
```

See *Application Examples*, on page 1224 for ways to use the attribute.

VHDL Syntax and Example

attribute syn_allowed_resources of *object* **:** *objectType* **is "***spec***=** *N***" ;**

object can be a signal that defines a compile point or a label of a component instance. For descriptions of the *spec* values, see the table in *Constraint File Syntax*, on page 1221.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
entity ram4 is
   port (d : in std_logic_vector(7 downto 0);
         addr : in std_logic_vector(2 downto 0);
         we : in std_logic;
         clk : in std_logic;
         ram_out : out std_logic_vector(7 downto 0) );
end ram4;
```


Copyright © 2013 Synopsys, Inc.

```
library synplify;
architecture rtl of ram4 is
type mem_type is array (127 downto 0) of
   std_logic_vector (7 downto 0);
signal mem : mem_type;
-- mem is the signal that defines the compile point

attribute syn_allowed_resources : string;
attribute syn_allowed_resources of mem : signal is "block_rams=1";

-- Other code
```

See *Application Examples*, on page 1224 for ways to use the attribute.

Application Examples

Example: Four RAM Resources

The following example specifies a maximum of four block RAMs for the compile point v.work.a2:

```
define_attribute {v.work.a2} syn_allowed_resources {blockrams=4}
```

Altera Example: One RAM Resource

This example (for certain Altera devices) specifies only one M512 RAM resource for the compile point v.work.single. Other memory elements in the compile point are mapped to logic resources. You can overuse the logic resources if the allowed RAM resources are inadequate for the available memory elements inferred in a given compile point.

```
define_attribute {v:work.single} syn_allowed_resources
   {M4Ks=0, M512s=1}
```

Altera Example: No RAM Resources

This example (for certain Altera devices) specifies that no Altera specific RAM resources are allowed for the compile point v.work.single. All memory elements in the compile point will be mapped to logic resources.

```
define_attribute {v:work.single} syn_allowed_resources
   {M4Ks=0, M512s=0}
```

You can overuse the logic resources if the allowed RAM resources are inadequate for the available memory elements inferred in a given compile point.

Copyright © 2013 Synopsys, Inc.

Xilinx Examples: DSP Resources

This example (for certain Xilinx devices) allows the design v.work.TOP to have two DSP blocks.

```
define_attribute {v:work.TOP} syn_allowed_resources {dsps = 2}
```

For backward compatibility, DSP48 resources are treated as blockmults resources. To manually limit the use of blockmults, apply syn_allowed_resources on the top view. For example,

```
define_attribute {v:work.dsp_48_route_check}
   syn_allowed_resources {blockmults=1}
```

The resource limit for the syn_allowed_resources value includes both inferred and instantiated components. If you set DSP usage to 1 and your design infers one and instantiates one DSP48E primitive, both DSP48E primitives are honored.

```
module top ( input clk,
   input [10:0] a1,
   input [10:0] a2,
   input [10:0] a3,
   input [10:0] b1,
   input [10:0] b2,
   input [10:0] b3,
   input [47:0] c1,
   input [47:0] c2,
   input [47:0] c3,

   input [6:0] opmode,
   output reg [47:0] out_1,
   output reg [47:0] out_2,
   output reg [47:0] out_3)
      /*synthesis syn_allowed_resources="dsps=1" */;

// always @(posedge clk)
//    out_2 <= (a2*b2);

always @(posedge clk)
   out_3 <= (a3*b3) ;

// always @(posedge clk)
//    out_1 <= (a1*b1) ;
//

   DSP48E   DSP48E_inst1 (
```


Copyright © 2013 Synopsys, Inc.

```
            .A(a1),
            .B(b1),
            .C(c1),
            .P(out_1),
            .CLK(clk),
            .OPMODE(opmode)
        );

        DSP48E   DSP48E_inst2 (
            .A(a2),
            .B(b2),
            .C(c2),
            .P(out_2),
            .CLK(clk),
            .OPMODE(opmode)
        );

    endmodule
```

The following schematic shows that two DSP48E primitives are honored.

Copyright © 2013 Synopsys, Inc.

If you set DSP usage to 2 and your design infers only one DSP48E primitive, then this resource limit cannot be honored.

```
module top ( input clk,
   input [10:0] a1,
   input [10:0] a2,
   input [10:0] a3,
   input [10:0] b1,
   input [10:0] b2,
   input [10:0] b3,
   input [47:0] c1,
   input [47:0] c2,
   input [47:0] c3,

   input [6:0] opmode,
   output reg [47:0] out_1,
   output reg [47:0] out_2,
   output reg [47:0] out_3)
      /*synthesis syn_allowed_resources="dsps=2" */;


// always @(posedge clk)
//    out_2 <= (a2*b2);

always @(posedge clk)
  out_3 <= (a3*b3) ;

always @(posedge clk)
  out_1 <= (a1*b1) ;

//
// DSP48E   DSP48E_inst1 (
//  .A(a1),
//  .B(b1),
//  .C(c1),
// .P(out_1),
// .CLK(clk), comp
// .OPMODE(opmode)
//   );

   DSP48E   DSP48E_inst2 (
      .A(a2),
   .B(b2),
   .C(c2),
   .P(out_2),
```

```
    .CLK(clk),
    .OPMODE(opmode)
  );

endmodule
```

The following schematic shows that only one DSP48E primitive was honored.

Copyright © 2013 Synopsys, Inc.

syn_assign_to_region

Attribute; Available in the *Synplify Premier* tools with *Xilinx Virtex-5 and later devices; this includes Virtex-7* Stacked Silicon Interconnect (SSI) technology that allows multiple dies to be combined in a single device.

This attribute allows you to assign logic to physical regions specified on the device that is supported in the Physical Plus feature. This region constraint must be provided in the PDC file or the *design*_synplify.sdc generated from logic synthesis.

Alternatively, you can use the Design Planner to create regions and assign logic to them. Physical Plus recognizes and integrates the placement of these region constraints.

Constraint File Syntax and Example

The constraint file syntax for the attribute is the following:

define_attribute {*object***} syn_assign_to_region {***sliceLLloc:sliceURloc***}**

Where:

- *object* – The instance name for the logic to be assigned to a region. Specify as i:*instanceName*.
- *sliceLLloc:sliceURloc* – Specifies the lower-left and upper-right SLICE coordinate system locations for the region to be created on the device. Format for this constraint is SLICE_X*llYll*:SLICE_X*urYur*.

In the following example, instance or1200_cpu is assigned to a region created by the lower-left and upper-right SLICE coordinate system X12Y233 and X127Y281 for the specified architecture.

```
define_attribute {i:or1200_cpu} syn_assign_to_region
   {SLICE_X12Y233:SLICE_X127Y281}
```

syn_assign_to_slr

Attribute; Available for *Xilinx Virtex-7* Stacked Silicon Interconnect (SSI) technology that allows multiple dies to be combined in a single device. Each die is called a Super Logic Region (SLR) and is connected with 49 Super Long Lines (SLL) per column between each adjacent SLR.

Xilinx place-and-route software treats the SLICE coordinate system for SSI as a monolithic device. However, there is a timing penalty for signals that cross SLR boundaries, so place and route tries to automatically partition logic to the SLR. If these results are not optimal, you can use the syn_assign_to_slr attribute to assign logic to a specific SLR for the device. You can use this attribute with logic synthesis flows targeting Virtex-7 SSI devices.

Note: No resource estimation is performed for SLR assignments. If too much logic is assigned to a specific SLR, an error will occur in the Xilinx place-and-route tool.

The attribute can be specified in a constraint file. This attribute can be applied to instances or ports in the design. For more information about assigning SLR regions for the device, see *Xilinx Stacked Silicon Interconnect (SSI) Technology*, on page 2065.

Constraint File Syntax and Example

The constraint file syntax for the attribute is the following:

define_attribute {*object*} syn_assign_to_slr {*slr_value*}

Where:

- *object* – Can be specified for the following:
  - i: instance
  - p: port
- *slr_value* – The SLR can be specified as 0, 1, 2, or 3.

The bottom-most SLR is 0 and the top-most SLR is number 3. Each SLR follows the same SLICE coordinate system for the architecture. Depending on the device selected, the number of SLRs can range between 2 and 4.

```
define_attribute {i:inst_abc} syn_assign_to_slr {0}
```


Copyright © 2013 Synopsys, Inc.

Note: 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

Copyright © 2013 Synopsys, Inc.

syn_auto_insert_bufg

Attribute: *Synplify Premier tools; Xilinx Virtex-6 devices*

Use this attribute to automatically insert a BUFG primitive on high fanout nets that are non-critical or less critical control signals such as resets, and internal clock nets. This should help reduce routing congestion for a physical synthesis design. The Physical Plus option must be enabled for this attribute to be in effect.

Fanout criteria used to determine automatic BUFG insertion is based on the following:

- Must exceed the Fanout Guide specified on the Device tab of the Implementation Options panel or the value specified for the syn_maxfan attribute.
- Considers the values for the user-specified syn_insert_buffer and syn_global_buffers attributes.
- Is limited by BUFG resources; Xilinx guidelines specify two BUFG components on non-clock nets for the entire design.

To enable this attribute, set syn_auto_insert_bufg to 1. The attribute is applied globally for the design. You can only specify this attribute in the constraint file.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_global_attribute syn_auto_insert_bufg {1}

For example:

```
define_global_attribute syn_auto_insert_bufg {1}
```


Copyright © 2013 Synopsys, Inc.

syn_auto_partition

Attribute; Xilinx Virtex-7 SSI devices with the Synplify Premier tools

This attribute replaces the syn_partitioner_effort_level_beta attribute.

Allows automatic partitioning to occur on Stacked Silicon Interconnect (SSI) technology that combines multiple dies on a device. This partitioning helps to place logic optimally for highest die utilization and routibility of the design. You can set this attribute to 0 or 1 so that partitioning constraints can be forward annotated in the XDC file to guide Vivado place and route.

Note: You must use this attribute with the Physical Plus option enabled. Otherwise, logic synthesis does not include partition constraints in the XDC constraint file.

Constraint File Syntax

You can specify this attribute in a constraint file (FDC or PDC) and only set this attribute globally:

define_global_attribute syn_auto_partition {0 |1}

Where:

- 0—automatic partitioning always occurs when Physical Plus is run. The XDC file forward annotates specific SLICE constraints that incorporate partitioning and Physical Plus placement. This is the default.
- 1—automatically partitions the generated netlist for logic synthesis. The XDC file forward annotates the partition constraints along with all its other constraints to Vivado place and route.

Copyright © 2013 Synopsys, Inc.

syn_black_box

Directive. Specifies that a module or component is a black box with only its interface defined for synthesis. The contents of a black box cannot be optimized during synthesis. This directive has an implicit Boolean value of 1 or true. Common uses of syn_black_box include the following:

- Vendor primitives and macros (including I/Os).
- User-designed macros whose functionality is defined in a schematic editor, IP, or another input source in which the place-and-route tool merges design netlists from different sources.

To instantiate vendor I/Os and other vendor macros, you usually do not need to define a black box since the synthesis tool provides pre-defined black boxes for the vendor macros.

A module can be a black box whether or not it is empty. In mixed language designs, if you have a black box defined in one language at the top level but also have a description for it in another language, the tool checks the existing black-box definition. If the tool can replace the black-box declaration with the description from the other language, it does not use the syn_black_box directive you set. If you truly want a black box in your design, do one of the following:

- Set a syn_black_box directive on the module or entity in the HDL file that contains the description, not at the top level.
- Use the compiler constraints file (`cdc`) to set a comprehensive black box directive at the top level. For example, the following sets the syn_black_box attribute on all architectures of the sub entity in the MyLib library:

  ```
  define_attribute {v:MyLib.sub}{syn_black_box}{1}
  ```

  See *Specifying Directives in a CDC File*, on page 248 in the *User Guide* for the procedure and syntax details, and *Compiler Directives File Examples*, on page 1168 for examples.
- If your project includes black box descriptions in `srs`, `ngc`, or `edf` formats, the tool uses these black box descriptions even if you have specified syn_black_box at the top level. If you want to black box such a module, you must remove its `srs`, `ngc`, or `edf` description from the project.

Copyright © 2013 Synopsys, Inc.

Once you define a black box with syn_black_box, you use other source-code directives to define timing for the black box. You must add the directives to the source code because the models are specific to individual instances. There are no corresponding Tcl directives you can use in a constraint file.

Black-box Source Code Directives

Use the following directives with syn_black_box to characterize black-box timing:

syn_isclock	Specifies a clock port on a black box.
syn_tpd<n>	Sets timing propagation for combinational delay through the black box.
syn_tsu<n>	Defines timing setup delay required for input pins relative to the clock.
syn_tco<n>	Defines the timing clock to output delay through the black box.

Black Box Source Code Directives for Gated Clocks

The gated clocks feature is available only in the Synplify Pro and Synplify Premier tools. To fix gated clocks on black boxes, you must specify the following directives in addition to the ones listed in *Black-box Source Code Directives*, on page 1236.

syn_force_seq_prim	Indicates that gated clocks should be fixed for this black box.
syn_gatedclk_clock_en	Specifies the enable pin to be used in fixing the gated clocks.
syn_gatedclk_clock_en_polarity	Indicates the polarity of the clock enable port on a black box so that the software can fix gated clocks.

Black Box Pin Definitions

You define the pins on a black box with these directives in the source code:

black_box_pad_pin	Indicates that a black box is an I/O pad for the rest of the design.
black_box_tri_pins	Indicates tristates on black boxes.

Copyright © 2013 Synopsys, Inc.

For more information on black boxes, see *Instantiating Black Boxes in Verilog*, on page 749, and *Instantiating Black Boxes in VHDL*, on page 961.

Verilog Syntax and Example

object **/* synthesis syn_black_box */ ;**

where object is a module declaration. For example:

```
module bl_box(out,data,clk) /* synthesis syn_black_box */;

// Other code
```

VHDL Syntax and Example

attribute syn_black_box of *object* **:** *objectType* **is true ;**

where *object* is a component declaration, label of an instantiated component to define as a black box, architecture, or component. Data type is Boolean. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
architecture top of top is
component ram4
   port (myclk : in bit;
         opcode : in bit_vector(2 downto 0);
         a, b : in bit_vector(7 downto 0);
         rambus : out bit_vector(7 downto 0) );
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of ram4: component is true;

-- Other coding
```

The syn_black_box directive can also be specified on the architecture of an entity which must be defined as a black box as shown below:

```
library ieee;
use ieee.std_logic_1164.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity bbx is
   port (bbx_clk : in std_logic;
   bbx_in : in std_logic;
   result : out std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of bbx_clk: signal is true;
end bbx;

architecture bbx of bbx is
attribute syn_black_box : boolean;
attribute syn_black_box of bbx : architecture is true;

begin
end bbx;

library ieee;
use ieee.std_logic_1164.all;

entity bbx_parent is
   port (data, bob: in std_logic;
         q: out std_logic );
end bbx_parent;

architecture bbx_parent of bbx_parent is
component bbx
   port (bbx_clk : in std_logic;
         bbx_in : in std_logic;
         result : out std_logic );
end component;

begin
-- Simple component instantiation
bbx1: bbx
   port map(bbx_clk => bob, bbx_in => data, result => q);
end bbx_parent;
```

syn_clean_reset

Attribute; Xilinx Virtex-4, Virtex-5, and Virtex-6. The syn_clean_reset attribute changes asynchronous reset registers, which cannot go into DSP48 blocks, into synchronous reset logic. The resulting implementation is not logically equivalent to the one specified in the RTL code and does not match the RTL and gate-level simulation.

Note: This attribute can only be applied on registers that get packed into the DSP48 blocks. This attribute does not work for all registers in the design.

Constraint File Syntax and Example

define_global_attribute syn_clean_reset {dsp48_no_simulate,one_flop}

```
define_global_attribute syn_clean_reset
   {dsp48_no_simulate,one_flop}
```

Verilog Syntax and Example

object **/* synthesis syn_clean_reset = "dsp48_no_simulate,one_flop" */ ;**

In the Verilog code (apply to a module):

```
module test(in1_p, in1_n, clk, out1)
   /* synthesis syn_clean_reset = "dsp48_no_simulate,one_flop" */;
input in1_p, in1_n ;
input clk;
output out1
```

VHDL Syntax and Example

attribute syn_clean_reset of *object* **:** *objectType* **is "dsp48_no_simulate,one_flop";**

In the VHDL code (apply to top-level architecture):

```
entity test is
   port (in1_p, in1_n : in std_logic;
         clk : in std_logic;
         out1 : out std_logic);
         attribute syn_clean_reset : string;
         attribute syn_clean_reset of test : architecture is
            "dsp48_no_simulate,one_flop";
end test;
```


Copyright © 2013 Synopsys, Inc.

See *VHDL Attribute and Directive Syntax*, on page 963 for differe‘nt ways to specify VHDL attributes and directives.

syn_clock_priority

Attribute; Xilinx. The syn_clock_priority attribute lets you set a clock priority to resolve logical or physical clock conflicts. This ensures that the clocks are forward-annotated correctly in the UCF file.

Use the syn_clock_priority attribute to ensure that the correct clock is forward-annotated. For example, you can use it in the following cases:

- 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 FPGA synthesis tools allow multiple clocks to propagate along a single net, either through unate logic or through a mux for timing all possible clocks and clock combinations. Xilinx ISE cannot do this, and uses the PRIORITY keyword in UCF to indicate clock priority in case of conflict. You can set clock priority with this attribute.

See *Setting Clock Priority in Xilinx Designs (Legacy)*, on page 196 in the *User Guide* for more information.

The tool forward-annotates the clock priority to the UCF file in a TIMESPEC or PERIOD statement. See *Forward-annotation of Clock Priority to UCF*, on page 1242 for details.

Constraint File Syntax and Example

You can only set this attribute in the constraint file. You set the attribute on ports or nets, and the tool applies the value to the clock. Clocks can be declared or derived clocks. You can also set it on a BUFG instance, but not on any other instances.

define_attribute {n:|p:|i:*netName|portName|BUFGname***}**
{syn_clock_priority} {*value***}**

The *value* must be greater than or equal to 1, with 1 being the highest priority. Take the following clock declaration:

```
define_clock {n:u_fx_clkrstgen.clk_100_dcm} -name {clk_100}
-period 10 -clockgroup sys_group -rise 0 -fall 5
```


Copyright © 2013 Synopsys, Inc.

If you apply syn_clock_priority as follows, the clock on the specified net is assigned the highest priority, 1:

```
define_attribute {n:u_fx_clkrstgen.clk_100_dcm}
{syn_clock_priority} {1}
```

The following example sets the highest priority for the DCM CLKFX output:

```
define_attribute {n:dcm_module_b.clk0fx} {syn_clock_priority} {1}
```

Forward-annotation of Clock Priority to UCF

A typical clock statement in the UCF file looks like this:

```
TIMESPEC "TS_u_fx_clkrstgen_clk_100_dcm" = PERIOD
"u_fx_clkrstgen_clk_100_dcm" 10.000 ns HIGH 50.00% PRIORITY 1;
```

The following shows an example where clka is the DCM source clock and is the root clock for other related clocks. There is an overriding clock, clk0fx.

```
#Begin clock constraints
NET "clk0fx" TNM_NET = "clk0fx";
TIMESPEC "TS_clk0fx" = PERIOD "clk0fx" 20.000 ns HIGH 50.00% PRIORITY 1;
NET "clka" TNM_NET = "clka";
TIMESPEC "TS_clka" = PERIOD "clka" 10.000 ns HIGH 50.00%;
NET "clkb" TNM_NET = "clkb";
TIMESPEC "TS_clkb" = PERIOD "clkb" "TS_clka" * 1.000000 HIGH 50.00%;
#End clock constraints

TIMESPEC "TS_clkf_clka" = FROM "clk0fx" TO "clka" TIG;
TIMESPEC "TS_clka_clkf" = FROM "clka" TO "clk0fx" TIG;
```

In the UCF file, a TIMESPEC statement with a priority value overrides a competing TIMESPEC without a priority value. For competing TIMESPECs that each have priority values, the one with the lowest positive integer value wins. If the priority values are the same, no priority is established between them.

The FPGA tools handle derived and declared clocks slightly differently when they forward-annotate the priority:

Attribute set on...	FPGA Synthesis Forward-annotation
Declared clock	Directly forward-annotated to the TIMESPEC/PERIOD for the clock.
Derived clock	First generates a TIMESPEC/PERIOD that represents the nature of the derived clock, and then forward-annotates this as a declared clock.

Messages and Warnings

The tool generates messages in certain situations:

Situation	Message
When derived clocks are overridden	Message to set clock priority with the syn_clock_priority attribute
When the tool detects multiple timing for a path	
When the priority set on the DCM base clock is higher than the priority of the override clock on a DCM output	Usage warnings
When you assign priority for a clock that is one of multiple clocks specified for the same object with the create_clock FPGA timing constraint	
When you assign the same priority to both inputs of a mux	

Copyright © 2013 Synopsys, Inc.

syn_cp_use_fast_synthesis

Attribute: You can use Fast Synthesis mode when specifying manual compile points in the Automatic Compile Point flow. Fast Synthesis is applied globally when set from the Project view. However, you can set the syn_cp_use_fast_synthesis attribute to apply fast synthesis mode for specific manual compile points. Fast Synthesis is available for the Synplify Premier tool.

To enable this attribute, set syn_cp_use_fast_synthesis to 1. The attribute can be applied on a selected manual compile point view/module for the design. Define the compile points in the top-level constraint file.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute syn_cp_use_fast_synthesis {v:*cp_name*} {1|0}

For example:

```
define_attribute syn_cp_use_fast_synthesis {v:datapath} {1}
```

For information about Automatic Compile Points, see *The Automatic Compile Point Flow*, on page 1109.

syn_diff_io

Attribute: Xilinx. The syn_diff_io attribute controls differential I/O buffer inferencing in Xilinx designs: IBUFDS, IBUFGDS, OBUFDS, OBUFTDS and IOBUFDS. Attach the attribute to the inputs of the buffer. Use this attribute to identify differential input buffers when using either combinatorial or sequential style to code it.

The syn_diff_io data type is Boolean. A value of 1 or true enables automatic differential I/O buffer inferencing. The default is false or 0, no automatic inferencing.

Constraint File Syntax and Example

define_attribute {*input*} syn_diff_io {true|false}

For example:

```
define_attribute {in1_n} syn_diff_io {true}
define_attribute {in1_p} syn_diff_io {true}
```

Verilog Syntax and Example

object **/* synthesis syn_diff_io = 1| 0*/ ;**

Example1:

```
module test(in1_p, in1_n, clk, out1);
input in1_p/* synthesis syn_diff_io = 1*/,
   in1_n/* synthesis syn_diff_io = 1*/, clk;
output out1;
reg out1;
reg in1;

always@(in1_p or in1_n)
   if (in1_p != in1_n) in1 = in1_p;

always@(posedge clk)
   out1 <= in1;

endmodule
```


Copyright © 2013 Synopsys, Inc.

Example 2:

```
module test ( din, clk , dout1 , dout2 ) ;
input din , clk ;
output dout1 /* synthesis syn_diff_io = 1 */ ;
output dout2 /* synthesis syn_diff_io = 1 */ ;
reg dout ;

always @(posedge clk)
dout <= din ;
assign dout1 = dout ;
assign dout2 = ~dout ;
endmodule
```

VHDL Syntax and Example

attribute syn_diff_io of *object* **:** *objectType* **is true | false;**

Example 1:

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
   port (in1_p : in std_logic;
         in1_n : in std_logic;
         clk : in std_logic;
         out1 : out std_logic );
attribute syn_diff_io: boolean;
attribute syn_diff_io of in1_p,in1_n: signal is true;
end test;

architecture arc of test is
signal in1 : std_logic;
begin
   process(in1_p,in1_n)
   begin
      if(in1_p /= in1_n)then
         in1 <= in1_p;
      end if;
   end process;
```

```
      process(clk)
      begin
         if(clk'event and clk = '1')then
            out1 <= in1;
         end if;
      end process;

   end arc;
```

Example 2:

```
   library ieee;
   use ieee.std_logic_1164.all;

   entity test is
      port (din : in std_logic;
            clk : in std_logic;
            dout1 : out std_logic;
            dout2 : out std_logic );
   attribute syn_diff_io: boolean;
   attribute syn_diff_io of dout1,dout2: signal is true;
   end test;

   architecture arc of test is
   signal dout : std_logic;
   begin
      process(clk)
      begin
         if(clk'event and clk = '1')then
            dout <= din;
         end if;
      end process;

   dout1 <= dout;
   dout2 <= not dout;
   end arc;
```


Copyright © 2013 Synopsys, Inc.

syn_direct_enable

Attribute, Directive

Controls the assignment of a clock enable net to the dedicated enable pin of a storage element (flip-flop).

Technology	Default Value	Global	Object	Synthesis Tool
Actel: Axcelerator	None	No	Net	Synplify, Synplify Pro
Altera: All	None	No	Net	All
Lattice: LatticeECP3, ECP2S, ECP2M, ECP2; LatticeECP, EC; LatticeXP2, XP; LatticeSC, SCM; MachXO, and ORCA	None	No	Net	Synplify, Synplify Pro
Xilinx: All	None	No	Net	All

syn_allow_retiming values

1 | true	Enables nets to be assigned to the clock enable pin.
0 | false	Does not assign nets to the clock enable pin.

Description

The syn_direct_enable attribute controls the assignment of a clock enable net to the dedicated enable pin of a storage element (flip-flop). Using this attribute, you can direct the mapper to use a particular net as the only clock enable when the design has multiple clock-enable candidates.

As a directive, you use syn_direct_enable to infer flip-flops with clock enables. To do so, enter syn_direct_enable as a directive in source code, not the SCOPE spreadsheet.

Copyright © 2013 Synopsys, Inc.

syn_direct_enable Syntax

FDC	**define_attribute {*object*} syn_direct_enable {1}**	FDC Example
Verilog	*object* **/* synthesis syn_direct_enable = 1 */ ;**	Verilog Example
VHDL	**attribute syn_direct_enable of** *object* **:** *objectType* **is true;**	VHDL Example

FDC Example

Enable	Object Type	Object	**Attribute**	Value	Value Type	Description
☑	<any>	<Global>	syn_direct_enable	1	boolean	Prefered clock enable

Verilog Example

```
module direct_enable(q1, d1, clk, e1, e2, e3);
parameter size=5;
input [size-1:0] d1;
input clk;
input e1,e2;
input e3 /* synthesis syn_direct_enable = 1 */;
output reg [size-1:0] q1;

(posedge clk)
   if (e1&e2&e3)
      q1 = d1;
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;

entity direct_enable is
   port (
         d1 : in  std_logic_vector(4 downto 0);
         e1,e2,e3,clk : in  std_logic;
         q1 : out std_logic_vector(4 downto 0) );
attribute syn_direct_enable: boolean;
attribute syn_direct_enable of e3: signal is true;
end;

architecture d_e of direct_enable is
begin
   process (clk) begin
      if (clk = '1' and clk'event) then
         if (e1='1' and e2='1' and e3='1') then
            q1<=d1;
         end if;
      end if;
   end process;
end architecture;
```

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Effect of Using syn_direct_enable

Before applying syn_direct_enable:

Verilog	`input e3 /* synthesis syn_direct_enable = 0 */;`
VHDL	`attribute syn_direct_enable of e3: signal is false;`

Copyright © 2013 Synopsys, Inc.

FDE
D
C
Q
CE
q1[4]
IBUF
e3
I
O
e3_ibuf
LUT3_80
IBUF
e2
I
O
e2_ibuf
q13
IBUF
e1
I
O
e1_ibuf

After applying syn_direct_enable:

Verilog	`input e3 /* synthesis syn_direct_enable = 1 */;`
VHDL	`attribute syn_direct_enable of e3: signal is true;`

Copyright © 2013 Synopsys, Inc.

syn_direct_reset

Attribute/Directive

Controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop).

Vendor	Technology
Lattice	ECP, SC, XP, MachXO families
Xilinx	Virtex-4 and newer families

syn_direct_reset Values

Value	Description
1 true	Enables the software to assign a net to the dedicated reset pin of a synchronous storage element (flip-flop).
0 false (Default)	Disables the software from assigning a net to the dedicated reset pin of a synchronous storage element (flip-flop).

Description

The syn_direct_reset attribute controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop). You can direct the mapper to only use a particular net as the reset when the design has a conditional reset on multiple candidates. This attribute can be applied to separate AND or OR logic and is valid for only one input of the reset logic cone.

syn_direct_reset Syntax

Global Attribute	Object
No	Port

Copyright © 2013 Synopsys, Inc.

The following table summarizes the syntax in different files:

FDC	**define_attribute syn_direct_reset {0|1}**	SCOPE Example
Verilog	*object* **/* synthesis syn_direct_reset = 0|1*/**	Example — Verilog syn_direct_reset
VHDL	**attribute syn_direct_reset : boolean;** **attribute syn_direct_reset of** *Object* **: signal is true|false;**	Example — VHDL syn_direct_reset

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑		p:a	syn_direct_reset	1		

Example — Verilog syn_direct_reset

Example — VHDL syn_direct_reset

Copyright © 2013 Synopsys, Inc.

Effect of Using syn_direct_reset

The following figure shows the attribute set to 1. The software assigns a net for the design to the dedicated reset pin of a synchronous storage element (flip-flop):

Verilog	**input a /*synthesis syn_direct_reset=1*/**;
VHDL	**attribute syn_direct_reset of a : signal is true;**

The next figure shows the attribute set to 0. The software does not assign a net for the design to the dedicated reset pin of a synchronous storage element (flip-flop):

Verilog	**input a /*synthesis syn_direct_reset = 0*/;**
VHDL	**attribute syn_direct_reset of a : signal is false;**

Copyright © 2013 Synopsys, Inc.

syn_direct_set

Attribute/Directive

Controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop).

Vendor	Technology
Lattice	ECP, SC, XP, MachXO families
Xilinx	Virtex-4 and newer families

syn_direct_set Values

Value	Description
1 true (Default)	Enables the software to assign a net to the dedicated set pin of a synchronous storage element (flip-flop).
0 false	Disables the software from assigning a net to the dedicated set pin of a synchronous storage element (flip-flop).

Description

The syn_direct_set attribute controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop). You can direct the mapper to only use a particular net as the set when the design has a conditional set on multiple candidates. This attribute can be applied to separate OR logic and is valid for only one input of the set logic cone.

syn_direct_set Syntax

Global Attribute	Object
No	Port

Copyright © 2013 Synopsys, Inc.

The following table summarizes the syntax in different files:

FDC	**define_attribute syn_direct_set {0|1}**	SCOPE Example
Verilog	*object* **/* synthesis syn_direct_set = 0|1*/**	Example — Verilog syn_direct_set
VHDL	**attribute syn_direct_set : boolean;** **attribute syn_direct_set of** *Object* **: signal is true|false;**	Example — VHDL syn_direct_set

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑		p:a	syn_direct_set	1		

Example — Verilog syn_direct_set

Example — VHDL syn_direct_set

Copyright © 2013 Synopsys, Inc.

Effect of Using syn_direct_set

The following figure shows the attribute set to 1. The software assigns a net for the design to the dedicated set pin of a synchronous storage element (flip-flop):

Verilog	**input a /*synthesis syn_direct_set=1*/;**
VHDL	**attribute syn_direct_set of a : signal is true;**

The next figure shows the attribute set to 0. The software does not assign a net for the design to the dedicated set pin of a synchronous storage element (flip-flop):

Verilog	**input a /*synthesis syn_direct_set = 0*/;**
VHDL	**attribute syn_direct_set of a : signal is false;**

Copyright © 2013 Synopsys, Inc.

syn_disable_purifyclock

Attribute: Xilinx. The syn_disable_purifyclock attribute disables purification of clocks. Clock purification is on by default. Clock purification duplicates nets that drive clock and data pins simultaneously, to ensure clock nets do not drive data pins. Currently, clock purification applies for generated clocks.

This attribute can be set globally or on a net. The syn_disable_purifyclock attribute is supported in the Synplify Premier tools.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute {*object*} syn_disable_purifyclock {1}

define_global_attribute syn_disable_purifyclock {1}

Where *object* is a net. For example:

```
n:clk1
```

Verilog Syntax and Examples

For this example, the syn_disable_purifyclock attribute is only applied on net c, which is the last variable in the wire declaration or variable associated with the directive declaration in the syntax below.

```
wire a,b,c /* synthesis syn_disable_purifyclock=1 */;
```

To apply syn_disable_purifyclock to all the nets, use one of the following methods:

Declare each individual net separately as shown below.

```
wire a /* synthesis syn_disable_purifyclock=1 */;
wire b /* synthesis syn_disable_purifyclock=1 */;
wire c /* synthesis syn_disable_purifyclock=1 */;
```

Use the Verilog 2001 Parenthetical Comments, to declare the syn_disable_purifyclock attribute as a single line statement.

```
(* syn_disable_purifyclock=1 *) wire a,b,c;
```

Copyright © 2013 Synopsys, Inc.

Example:

```
module FF2LUT2FF
(
input  wire clk,clk1,
input  wire din,din_r,din_u1,
output   dout,dout_r,
input [4:0] din1,din2,
output reg [4:0]  dout1,dout2
);
wire gen_clk /* synthesis syn_disable_purifyclock=1*/
   /* synthesis syn_keep=1 */;

FD u0(.C(clk),.D(~gen_clk),.Q(gen_clk));
FD ud(.C(clk),.D(gen_clk),.Q(dout));
FD uc(.D(din_r),.Q(dout_r),.C(gen_clk));

endmodule
```

VHDL Syntax and Example

attribute syn_disable_purifyclock of *object*:*objectType* **is true ;**

Where *object* is a signal.

Use the syn_keep attribute on the nets along with this attribute to ensure that net names are preserved.

Copyright © 2013 Synopsys, Inc.

syn_dspstyle

Attribute

Determines whether the object is mapped to a technology-specific DSP component.

Vendor	Technologies	Tool
Achronix	Speedster22iHD	Synplify Pro
Xilinx	Spartan-6, Spartan-3A-DSP Virtex-7 and earlier families	All

The syn_dspstyle options for designs with DSPs vary, according to the technology family selected. See the following for specific details:

- syn_dspstyle for Achronix, on page 1266
  - Achronix syn_dspstyle Syntax, on page 1266
  - Achronix Verilog Example, on page 1267
  - Achronix VHDL Example, on page 1268
- syn_dspstyle for Xilinx, on page 1269
  - Xilinx syn_dspstyle Syntax, on page 1270
  - Xilinx Constraint Examples, on page 1271
  - Xilinx Verilog Example, on page 1271
  - Xilinx VHDL Example, on page 1271
  - Effect of Using syn_dspstyle in a Xilinx Design, on page 1272
  - syn_dspstyle in Xilinx SIMD Mode, on page 1275

Copyright © 2013 Synopsys, Inc.

syn_dspstyle Values

Value	Description
logic	Maps the object to logic instead of the DSP resources.
dsp	*Achronix* Maps the specified object to BMACC56. See *syn_dspstyle for Achronix*, on page 1266 for a description of how different objects are handled.
dsp48 (Default)	*Xilinx.* Maps the specified object to a DSP48. See *syn_dspstyle for Xilinx*, on page 1269 for a description of how different objects are handled.
simd	*Xilinx Virtex 5 and later* Specifies the DSP configuration available in Single Instruction Multiple Data (SIMD) mode, and leaves it to the synthesis software to determine how to combine adders and pack them into one DSP slice. See *syn_dspstyle in Xilinx SIMD Mode*, on page 1275 for details.
simd:*i*	*Xilinx Virtex 5 and later* Specifies the DSP configuration available in SIMD mode. i is an index value that is a positive integer less than or equal to 24. Specify an index value to pair and pack multiple adders into the same DSP slice. You can pack the following components: • Two adders or subtractors with the same index value and functionality, with an input size less than or equal to 24 bits • Four adders, subtractors, or multipliers with the same index value and functionality, with an input size equal to or less than 12 bits. See *syn_dspstyle in Xilinx SIMD Mode*, on page 1275 for details.

Copyright © 2013 Synopsys, Inc.

syn_dspstyle for Achronix

For Achronix designs, the value of this attribute is either logic or dsp. The object you apply the attribute to affects what gets mapped into the DSP resource. The following table provides details:

Object	Behavior
Multiplier	Setting syn_dspstyle to dsp maps the multiplier logic structures into the BMACC56. You can apply this attribute to multipliers, which are packed into the BMACC56 component. By default, the tool implements BMACC56 when the input width of the multiplier is greater than 2-bits wide. When you set the attribute value to logic on the multiplier logical structures prevents the register from being mapped into the BMACC56 component.
Register	Setting syn_dspstyle to dsp maps the multiplier logic structures into the BMACC56. You can apply this attribute to modules or registers, which are packed into the BMACC56 component. By default, the tool implements BMACC56 when the input width of the multiplier is greater than 2-bits wide. When you set the attribute value to logic on the multiplier logical structures prevents the register from being mapped into the BMACC56 component.
View, Module, Architecture	Setting the attribute on a view, module, or architecture applies the attribute to the entire view. This attribute can be used globally.

Achronix syn_dspstyle Syntax

Global Support	Objects
Yes	Multipliers, registers, views, modules, architectures See *syn_dspstyle for Achronix*, on page 1266 for details.

The following table shows how to specify the attribute in different files:

FDC	**define_attribute {*object*} syn_dspstyle {dsp | logic}** **define_global_attribute syn_dspstyle {dsp | logic}**	
Verilog	*object* **/* synthesis syn_dspstyle = "dsp | logic" */ ;**	Achronix Verilog Example
VHDL	**attribute syn_dspstyle of** *object* **:** *objectType* **is dsp | logic;**	Achronix VHDL Example

Achronix Verilog Example

```
module test (clk,reset,enable,a,b,q);
input clk;
input reset;
input enable;
input signed [27:0] a;
input signed[27:0] b;
reg signed [27:0] a_reg;
reg signed[27:0] b_reg;
output signed [55:0] q;
wire  signed  [55:0] q1;
reg   signed [55:0] q/*synthesis syn_dspstyle = logic*/;

assign q1 = a_reg * b_reg;  // Multiply

always @(posedge clk)
begin
   if (reset)
    begin
      a_reg <= 0;
      b_reg <= 0;
      q <= 0;
    end
   else if (enable)
    begin
      a_reg <= a;
      b_reg <= b;
      q  <= q1 + q; // Accumulate
    end
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

Achronix VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity test is
generic (widtha : integer := 2;
         widthb : integer := 2;
         width_out : integer := 4
         );
port (clk : in std_logic;
      ina : in std_logic_vector(widtha-1 downto 0);
      inb : in std_logic_vector(widthb-1 downto 0);
      dout : out std_logic_vector(width_out-1 downto 0)
   );
end entity test;

architecture arc of test is
   signal dout_t : signed (width_out-1 downto 0);
   attribute syn_dspstyle : string ;
   attribute syn_dspstyle of dout_t : signal is "dsp" ;
begin

dout_t <= signed(ina) * signed(inb);

process (clk)
begin
   if(clk'event and clk='1')then
      dout <= CONV_STD_LOGIC_VECTOR(dout_t,4);
   end if;
end process;

end arc;
```

Copyright © 2013 Synopsys, Inc.

syn_dspstyle for Xilinx

This attribute determines if the object to which the attribute is applied is mapped to a Xilinx DSP48 component. The value of this attribute is either logic or dsp48. The object to which you apply this attribute determines what gets mapped into the DSP48.

Object	Behavior
Multiplier	By default, multipliers are mapped to DSP48, so setting syn_dspstyle to dsp48 on a multiplier is redundant. When a multiplier is mapped, the mapper also packs the adder/subtractor driven by the multiplier into the same DSP48 if possible. If your design has constants driving multiplier logic, the tool could perform constant propagation before DSP48 packing. Use syn_keep to prevent constant propagation optimizations. When you set the attribute value to logic, the mapper uses logic to implement the multiplier.
Adder, Subtractor, Counter	By default an adder/subtractor without a multiplier feeding it or a counter is mapped to logic. You do not need to explicitly set syn_dspstyle=logic on an adder/subtractor unless you want to decouple the mult-add or mult-subtract structure and map the multiplier into the DSP48 without the adder/subtractor. When the attribute is set to dsp48, the adder/subtractor or counter is mapped into the DSP48. If your design has constants driving multiplier logic, the tool could perform constant propagation before DSP48 packing. Use the syn_keep directive to prevent any constant propagation optimizations. By default, only one adder is packed into a single DSP48 slice. The Xilinx DSP structure supports two 24-bit adders and four 12-bit adders packed into a single DSP48 slice. For details, see *syn_dspstyle in Xilinx SIMD Mode*, on page 1275. By default, the tool automatically maps counters with a large bit size that are not timing-critical to DSP48s. If you want to map these counters to logic, you must explicitly set syn_dspstyle to logic.

Copyright © 2013 Synopsys, Inc.

Object	Behavior
Register	When a register is associated with an adder/subtractor or multiplier, the mapper determines if the register needs to be mapped into the DSP48. Setting syn_dspstyle to dsp48 has no effect. Setting syn_dspstyle to logic on a register prevents the register from being mapped into the DSP48. If you do not want to map a register to a DSP48, set syn_dspstyle to logic for the register. Note that when the attribute is attached to a register, it only applies to the register, not to the associated multipliers, adders, or other objects. You can set the SIMD values on a view.
View, Module, Architecture	Setting the attribute on a view, module, or architecture applies the attribute to the entire view. You can set the SIMD values on a view.

Xilinx syn_dspstyle Syntax

Global Support	Valid Objects
Yes	Multipliers, adders, subtractors, counters, registers, views, modules, architectures. See *syn_dspstyle for Xilinx*, on page 1269 for details.

This table shows how to specify the attribute in different files:

FDC	**define_attribute {***object***} syn_dspstyle {dsp48 | logic}** **define_global_attribute syn_dspstyle {dsp48 | logic}**	Xilinx Constraint Examples
Verilog	*object* **/* synthesis syn_dspstyle = "dsp48 | logic" */ ;**	Xilinx Verilog Example
VHDL	**attribute syn_dspstyle of** *object* **:** *objectType* **is dsp48 | logic;**	Xilinx VHDL Example

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Xilinx Constraint Examples

	Enabled	Object Type	Object	Attribute	Value	Val Type
1	✔	<any>	i:P_1[16:0]	syn_dspstyle	dsp48	string
2	✔	<any>	i:un1_b_d[15:0]	syn_dspstyle	logic	string

define_global_attribute {syn_dspstyle} {simd}

define_attribute {reg1} syn_dspstyle {dsp48}

define_attribute {my_add1} syn_dspstyle {simd:1}

define_attribute {v:work.mult} syn_dspstyle {logic}

Xilinx Verilog Example

```
module dsp_style (clk,A,B,PC,P);
input clk;
input [7:0] A,B,PC;
output reg [16:0] P;
reg [7:0]a_d,b_d;
reg [16:0] m /* synthesis syn_dspstyle = "logic" */;

always @(posedge clk) begin
   a_d <= A;
   b_d <= B;
   m   <= a_d * b_d;
   P   <= m + PC;
end

endmodule
```

See *Example 1 – Specifying SIMD*, on page 1276 and *Example 2 – Specifying a SIMD Index Value*, on page 1278 for Verilog examples of the attribute specified with SIMD mode values.

Xilinx VHDL Example

```
library ieee ;
use ieee.std_logic_1164.all ;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity dsp_style is
port (clk : in std_logic ;
   A : in std_logic_vector(7 downto 0) ;
   B : in std_logic_vector(7 downto 0) ;
   PC : in std_logic_vector(7 downto 0) ;
   P : out std_logic_vector(15 downto 0));
end dsp_style ;

architecture rtl of dsp_style is
signal m : std_logic_vector(15 downto 0) ;
attribute syn_dspstyle : string ;
attribute syn_dspstyle of m : signal is "logic" ;
begin
   process(clk)
   begin
   if (clk'event and clk = '1') then
      m <= A * B;
      P <= m + PC;
   end if ;
   end process ;
end rtl ;
```

Effect of Using syn_dspstyle in a Xilinx Design

In the example below, the syn_dspstyle attribute is set to logic globally on the module, so all four multipliers in the design are implemented as logic.

FDC	`define_global_attribute  syn_dspstyle {logic}`
Verilog	`module dsp (a,b,c,d,e,f,g,h,r1,r2)` `/*synthesis syn_dspstyle="logic"*/;`

```
module dsp(a,b,c,d,e,f,g,h,r1,r2) /* synthesis
syn_dspstyle="logic"*/;
input [7:0] a,b,c,d,e,f,g,h;
output [15:0] r1;
output [15:0] r2;
wire [15:0] temp1;
wire [15:0] temp2;
wire [15:0] temp3;
wire [15:0] temp4;
assign temp1 = a*b;
assign temp2 = c*d;
assign temp3 = e*f;
assign temp4 = g*h;
assign r1 = temp1 + temp2;
assign r2 = temp3 + temp4; endmodule
```

syn_dspstyle=dsp48

The following figure shows a multiplier, adder, and registers implemented in a DSP48 block, the default setting:

Verilog	`reg [16:0] m /* synthesis syn_dspstyle = “dsp48”*/;`
VHDL	`attribute syn_dspstyle of m : signal is “dsp48”;`

Copyright © 2013 Synopsys, Inc.

syn_dspstyle=logic

The next figure shows how the multipliers and adders are implemented when the attribute is set to logic:

Verilog	`reg [16:0] m /* synthesis syn_dspstyle = "logic"*/;`
VHDL	`attribute syn_dspstyle of m : signal is "logic";`

Mixed Values for syn_dspstyle

The following figure shows the implementation when syn_dspstyle is set to logic for the multiplier and dsp48 for the adder:

Verilog	`output reg [16:0] P /* synthesis syn_dspstyle = "dsp48"*/;` `reg [16:0] m /* synthesis syn_dspstyle = "logic"*/;`
VHDL	`attribute syn_dspstyle of P : signal is "dsp48";` `attribute syn_dspstyle of m : signal is "logic";`

syn_dspstyle in Xilinx SIMD Mode

Single Instruction Multiple Data (SIMD) mode is supported for newer Xilinx devices. By default, only one adder is packed into a single DSP slice. In SIMD mode, the Xilinx DSP structure can support two 24-bit adders and four 12-bit adders in one DSP48 slice. To implement this DSP configuration, use the syn_dspstyle attribute with one of the SIMD mode values, as described in *syn_dspstyle Values*, on page 1265. There are some limitations to DSP packing in SIMD mode; see *Limitations with SIMD Mode Packing*, on page 1276 for details.

Copyright © 2013 Synopsys, Inc.

The following Verilog syntax shows the same index set on two adders to map them into one DSP slice:

```
wire [9:0] my_add1 /* synthesis syn_dspstyle = simd:1 */;
wire [9:0] my_add2 /* synthesis syn_dspstyle = simd:1 */;
```

See the examples below for more detail.

Limitations with SIMD Mode Packing

DSP-packing in SIMD mode has the following limitations:

- Accumulators are packed only if they have the same reset, enable, and clock pins for the output registers.
- Add-Sub functions are packed only if they have the same select pin for dynamically choosing the adder or subtractor.
- Add-Mux functions are not supported with SIMD mode.

Example 1 – Specifying SIMD

The following example shows four 12-bit adders specified with SIMD mapped to one DSP48 block.

```
module test(a_ip, b_ip, c_ip, d_ip, e_ip, f_ip, g_ip, h_ip,
dout_op0, dout_op1, dout_op2, dout_op3);

//// Input declaration
input  [10 : 0] a_ip  ;
input  [10 : 0] b_ip  ;
input  [10 : 0] c_ip  ;
input  [10 : 0] d_ip  ;
input  [10 : 0] e_ip  ;
input  [10 : 0] f_ip  ;
input  [10 : 0] g_ip  ;
input  [10 : 0] h_ip  ;

//// Output declaration
output  [11 : 0] dout_op0/* synthesis syn_dspstyle="simd" */;
output  [11 : 0] dout_op1/* synthesis syn_dspstyle="simd" */;
output  [11 : 0] dout_op2/* synthesis syn_dspstyle="simd" */;
output  [11 : 0] dout_op3/* synthesis syn_dspstyle="simd" */;

assign dout_op0 = a_ip + b_ip ;
```

```
assign dout_op1 = c_ip + d_ip ;
assign dout_op2 = e_ip + f_ip ;
assign dout_op3 = g_ip + h_ip ;

endmodule
```

Copyright © 2013 Synopsys, Inc.

Example 2 – Specifying a SIMD Index Value

The following example shows two 24-bit adders specified with a SIMD index value mapped to one DSP48 block.

```
module test(clk,a_ip,b_ip,c_ip,d_ip,c_op0,c_op1);

//// Input declaration
input clk;
input signed [23 : 0] a_ip;
input signed [23 : 0] b_ip;
input signed [23 : 0] c_ip;
input signed [23 : 0] d_ip;

///Output declaration
output signed [23 : 0] c_op0/
   * synthesis syn_dspstyle = "simd:1" */;
output signed [23 : 0] c_op1
   /* synthesis syn_dspstyle = "simd:1" */;

assign c_op0 = a_ip  - b_ip ;
assign c_op1 = c_ip  - d_ip ;

endmodule
```

Copyright © 2013 Synopsys, Inc.

syn_edif_bit_format

Attribute; Xilinx. Global character formatting attribute to use with vector (bus) port names to control the naming style in the EDIF output file. The first argument (%n, %d, or %u) controls the case of the names, while the second argument (%i) controls the appearance of the vector range. The default preserves the character case of names as they appear in source code. To change the character formatting of scalar ports, see the syn_edif_scalar_format attribute.

Use this attribute to control the following:

- Up-shifting (%u) or down-shifting (%d) the character case of entire names; %n means leave case as is. The following is an example that changes all characters to upper-case.

Verilog	input[31:0] AbCd;
Synthesis Attribute	define_global_attribute syn_edif_bit_format {%u(%i)} Shifts names to upper case (`%u`) and uses parentheses as bus delimiters (`(%i)`).
EDIF Output	Uppercase name and parentheses used as bus delimiters: (port (array (rename abcd "ABCD(31:0)") 32) (direction INPUT))

- Appending the vector range to the end of names.

Verilog	input[31:0] AbCd;
Synthesis Attribute	define_global_attribute syn_edif_bit_format {%n_%i} Appends the vector range (`_%i`).
EDIF Output	Vector ranges are now appended: (port (array (rename abcd "AbCd_31:0") 32) (direction INPUT))

- Appending additional characters to the end of names.

Verilog	input[31:0] AbCd;
Synthesis Attribute	define_global_attribute syn_edif_bit_format {%n_mydesign(%i)} Appends additional character (`_mydesign`) and uses parentheses as bus delimiters (`(%i)`).
EDIF Output	Appends characters and uses parentheses as bus delimiters: (port (array (rename abcd "AbCd_mydesign(31:0)") 32) (direction INPUT))

- Converting the default bus delimiters to another representation (parentheses, square brackets or angle brackets). By *default*, Verilog designs produce square bracket delimiters in EDIF, and VHDL designs produce parenthesis delimiters in EDIF.

Verilog	input[31:0] AbCd;
Synthesis Attribute	define_global_attribute syn_edif_bit_format {%n<%i>} Specifies conversion of bus delimiters to angle brackets (`<%i>`).
EDIF Output	Uses angle brackets as bus delimiters. (port (array (rename abcd "AbCd<31:0>") 32) (direction INPUT))

Constraint File Syntax and Examples

This is the constraint file syntax for the global attribute:

define_global_attribute syn_edif_bit_format {%n|%u|%d[*char*]%i[*char*]}

For example:

```
define_global_attribute syn_edif_bit_format {%n<%i>}
```


Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Example

object /* **synthesis syn_edif_bit_format = "%n | %u | %d** [*char*] **%i** [*char*]**"** */ **;**

The following example shows how to set the bus format for a Verilog module. In the EDIF output file, the case of the name will be unchanged (`%n`) and the bus delimiters will be angle brackets (`<%i>`):

```
module pci_lc_i (...)
   /* synthesis syn_black_box syn_edif_bit_format = "%n<%i>" */;
```

VHDL Syntax and Example

attribute syn_edif_bit_format of *object* **:** *objectType* **is "%n | %u | %d** [*ch*] **%i** [*ch*]**" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

The following is an example for syn_edif_bit_format:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library synplify;
use synplify.attributes.all;

entity addern is
generic (num_bits : integer := 8) ;
   port (a,b : in  std_logic_vector(num_bits -1 downto 0);
         result : out std_logic_vector(num_bits -1 downto 0) );
end addern;

architecture behave of addern is : component is string ;
attribute syn_edif_bit_format
attribute syn_edif_bit_format of behave :
   architecture is "%n<%i>";
begin
   result <= a + b;
end;
```

Copyright © 2013 Synopsys, Inc.

The following table shows the original and new EDIF formats:

Original EDIF Format	(interface (port (array (rename a "a(7:0)") 8) (direction INPUT)) (port (array (rename b "b(7:0)") 8) (direction INPUT)) (port (array (rename result "result(7:0)") 8) (direction OUTPUT))) (net (rename a_0 "a(0)") (joined (portRef (member a 7)) (portRef I (instanceRef a_ibuf_0)))) (net (rename a_1 "a<1>") (joined
New EDIF Format	(interface (port (array (rename a "a<7:0>") 8) (direction INPUT)) (port (array (rename b "b<7:0>") 8) (direction INPUT)) (port (array (rename result "result<7:0>") 8) (direction OUTPUT))) (net (rename a_0 "a<0>") (joined (portRef (member a 7)) (portRef I (instanceRef a_ibuf_0)))) (net (rename a_1 "a<1>") (joined

The following example shows how to specify angle brackets as the bus delimiters for the VHDL black-box component pci_lc_i:

```
component pci_lc_i
   port (
         .
         .
         .
         );
end component;
attribute syn_edif_bit_format : component is string ;
attribute syn_edif_bit_format of pci_lc_i : component is "%n<%i>";
```

The following example shows how to shift the name to uppercase and specify square brackets as the bus delimiters for the VHDL black-box architecture rtl:

```
architecture rtl of pci_top is
attribute syn_edif_bit_format : architecture is string ;
attribute syn_edif_bit_format of rtl : architecture is "%u[%i]";
```


Copyright © 2013 Synopsys, Inc.

syn_edif_name_length

Attribute; Altera. Global truncation attribute that determines the length of port names written to the output file for Altera designs. If your target place-and-route tool is Quartus II, the length of the port names does not matter.

You specify this attribute globally on the top-level module or architecture. The value of the attribute can be either restricted or unrestricted. The restricted setting (the default) limits the port names to 30 characters. Use this setting with Max+Plus II. If you set the attribute to unrestricted, the port names are not truncated. Use this value if you are targeting Quartus II as the place-and-route tool.

Constraint File Syntax

The constraint file syntax for the global attribute is as follows:

define_global_attribute syn_edif_name_length {restricted|unrestricted}

Verilog Syntax and Example

object **/* synthesis syn_edif_name_length = "restricted|unrestricted" */ ;**

Attach the attribute to the top-level module. The following is an example:

```
module test(in1, in2 , out )
      /* synthesis syn_edif_name_length = "restricted" */;
input in1, in2;
output out;
testbox inst1(.in1(in1), .in2(in2),
   .abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz(out));
endmodule
```

VHDL Syntax and Example

attribute syn_edif_name_length of *object*:*objectType* **is "restricted | unrestricted" ;**

Attach the attribute to the top-level entity, as shown in this example:

```
architecture beh of test is
component testbox is port(
in1, in2 : in std_logic;
abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz : out
   std_logic);
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of testbox : component is true;
attribute syn_edif_name_length : string;
attribute syn_edif_name_length of beh :
   architecture is "Unrestricted";
begin
u0: testbox port map (in1 => in1, in2 => in2,
   abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz
   => out1);
end beh;
```

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Copyright © 2013 Synopsys, Inc.

syn_edif_scalar_format

Attribute.

Helps change the case of the object name in an edif file automatically.

Vendor	Technology
Xilinx	Alls

syn_edif_scalar_format Values

Value	Description
%n (Default)	Preserves case in the names of edif objects
%u	Makes the names of edif objects to uppercase
%d	Makes the names of edif objects to lowercase

Description

Global character formatting attribute to use with scalar port names to up-shift (%u) or down-shift (%d) the character case of the name in the EDIF output file. The default (syn_edif_scalar_format {%n}) preserves the character case of the names as they appear in source code. To change the character formatting and style of vector (bus) signals and ports, see syn_edif_bit_format.

syn_edif_scalar_format Syntax

Global Support	Object
Yes	Verilog module or VHDL architecture

Copyright © 2013 Synopsys, Inc.

The following table summarizes the syntax in different files:

FDC	`define_global_attribute syn_edif_scalar_format {%u|%d|%n}`
Verilog	`object /*synthesis syn_edif_scalar_format = " value" */;`
VHDL	`attribute syn_edif_scalar_format of object : objectType : architecture is " value";`

Constraint Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	☑	global	<global>	syn_edif_scalar_format	%u	string	Format scaler names	

Verilog Example

```
module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format =
" %u" */;
input aBcd,clk;
output reg q;
always@(posedge clk)
q<=aBcd;
endmodule
```

VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;
entity d_ff_srss is
port (
aBcd,clk: in bit;
q : out bit);
end d_ff_srss;
architecture d_ff of d_ff_srss is
attribute syn_edif_scalar_format : string ;
attribute syn_edif_scalar_format of d_ff : architecture is "%u";
begin
process(clk)
begin
```


Copyright © 2013 Synopsys, Inc.

```
if clk'event and clk='1' then
q <= aBcd;
end if;
end process;
end d_ff;
```

Effect of Using syn_edif_scalar_format

The following table shows a Xilinx design before applying the syn_edif_scalar_format attribute:

Verilog

```
module edif_case(aBcd,clk,q)
/*synthesis syn_edif_scalar_format=" %n"*/;
```

VHDL

```
attribute syn_edif_scalar_format : string ;
attribute syn_edif_scalar_format of d_ff : architecture is "
%n";
```

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell edif_case (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port aBcd (direction INPUT))
           (port clk (direction INPUT)

           (port q (direction OUTPUT))
          (net aBcd_c (joined
           (portRef O (instanceRef aBcd_ibuf))
           (portRef D (instanceRef qZ0))
          ))
          (net (rename abcd "aBcd") (joined
           (portRef aBcd)
           (portRef I (instanceRef aBcd_ibuf))
          ))
          (net clk_c (joined
           (portRef O (instanceRef clk_ibuf))
           (portRef C (instanceRef qZ0))
```


Copyright © 2013 Synopsys, Inc.

```
))
          (net clk (joined
           (portRef clk)
           (portRef I (instanceRef clk_ibuf_iso))
          ))
          (net q_c (joined
           (portRef Q (instanceRef qZ0))
           (portRef I (instanceRef q_obuf))
          ))
          (net q (joined
           (portRef O (instanceRef q_obuf))
           (portRef q)
          ))
          (net (rename clk_ibuf_isoZ0 "clk_ibuf_iso") (joined
           (portRef O (instanceRef clk_ibuf_iso))
           (portRef I (instanceRef clk_ibuf))
          ))
         )
        (property mapper_option (string ""))
)
```

The following table shows a Xilinx design after applying the syn_edif_scalar_format attribute with the `%u` value:

Verilog

```
module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format="
%u"*/;
```


Copyright © 2013 Synopsys, Inc.

VHDL

```
attribute syn_edif_scalar_format : string ;
attribute syn_edif_scalar_format of d_ff : architecture is "
%u";
```

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell edif_case (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port ABCD (direction INPUT))
           (port CLK (direction INPUT))
           (port Q (direction OUTPUT))

          (net (rename aBcd_c "ABCD_C") (joined
           (portRef O (instanceRef aBcd_ibuf))
           (portRef D (instanceRef qZ0))
          ))
          (net (rename abcd "ABCD") (joined
           (portRef ABCD)
           (portRef I (instanceRef aBcd_ibuf))
          ))
          (net (rename clk_c "CLK_C") (joined
           (portRef O (instanceRef clk_ibuf))
           (portRef C (instanceRef qZ0))
          ))
```

```
(net (rename clk "CLK") (joined
           (portRef CLK)
           (portRef I (instanceRef clk_ibuf_iso))
          ))
          (net (rename q_c "Q_C") (joined
           (portRef Q (instanceRef qZ0))
           (portRef I (instanceRef q_obuf))
          ))
          (net (rename q "Q") (joined
           (portRef O (instanceRef q_obuf))
           (portRef Q)
          ))
          (net (rename clk_ibuf_isoZ0 "CLK_IBUF_ISO") (joined
           (portRef O (instanceRef clk_ibuf_iso))
           (portRef I (instanceRef clk_ibuf))
          ))
         )
        (property mapper_option (string ""))
       )
```

The following table shows a Xilinx design after applying the syn_edif_scalar_format attribute with the `%d` value:

Copyright © 2013 Synopsys, Inc.

Verilog

```
module edif_case(aBcd,clk,q)
/*synthesis syn_edif_scalar_format=" %d"*/;
```

VHDL

```
attribute syn_edif_scalar_format : string ;
attribute syn_edif_scalar_format of d_ff : architecture is
" %d";
```

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell edif_case (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port abcd (direction INPUT))
           (port clk (direction INPUT))
           (port q (direction OUTPUT))

          (net (rename aBcd_c "abcd_c") (joined
           (portRef O (instanceRef aBcd_ibuf))
           (portRef D (instanceRef qZ0))
          ))
          (net abcd (joined
           (portRef abcd)
           (portRef I (instanceRef aBcd_ibuf))
          ))
          (net clk_c (joined
           (portRef O (instanceRef clk_ibuf))
           (portRef C (instanceRef qZ0))
          ))
```


Copyright © 2013 Synopsys, Inc.

```
(net clk (joined
           (portRef clk)
           (portRef I (instanceRef clk_ibuf_iso))
          ))
          (net q_c (joined
           (portRef Q (instanceRef qZ0))
           (portRef I (instanceRef q_obuf))
          ))
          (net q (joined
           (portRef O (instanceRef q_obuf))
           (portRef q)
          ))
          (net (rename clk_ibuf_isoZ0 "clk_ibuf_iso") (joined
           (portRef O (instanceRef clk_ibuf_iso))
           (portRef I (instanceRef clk_ibuf))
          ))
         )
        (property mapper_option (string ""))
       )
```

Global Support

If a design is has more than one edif netlist, apply this attribute only on the top level netlist. It has the same impact as when it is applied on individual netlists. For example, if two modules have been instantiated, the attribute should be applied only on the top-level module.

Verilog Example

```
module mux(out, a, b, sel); // mux
output  out;
input  a, b;
input sel;
assign out = sel ? a : b;
endmodule
```

```
module reg8(q, data, clk, rst); //  register
output  q;
input data;
input clk, rst;
reg  q;
always @(posedge clk or posedge rst)
begin
if (rst)
q = 0;
else
q = data;
end
endmodule

module top1(q1, a, b, sel, clk, rst)/*synthesis
syn_edif_scalar_format="%u"*/;
output q1;
input a, b;
input sel, clk, rst;
wire  mux_out;
mux mux_1 (mux_out, a, b, sel);
reg8 reg8_1 (q1, mux_out, clk, rst);
endmodule
```

The following table shows a Xilinx design with the `syn_edif_scalar_format` attribute applied to the top-level module.

Copyright © 2013 Synopsys, Inc.

FDC File	`define_global_attribute {syn_edif_scalar_format} {%u}`
FDC File	module top1(q1, a, b, sel, clk, rst)`/*synthesis syn_edif_scalar_format="%u"*/;`

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell reg8 (cellType GENERIC)
       (view netlist (viewType NETLIST)
         (interface
           (port RST_C (direction INPUT))
           (port CLK_C (direction INPUT))
           (port MUX_OUT (direction INPUT))
           (port Q1_C (direction OUTPUT))
         )
         (contents
          (instance q (viewRef PRIM (cellRef FDC (libraryRef UNILIB)))
           )
```

```
(net (rename q1_c "Q1_C") (joined
           (portRef Q (instanceRef q))
           (portRef Q1_C)
          ))
          (net (rename mux_out "MUX_OUT") (joined
           (portRef MUX_OUT)
           (portRef D (instanceRef q))
          ))
          (net (rename clk_c "CLK_C") (joined
           (portRef CLK_C)
           (portRef C (instanceRef q))
          ))
          (net (rename rst_c "RST_C") (joined
           (portRef RST_C)
           (portRef CLR (instanceRef q))
          ))
         )
       )
    )
```


Copyright © 2013 Synopsys, Inc.

```
(cell mux (cellType GENERIC)
       (view netlist (viewType NETLIST)
         (interface
           (port MUX_OUT (direction OUTPUT))
           (port SEL_C (direction INPUT))
           (port B_C (direction INPUT))
           (port A_C (direction INPUT))
         )
         (contents
          (instance out (viewRef PRIM (cellRef LUT3_L (libraryRef
VIRTEX)))
           (property INIT (string "8'hAC"))
          )
          (net (rename a_c "A_C") (joined
           (portRef A_C)
           (portRef I0 (instanceRef out))
          ))
          (net (rename b_c "B_C") (joined
           (portRef B_C)
           (portRef I1 (instanceRef out))
          ))
          (net (rename sel_c "SEL_C") (joined
           (portRef SEL_C)
           (portRef I2 (instanceRef out))
          ))
          (net (rename mux_out "MUX_OUT") (joined
           (portRef LO (instanceRef out))
           (portRef MUX_OUT)
          ))
         )
       )
    )
```

```
(cell top1 (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port Q1 (direction OUTPUT))
           (port A (direction INPUT))
           (port B (direction INPUT))
           (port SEL (direction INPUT))
           (port CLK (direction INPUT)
 )
           (port RST (direction INPUT))
         )
         (contents
          (instance clk_ibuf (viewRef PRIM (cellRef BUFG (libraryRef
VIRTEX)))          )
          (instance clk_ibuf_iso (viewRef PRIM (cellRef IBUFG
(libraryRef VIRTEX)))
          )
          (instance q1_obuf (viewRef PRIM (cellRef OBUF (libraryRef
VIRTEX)))
          )
          (instance rst_ibuf (viewRef PRIM (cellRef IBUF (libraryRef
VIRTEX)))
          )
          (instance sel_ibuf (viewRef PRIM (cellRef IBUF (libraryRef
VIRTEX)))
          )
          (instance b_ibuf (viewRef PRIM (cellRef IBUF (libraryRef
VIRTEX)))
          )
          (instance a_ibuf (viewRef PRIM (cellRef IBUF (libraryRef
VIRTEX)))
          )
          (instance mux_1 (viewRef netlist (cellRef mux))
          )
          (instance reg8_1 (viewRef netlist (cellRef reg8))
          )
```


Copyright © 2013 Synopsys, Inc.

```
(net (rename mux_out "MUX_OUT") (joined
          (portRef MUX_OUT (instanceRef mux_1))
          (portRef MUX_OUT (instanceRef reg8_1))
         ))
         (net (rename a_c "A_C") (joined
          (portRef O (instanceRef a_ibuf))
          (portRef A_C (instanceRef mux_1))
         ))
         (net (rename a "A") (joined
          (portRef A)
          (portRef I (instanceRef a_ibuf))
         ))

         (net (rename b_c "B_C") (joined
          (portRef O (instanceRef b_ibuf))
```

```
(portRef B_C (instanceRef mux_1))
         ))
         (net (rename b "B") (joined
          (portRef B)
          (portRef I (instanceRef b_ibuf))
         ))
         (net (rename sel_c "SEL_C") (joined
          (portRef O (instanceRef sel_ibuf))
          (portRef SEL_C (instanceRef mux_1))
         ))
         (net (rename sel "SEL") (joined
          (portRef SEL)
          (portRef I (instanceRef sel_ibuf))
         ))
         (net (rename clk_c "CLK_C") (joined
          (portRef O (instanceRef clk_ibuf))
          (portRef CLK_C (instanceRef reg8_1))
         ))
         (net (rename clk "CLK") (joined
          (portRef CLK)
          (portRef I (instanceRef clk_ibuf_iso))
         ))
         (net (rename rst_c "RST_C") (joined
          (portRef O (instanceRef rst_ibuf))
          (portRef RST_C (instanceRef reg8_1))
         ))
```


Copyright © 2013 Synopsys, Inc.

```
(net (rename rst "RST") (joined
          (portRef RST)
          (portRef I (instanceRef rst_ibuf))
         ))
         (net (rename q1_c "Q1_C") (joined
          (portRef Q1_C (instanceRef reg8_1))
          (portRef I (instanceRef q1_obuf))
         ))
         (net (rename q1 "Q1") (joined
          (portRef O (instanceRef q1_obuf))
          (portRef Q1)
         ))
         (net (rename clk_ibuf_isoZ0 "CLK_IBUF_ISO") (joined
          (portRef O (instanceRef clk_ibuf_iso))
          (portRef I (instanceRef clk_ibuf))
         ))
        )
       (property mapper_option (string ""))
```

syn_encoding

Attribute

Overrides the default FSM Compiler encoding for a state machine and applies the specified encoding.

Vendor	Devices
Achronix	Speedster
Altera	Stratix V,Cyclone V ,Older devices
Lattice	ECP3,ECP4 ,Older devices
Microsemi	ProASIC3, Fusion, SmartFusion4,Older devices
Xilinx	Virtex 6, Spartan6,Older devices

syn_encoding Values

The default is that the tool automatically picks an encoding style that results in the best performance. To ensure that a particular encoding style is used, explicitly specify that style, using the values below:

Value	Description
onehot	Only two bits of the state register change (one goes to 0, one goes to 1) and only one of the state registers is hot (driven by 1) at a time. For example: 0001, 0010, 0100, 1000 Because onehot is not a simple encoding (more than one bit can be set), the value must be decoded to determine the state. This encoding style can be slower than a gray style if you have a large output decoder following a state machine.

Copyright © 2013 Synopsys, Inc.

Value	Description
gray	More than one of the state registers can be hot. The synthesis tool *attempts* to have only one bit of the state registers change at a time, but it can allow more than one bit to change, depending upon certain conditions for optimization. For example: 000, 001, 011, 010, 110 Because gray is not a simple encoding (more than one bit can be set), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.
sequential	More than one bit of the state register can be hot. The synthesis tool makes no attempt at limiting the number of bits that can change at a time. For example: 000, 001, 010, 011, 100 This is one of the smallest encoding styles, so it is often used when area is a concern. Because more than one bit can be set (1), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.
safe	This implements the state machine in the default encoding and adds reset logic to force the state machine to a known state if it reaches an invalid state. This value can be used in combination with any of the other encoding styles described above. You specify safe before the encoding style. The safe value is only valid for a state register, in conjunction with an encoding style specification. For example, if the default encoding is onehot and the state machine reaches a state where all the bits are 0, which is an invalid state, the safe value ensures that the state machine is reset to a valid state. If recovery from an invalid state is a concern, it may be appropriate to use this encoding style, in conjunction with onehot, sequential or gray, in order to force the state machine to reset. When you specify safe, the state machine can be reset from an unknown state to its reset state.
original	This respects the encoding you set, but the software still does state machine and reachability analysis.

You can specify multiple values. This snippet uses safe,gray. The encoding style for register OUT is set to gray, but if the state machine reaches an invalid state the synthesis tool will reset the values to a valid state.

```
module prep3 (CLK, RST, IN, OUT);
input CLK, RST;
input [7:0] IN;
output [7:0] OUT;
reg [7:0] OUT;
reg [7:0] current_state /* synthesis syn_encoding="safe,gray" */;

// Other code
```

Description

This attribute takes effect only when FSM Compiler is enabled. It overrides the default FSM Compiler encoding for a state machine. For the specified encoding to take effect, the design must contain state machines that have been inferred by the FSM Compiler. Setting this attribute when syn_state_machine is set to 0 will not have any effect.

The default encoding style automatically assigns encoding based on the number of states in the state machine. Use the syn_encoding attribute when you want to override these defaults. You can also use syn_encoding when you want to disable the FSM Compiler globally but there are a select number of state registers in your design that you want extracted. In this case, use this attribute with the syn_state_machine directive on for just those specific registers.

The encoding specified by this attribute applies to the final mapped netlist. For other kinds of enumerated encoding, use syn_enum_encoding. See *syn_enum_encoding*, on page 1314 and *syn_encoding Compared to syn_enum_encoding*, on page 1317 for more information.

Encoding Style Implementation

The encoding style is implemented during the mapping phase. A message appears when the synthesis tool extracts a state machine, for example:

```
@N: CL201 : "c:\design\..."|Trying to extract state machine for
register current_state
```

The log file reports the encoding styles used for the state machines in your design. In the Synplify Pro and Synplify Premier tools, this information is also available in the FSM Viewer.

See also the following:

Copyright © 2013 Synopsys, Inc.

- For information on enabling state machine optimization for individual modules, see *syn_state_machine*, on page 1529.
- For VHDL designs, see *syn_encoding Compared to syn_enum_encoding*, on page 1317 for comparative usage information.

Syntax Specification

Global	Object
No	Instance, register

This table shows how to specify the attribute in different files:

FDC	define_attribute {*object*} syn_encoding {*value*}	SCOPE Example
Verilog	*Object* /* synthesis syn_encoding = "*value*" */;	Verilog Example
VHDL	attribute syn_encoding of *object*: objectType is "*value*";	VHDL Example

If you specify the syn_encoding attribute in Verilog or VHDL, all instances of that FSM use the same syn_encoding value. To have unique syn_encoding values for each FSM instance, use different entities or modules, or specify the syn_encoding attribute in a constraint file.

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	fsm	i:state[3:0]	syn_encoding	gray	string	FSM encoding (onehot, sequential, gray, original, safe)

The *object* must be an instance prefixed with **i:**, as in **i:***instance*. The instance must be a sequential instance with a view name of statemachine.

Although you cannot set this attribute globally, you can define a SCOPE collection and then apply the attribute to the collection. For example:

```
define_scope_collection sm {find -hier -inst * -filter
   @inst_of==statemachine}
define_attribute {$sm} {syn_encoding} {safe}
```

Verilog Example

The object can be a register definition signals that hold the state values of state machines.

```
module fsm (clk, reset, x1, outp);
input        clk, reset, x1;
output       outp;
reg          outp;
reg    [1:0] state /* synthesis syn_encoding = "onehot" */;
parameter s1 = 2'b00; parameter s2 = 2'b01;
parameter s3 = 2'b10; parameter s4 = 2'b11;

always @(posedge clk or posedge reset)
begin
   if (reset)
      state <= s1;
   else begin
      case (state)
      s1: if (x1 == 1'b1)
         state <= s2;
      else
         state <= s3; s2: state <= s4;
   s3: state <= s4;
   s4: state <= s1;
   endcase
end
end

always @(state) begin
   case (state)
      s1: outp = 1'b1;
      s2: outp = 1'b1;
      s3: outp = 1'b0;
      s4: outp = 1'b0;
   endcase
   end
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity fsm is
   port ( x1  : in std_logic;
   reset : in std_logic;
   clk : in std_logic;
   outp : out std_logic);
end fsm;

architecture rtl of fsm is
signal state : std_logic_vector(1 downto 0);
constant s1 : std_logic_vector := "00";
constant s2 : std_logic_vector := "01";
constant s3 : std_logic_vector := "10";
constant s4 : std_logic_vector := "11";
attribute syn_encoding : string;
attribute syn_encoding of state : signal is "onehot";

begin
process (clk,reset)
   begin
   if (clk'event and clk = '1') then
      if (reset = '1') then
         state <= s1 ;
      else
         case state is
            when s1 =>
            if x1 = '1' then
               state <= s2;
            else
               state <= s3;
            end if;
            when s2 =>
               state <= s4;
            when s3 =>
               state <= s4;
            when s4 =>
               state <= s1;
         end case;
      end if;
   end if;
end process;
```

```
process (state)
begin
   case state is
      when s1 =>
         outp <= '1';
      when s2 =>
         outp <= '1';
      when s3 =>
         outp <= '0';
      when s4 =>
         outp <= '0';
      end case;
end process;
end rtl;
```

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Effect of Using syn_encoding

The following figure shows the default implementation of a state machine, with these encoding details reported:

```
Encoding state machine state [3:0] (netlist: statemachine)
original code -> new code
   00 -> 00
   01 -> 01
   10 -> 10
   11 -> 11
```

Copyright © 2013 Synopsys, Inc.

The next figure shows the state machine when the syn_encoding attribute is set to onehot, and the accompanying changes in the code:

Verilog	`reg [1:0] state /* synthesis syn_encoding = "onehot"*/;`
VHDL	`attribute syn_encoding of state : signal is "onehot";`

```
Encoding state machine state [3:0] (netlist: statemachine)
original code -> new code
  00 -> 0001
  01 -> 0010
  10 -> 0100
  11 -> 1000
```

The next figure shows the state machine when the syn_encoding attribute is set to gray:

Verilog	`reg [1:0] state /* synthesis syn_encoding = "gray"*/;`
VHDL	`attribute syn_encoding of state : signal is "gray";`

```
Encoding state machine state [3:0] (netlist: statemachine)
original code -> new code
   00 -> 00
   00 -> 01
   10 -> 11
   11 -> 10
```


Copyright © 2013 Synopsys, Inc.

syn_enum_encoding

Directive. For VHDL designs. Defines how enumerated data types are implemented. The type of implementation affects the performance and device utilization.

If FSM Compiler is enabled, this directive has no effect on the encoding styles of extracted state machines; the tool uses the values specified in the syn_encoding attribute instead. However, if you have enumerated data types and you turn off the FSM Compiler so that no state machines are extracted, the syn_enum_encoding style is implemented in the final circuit. See *syn_encoding Compared to syn_enum_encoding*, on page 1317 for more information. For step-by-step details about setting coding styles with this attribute see *Defining State Machines in VHDL*, on page 341 of the *User Guide*.

Values for syn_enum_encoding

Values for syn_enum_encoding are as follows:

- default – Automatically assigns an encoding style that results in the best performance.
- sequential – More than one bit of the state register can change at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 010, 011, 100
- onehot – Only two bits of the state register change (one goes to 0; one goes to 1) and only one of the state registers is hot (driven by a 1) at a time. For example: 0000, 0001, 0010, 0100, 1000
- gray – Only one bit of the state register changes at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 011, 010, 110
- *string* – This can be any value you define. For example: 001, 010, 101. See *Example of syn_enum_encoding for User-Defined Encoding*, on page 1318.

A message appears in the log file when you use the syn_enum_encoding directive; for example:

```
CD231: Using onehot encoding for type mytype (red="10000000")
```

Effect of Encoding Styles

The following figure provides an example of two versions of a design: one with the default encoding style, the other with the syn_enum_encoding directive overriding the default enumerated data types that define a set of eight colors.

syn_enum_encoding = “default” Based on 8 states, onhot assigned

syn_enum_encoding = “sequential”

In this example, using the default value for syn_enum_encoding, onehot is assigned because there are eight states in this design. The onehot style implements the output color as 8 bits wide and creates decode logic to convert the input sel to the output. Using sequential for syn_enum_encoding, the logic is reduced to a buffer. The size of output color is 3 bits.

See the following section for the source code used to generate the schematics above.

VHDL Syntax and Examples

attribute syn_enum_encoding of *object* **:** *objectType* **is "***value***" ;**

Copyright © 2013 Synopsys, Inc.

Where *object* is an enumerated type and *value* is one of the following: default, sequential, onehot or gray. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Here is the code used to generate the second schematic in the previous figure. (The first schematic will be generated instead, if "sequential" is replaced by "onehot" as the syn_enum_encoding value.)

```
package testpkg is
type mytype is (red, yellow, blue, green, white,
    violet, indigo, orange);
attribute syn_enum_encoding : string;
attribute syn_enum_encoding of mytype : type is "sequential";
end package testpkg;

library IEEE;
use IEEE.std_logic_1164.all;
use work.testpkg.all;

entity decoder is
   port (sel : in std_logic_vector(2 downto 0);
   color : out mytype );
end decoder;
architecture rtl of decoder is
begin
   process(sel)
   begin
      case sel is
         when "000" => color <= red;
         when "001" => color <= yellow;
         when "010" => color <= blue;
         when "011" => color <= green;
         when "100" => color <= white;
         when "101" => color <= violet;
         when "110" => color <= indigo;
         when others => color <= orange;
      end case;
   end process;
end rtl;
```

syn_enum_encoding, enum_encoding, and syn_encoding

Custom attributes are attributes that are not defined in the IEEE specifications, but which you or a tool vendor define for your own use. They provide a convenient back door in VHDL, and are used to better control the synthesis

and simulation process. enum_encoding is one of these custom attributes that is widely used to allow specific binary encodings to be attached to objects of enumerated types.

The enum_encoding attribute is declared as follows:

```
attribute enum_encoding: string;
```

This can be either written directly in your VHDL design description, or provided to you by the tool vendor in a package. Once the attribute has been declared and given a name, it can be referenced as needed in the design description:

```
type statevalue is (INIT, IDLE, READ, WRITE, ERROR);
attribute enum_encoding of statevalue: type is
   "000 001 011 010 110";
```

When this is processed by a tool that supports the enum_encoding attribute, it uses the information about the statevalue encoding. Tools that do not recognize the enum_encoding attribute ignore the encoding.

Although it is recommended that you use syn_enum_encoding, the Synopsys FPGA tools recognize enum_encoding and treat it just like syn_enum_encoding. The tool uses the specified encoding when the FSM compiler is disabled, and ignores the value when the FSM Compiler is enabled.

If enum_encoding and syn_encoding are both defined and the FSM compiler is enabled, the tool uses the value of syn_encoding. If you have both syn_enum_encoding and enum_encoding defined, the value of syn_enum_encoding prevails.

syn_encoding Compared to syn_enum_encoding

To implement a state machine with a particular encoding style when the FSM Compiler is enabled, use the syn_encoding attribute. The syn_encoding attribute affects how the technology mapper implements state machines in the final netlist. The syn_enum_encoding directive only affects how the compiler interprets the associated enumerated data types. Therefore, the encoding defined by syn_enum_encoding is *not propagated* to the implementation of the state machine. However, when FSM Compiler is disabled, the value of syn_enum_encoding is implemented in the final circuit.

Copyright © 2013 Synopsys, Inc.

Example of syn_enum_encoding for User-Defined Encoding

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
   port (clk, rst : bit;
         O : out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
   process (clk, rst)
   begin
      if rst = '1' then
         machine <= S0;
      elsif clk = '1' and clk'event then
         case machine is
            when S0 => machine <= S1;
            when S1 => machine <= S2;
            when S2 => machine <= S0;
         end case;
      end if;
   end process;

   with machine select
      O <= "001" when S0,
      "010" when S1,
      "101" when S2;
end behave;
```

syn_expand_coreloc

Attribute; When back-annotation is run after placement and routing, only I/O pad locations are written to the coreloc.sdc file by default. Use the syn_expand_coreloc attribute so that locations for the RAM and DSP blocks, as well as I/O pads are written to the coreloc.sdc file.

To enable this attribute, set syn_expand_coreloc to 1. The attribute can only be applied globally for the design.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_global_attribute syn_expand_coreloc {1}

For example:

```
define_global_attribute syn_expand_coreloc {1}
```


Copyright © 2013 Synopsys, Inc.

syn_fast_auto

Attribute.

Speeds up signal transitions on output ports.

Vendor	Technology
Xilinx	All

syn_fast_auto Values

Value	Description
"False"/ "0"	Keeps the transition time slow.
"True"/ "1"	Infers fast output buffers.

Description

The synthesis tool infers the fast output buffers OBUF_F_24, OBUFT_F_24, and IOBUF_F_24. Use the `syn_fast_auto` global attribute to force inference of slow buffers. In the HDL source code, specify `syn_fast_auto` for the top-level module or architecture.

You can override this attribute on an individual basis by using the `syn_padtype` attribute. The `syn_fast_auto` attribute has no effect on output ports specified with the `syn_padtype` attribute.

syn_fast_auto Syntax

Default	Global	Object
`"True" / "1"`	Yes	Module/Architecture

The following table summarizes the syntax in different files:

FDC	`define_global_attribute syn_fast_auto {0|1}`
Verilog	`Object /*synthesis syn_fast_auto= value 0|1 */;`
VHDL	`attribute syn_fast_auto : boolean;` `attribute syn_fast_auto of Object : Object Type is {value};`

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	global	<global>	xc_fast_auto	1	boolean	Enable automatic fast output buffer use

Verilog Example

```
module ramtest (q,d,addr,we,clk)/*synthesis syn_fast_auto=0 */;
input we,clk;
input [2:0]addr;
input [3:0]d;
output reg [3:0]q;
reg [3:0] mem [7:0];

always @(posedge clk)
begin
q <= mem[addr];
if (we == 1)
mem[addr] <= d;
end
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
```


Copyright © 2013 Synopsys, Inc.

```
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
attribute syn_fast_auto : boolean;
attribute syn_fast_auto of rtl : architecture is false;

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;
```

Effect of Using syn_fast_auto

The following table shows a Xilinx design before applying the `syn_fast_auto` attribute.

Verilog	No attribute statement.

VHDL No attribute statement.

Constraint Scope attribute definition disabled:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	global	<global>	xc_fast_auto	1	boolean	Enable automatic fast output buffer use

Check the following lines in the generated *.edf file:

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell ramtest (cellType GENERIC)
       (view rtl (viewType NETLIST)
         (
           ------------
           ------------
           ------------
           (property mapper_option (string ""))
         )
       )
     )
     (design ramtest (cellRef ramtest (libraryRef work))
       (property mapper_option (string ""))
       (property PART (string "xc5vlx20tff323-1") (owner "Xilinx")))
     )
```

The following table shows a Xilinx design after applying the `syn_fast_auto` attribute.

Verilog module ramtest (q,d,addr,we,clk)`/*synthesis syn_fast_auto=1 */;`

Copyright © 2013 Synopsys, Inc.

VHDL

```
attribute syn_fast_auto : boolean;
attribute syn_fast_auto of rtl : architecture is true;
```

Constraint

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	global	<global>	xc_fast_auto	1	boolean	Enable automatic fast output buffer use

Search for the word "syn_fast_auto" in the *.edf file.

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell ramtest (cellType GENERIC)
       (view rtl (viewType NETLIST)
         (
           ------------
           ------------
           ------------
           (property mapper_option (string ""))
           (property syn_fast_auto (integer 1))
          )
       )
    )
    (design ramtest (cellRef ramtest (libraryRef work))
      (property mapper_option (string ""))
      (property PART (string "xc5vlx20tff323-1") (owner "Xilinx")))
    )
```

The following table shows a Xilinx design with the value of `syn_fast_auto` attribute set to false.

Verilog

```
module ramtest (q,d,addr,we,clk)/*synthesis syn_fast_auto=0
*/;
```

VHDL

```
attribute syn_fast_auto : boolean;
attribute syn_fast_auto of rtl : architecture is false;
```

Constraint

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	global	<global>	xc_fast_auto	0	boolean	Enable automatic fast output buffer use

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell ramtest (cellType GENERIC)
       (view rtl (viewType NETLIST)
         (
         -----------
         -----------
         -----------
         (property mapper_option (string ""))
         (property syn_fast_auto (integer 0))
       )
    )
  )
  (design ramtest (cellRef ramtest (libraryRef work))
    (property mapper_option (string ""))
    (property PART (string "xc5vlx20tff323-1") (owner "Xilinx")))
  )
```

The 'FAST' keyword is not forward annotated to any of the Xilinx P&R files. The following files have forward annotated "SLOW" keyword attached to output ports in the design.

In the file, "....pr_1\ramtest_map.mrp", the following statement appears:

INFO:LIT:244—All the single ended outputs in this design use slew rate limited output drivers. The delay on speed critical single ended outputs can be dramatically reduced by designating them as fast outputs.

```
..\rev_1\pr_1\ramtest_map.xrpt(281): <item label="Slew&#xA;Rate" stringID="SLEW_RATE"value="SLOW" />
..\rev_1\pr_1\ramtest_pad.csv(274):R1,q[2],IOB,IO_L0P_CC_RS1_2,OUTPUT,LVCMOS25*,2,12,SLOW,,,,UNLOCATE
D,NO,NONE,
..\rev_1\pr_1\ramtest.pad(274):
R1|q[2]|IOB|IO_L0P_CC_RS1_2|OUTPUT|LVCMOS25*|2|12|SLOW||||UNLOCATED|NO|NONE|
```

```
..\rev_1\pr_1\ramtest_map.mrp(128): | q[0] | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | | | |
..\rev_1\pr_1\ramtest_pad.txt(259): |P3 |q[1] |IOB |IO_L2N_A22_2 |OUTPUT |LVCMOS25* |2 |12 |SLOW | |
| |UNLOCATED |NO |NONE |
..\rev_1\pr_1\ramtest.pad(258):P3|q[1]|IOB|IO_L2N_A22_2|OUTPUT|LVCMOS25*|2|12|SLOW||||UNLOCATED|NO|NO
NE|
```

The attribute can also be applied individually in the fdc file as follows:

```
define_attribute  {p:q[3:0]} {xc_fast_auto} {0}
```


Copyright © 2013 Synopsys, Inc.

Global Support

The `syn_fast_auto` attribute is defined globally by default as shown in the above examples and can be overridden individually by using the `syn_padtype` attribute.

The following example shows the `syn_fast_auto` attribute when applied on collections.

FDC

```
define_scope_collection  output_bufs {q[3:0]}
define_attribute {$output_bufs} {xc_fast_auto} {0}
```

Constraint

Collections Tab:

	Enabled	Collection Name	Command	Command Arguments
1	✔	output_bufs	q[3:0]	q[3:0]

Attributes Tab:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	✔	input_port	$output_bufs	xc_fast_auto	0	boolean	Enable automatic fast output buffer use

The following lines can be seen in the log file when the attribute is applied using collections.

```
Adding property xc_fast_auto, value 0, to inst q[3:0]
Adding property xc_fast_auto, value 0, to port q[3:0]
Adding property xc_fast_auto, value 0, to net q[0]
Adding property xc_fast_auto, value 0, to net q[1]
Adding property xc_fast_auto, value 0, to net q[2]
Adding property xc_fast_auto, value 0, to net q[3]
```

Copyright © 2013 Synopsys, Inc.

syn_force_pads

Attribute; Lattice. Prevents unused ports from being optimized away to allow I/O pad insertion on the unused port. The attribute can be applied to individual ports or at the global level.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute {*object*} syn_force_pads {1|0}

define_global_attribute syn_force_pads {1|0}

For example:

```
define_global_attribute syn_force_pads {1}
define_attribute {dataout[63]} syn_force_pads {0}
```


Copyright © 2013 Synopsys, Inc.

syn_force_seq_prim

Directive. Applies the "fix gated clocks" algorithm to the associated primitive.

To use the syn_force_seq_prim directive with a black box, you must also identify the clock signal using the syn_isclock directive and the enable signal using the syn_gatedclk_clock_en directive. The data type is Boolean.

The syn_force_seq_prim directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

For information about using this directive in working with gated clocks, see *Using Gated Clocks for Black Boxes*, on page 489 of the *User Guide*.

Verilog Syntax and Examples

object **/* synthesis syn_force_seq_prim = 1 */ ;**

where *object* is the module name of the black box.

```
module bbe (ena, clk, data_in, data_out)
   /* synthesis syn_black_box */
   /* synthesis syn_force_seq_prim=1 */ ;
input clk /* synthesis syn_isclock = 1 */
   /* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

VHDL Syntax and Examples

attribute syn_force_seq_prim of *object*: *objectType* **is true ;**

where *object* is the entity name of the black-box.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity bbram is
   port (addr: IN std_logic_VECTOR(6 downto 0);
         din: IN std_logic_VECTOR(7 downto 0);
         dout: OUT std_logic_VECTOR(7 downto 0);
         clk: IN std_logic;
         en: IN std_logic;
         we: IN std_logic );
attribute syn_black_box : boolean ;
attribute syn_black_box of bbram : entity is true ;
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 entity bbram;

architecture bb of bbram is
attribute syn_force_seq_prim : boolean;
attribute syn_force_seq_prim of bb : architecture is true;
begin
end architecture bb;
```


Copyright © 2013 Synopsys, Inc.

syn_forward_io_constraints

Attribute; Altera, Xilinx. Enables/disables forward annotation of the define_input_delay and define_output_delay timing constraints in the `tcl` or `ncf` file for the place-and-route tool. The default is 1 (I/O constraints are forward-annotated).

Use this attribute on the top level of a VHDL or Verilog file, or you can place syn_forward_io_constraints on a global object in the constraint file.

Constraint File Syntax and Example

define_global_attribute syn_forward_io_constraints {1|0}

For example:

```
define_global_attribute syn_forward_io_constraints {0}
```

Verilog Syntax and Example

object **/* synthesis syn_forward_io_constraints = 1 | 0 */;**

For example:

```
module test (a,b,c,d,clk,rst)
   /* synthesis syn_forward_io_constraints = 0 */;
input a,b,c;
input clk, rst;
output d;
wire temp;
reg d;
assign temp = a && b || c;

always@(posedge clk or posedge rst)
if (rst)
   d = 1 'b0;
else
   d=temp;
endmodule
```

VHDL Syntax and Example

attribute syn_forward_io_constraints of *object* **:** *objectType* **is true | false ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library IEEE;
use ieee.std_logic_1164.all;

entity test is
   port (a : in std_logic;
         b : in std_logic;
         clk :in std_logic;
         rst : in std_logic;
         d: out std_logic );
end test;

architecture archtest of test is
attribute syn_forward_io_constraints : boolean;
attribute syn_forward_io_constraints of all :
   architecture is true;
signal temp : std_logic;
begin
temp <= a and b;
   process (clk, rst)begin
      if (rst ='1')then
         d <='0';
      elsif (clk'event and clk ='1')then
         d <= temp;
      end if;
   end process;
end archtest;
```


Copyright © 2013 Synopsys, Inc.

syn_fsm_correction

Directive; This directive lets you specify a Hamming 3 encoding to be used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers. When you specify the attribute value as hamming3, the value is propagated to the state machines in the module.

If the hamming3 value and distributed TMR (see *syn_radhardlevel*, on page 1462) are both applied to a module that contains an FSM, the FSM is triplicated along with the rest of the module, and the hamming3 encoding specification is ignored. You also get a warning message in the log file.

Verilog Syntax and Example

module **/* syn_fsm_correction="hamming3" */;**

For example:

```
module top (input a, output b)
   /* synthesis syn_fsm_correction="hamming3" */
```

VHDL Syntax and Example

attribute syn_fsm_correction : string;

attribute syn_fsm_correction of *architectureName* **: architecture is "hamming3";**

For example:

```
library ieee;
use ieee.std_logic_1164.all;


entity test is
port (a input std_logic;
      b out std_logic);
end test;


architecture rtl of test is
attribute syn_fsm_correction : string;
attribute syn_fsm_correction of rtl : architecture is "hamming3";
```

syn_gatedclk_clock_en

Directive. Synplify Pro and Synplify Premier tools. Specifies the name of the enable pin inside a black box to support the "fix gated clocks" feature.

To use the syn_gatedclk_clock_en directive with a black box, you must also identify the clock signal with the syn_isclock directive and indicate that the fix gated clocks algorithm can be applied with the syn_force_seq_prim directive. The data type is String.

The syn_gatedclk_clock_en directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

For information about using this directive in working with gated clocks, see *Using Gated Clocks for Black Boxes*, on page 489 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Examples

object /* **synthesis syn_gatedclk_clock_en = "***value***"** */ **;**

where *object* is the module name of the black box and *value* is the name of the enable pin.

```
module bbe (ena, clk, data_in, data_out)
   /* synthesis syn_black_box */
   /* synthesis syn_force_seq_prim=1 */;
input clk
   /* synthesis syn_isclock = 1 */
   /* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

VHDL Syntax and Examples

attribute syn_gatedclk_clock_en of *object***:** *objectType* **is** *value* **;**

where *object* is the entity name of the black-box.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
architecture top of top is
component bbram
   port (myclk : in bit;
         opcode : in bit_vector(2 downto 0);
         a, b : in bit_vector(7 downto 0);
         rambus : out bit_vector(7 downto 0) );
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of bbram: component is true;
attribute syn_force_seq_prim : boolean
attribute syn_force_seq_prim of bbram: component is true;
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of bbram: signal is "ena";

-- Other code
```

syn_gatedclk_clock_en_polarity

Directive. Synplify Pro and Synplify Premier tools. Indicates the polarity of the clock enable port on a black box, so that the software can apply the algorithm to fix gated clocks. If you do not set any polarity with this attribute, the software assumes a positive polarity.

The syn_gatedclk_clock_en_priority directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Verilog Syntax and Examples

object **/* synthesis syn_gatedclk_clock_en_polarity = 1 | 0 */ ;**

object is the module name of the black box.

The value can be 1 or 0. A value of 1 indicates positive polarity of the enable signal (active high) and a value of 0 indicates negative polarity (active low). If the attribute is not defined, the tool assumes a positive polarity by default.

The following code segment provides an example.

```
module bbe1 (ena, clk, data_in, data_out)
   /* synthesis syn_black_box */
   /* synthesis syn_force_seq_prim=1 */;
input clk /* synthesis syn_isclock = 1 */
   /* synthesis syn_gatedclk_clock_en="ena" */
   /* synthesis syn_gatedclk_clock_en_polarity = 0 */;
input data_in,ena;
output data_out;
endmodule

module bbe2 (ena, clk, data_in, data_out)
   /* synthesis syn_black_box */
   /* synthesis syn_force_seq_prim=1 */;
input clk /* synthesis syn_isclock = 1 */
   /* synthesis syn_gatedclk_clock_en_polarity = 1 */
   /* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```


Copyright © 2013 Synopsys, Inc.

```
module top (ena, clk, data_in , data_in2, data_out, data_out2 ) ;
input ena , clk , data_in , data_in2;
output data_out, data_out2 ;
wire clk_in ;
wire inv_enable;
wire clk_in2 ;
assign inv_enable = ~ena;
assign clk_in = inv_enable & clk ;
assign clk_in2 = ena & clk ;
bbe1 u1 ( ena , clk_in , data_in , data_out ) ;
bbe2 u2 ( ena, clk_in2, data_in2, data_out2 ) ;
endmodule
```

VHDL Syntax and Examples

attribute syn_gatedclk_clock_en_polarity of *object*: *objectType* **is true | false;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity bbe1 is
   port (ena : in std_logic;
         clk : in std_logic;
         data_in : in std_logic;
         data_out : out std_logic );
attribute syn_black_box : boolean;
attribute syn_force_seq_prim : boolean;
attribute syn_gatedclk_clock_en_polarity : boolean;
attribute syn_gatedclk_clock_en : string;
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en_polarity of clk: signal is false;
attribute syn_gatedclk_clock_en of clk: signal is "ena";
attribute syn_force_seq_prim of clk: signal is true ;
end bbe1;
```

Copyright © 2013 Synopsys, Inc.

```
architecture arch_bbe1 of bbe1 is
attribute syn_black_box : boolean;
attribute syn_black_box of arch_bbe1: architecture is true;
attribute syn_force_seq_prim of arch_bbe1: architecture is true;
begin
end arch_bbe1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity bbe2 is
   port (ena : in std_logic;
         clk : in std_logic;
         data_in2 : in std_logic;
         data_out2 : out std_logic );
attribute syn_black_box : boolean;
attribute syn_gatedclk_clock_en_polarity : boolean;
attribute syn_force_seq_prim : boolean;
attribute syn_gatedclk_clock_en : string;
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en_polarity of clk: signal is true;
attribute syn_gatedclk_clock_en of clk: signal is "ena";
attribute syn_force_seq_prim of clk: signal is true ;
end bbe2;

architecture arch_bbe2 of bbe2 is
attribute syn_black_box : boolean;
attribute syn_black_box of arch_bbe2: architecture is true;
attribute syn_force_seq_prim of arch_bbe2: architecture is true;
begin
end arch_bbe2;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity top is
   port (ena : in std_logic;
         clk : in std_logic;
         data_in, data_in2 : in std_logic;
         data_out, data_out2 : out std_logic );
end top;
```


Copyright © 2013 Synopsys, Inc.

```
architecture arch_top of top is
component bbe1 is
   port (ena : in std_logic;
         clk : in std_logic;
         data_in : in std_logic;
         data_out : out std_logic );
end component;

component bbe2 is
   port (ena : in std_logic;
         clk : in std_logic;
         data_in2 : in std_logic;
         data_out2 : out std_logic);
end component;

signal clk_in, inv_enable, clk_in2 : std_logic;
begin
inv_enable <= not(ena);
clk_in <= inv_enable and clk;
clk_in2 <= ena and clk;
U1 : bbe1 port map (ena => ena, clk => clk_in,
   data_in => data_in, data_out => data_out);
U2 : bbe2 port map (ena => ena, clk => clk_in2,
   data_in2 => data_in2, data_out2 => data_out2);
end arch_top;
```

syn_global_buffers

Attribute; Microsemi IGLOO/IGLOOe, ProASIC3/3E, Lattice iCE65/iCE40, and Xilinx. Specifies the number of global buffers to be used in a design. The synthesis tool automatically adds global buffers for clock nets with high fanout; use this attribute to specify a maximum number of buffers and restrict the amount of global buffer resources used. Also, if there is a black box in the design that has global buffers, you can use syn_global_buffers to prevent the synthesis tool from inferring clock buffers or exceeding the number of global resources.

You specify the attribute globally on the top-level module/entity or view. For Microsemi designs, it can be any integer between 6 and 18. If you specify an integer less than 6, the software infers six global buffers. For Xilinx designs, the value varies with the technology.

For Xilinx designs, the syn_global_buffers attribute works just like the Xilinx xc_global_buffers attribute (*xc_global_buffers*, on page 1605); the only difference is that you can specify syn_global_buffers in the source code or in a constraint file. If both attributes are specified for the same Xilinx design, syn_global_buffers overwrites xc_global_buffers. Use this attribute instead of the Xilinx global buffers attribute xc_global_buffers.

Constraint File Syntax and Example

define_attribute {*view***} syn_global_buffers {***maximum***}**

define_global_attribute syn_global_buffers {*maximum***}**

For example:

```
define_global_attribute syn_global_buffers {10}
```

Verilog Syntax and Example

object **/* synthesis syn_global_buffers =** *maximum* ***/;**

For example:

```
module top (clk1, clk2, clk3, clk4, clk5, clk6, clk7,clk8,clk9,
   clk10, clk11, clk12, clk13, clk14, clk15, clk16, clk17, clk18,
   clk19, clk20, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11,
   d12, d13, d14, d15, d16, d17, d18, d19, d20, q1, q2, q3, q4, q5,
   q6, q7, q8, q9, q10, q11, q12, q13, q14, q15, q16, q17, q18,
```


Copyright © 2013 Synopsys, Inc.

```
   q19, q20, reset) /* synthesis syn_global_buffers = 10 */;
input clk1, clk2, clk3, clk4, clk5, clk6, clk7,clk8,clk9, clk10,
   clk11, clk12, clk13, clk14, clk15, clk16, clk17, clk18,
   clk19, clk20;
input d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14,
   d15, d16, d17, d18, d19, d20;
output q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14,
   q15, q16, q17, q18, q19, q20;
input reset;
reg q1, q2, q3, q4, q5, q6, q7, q8, q9, q10,
    q11, q12, q13, q14, q15, q16, q17, q18, q19, q20;

always @(posedge clk1 or posedge reset)
   if (reset)
      q1 <= 1'b0;
   else
      q1 <= d1;

always @(posedge clk2 or posedge reset)
   if (reset)
      q2 <= 1'b0;
   else
      q2 <= d2;

always @(posedge clk3 or posedge reset)
   if (reset)
      q3 <= 1'b0;
   else
      q3 <= d3;

always @(posedge clk4 or posedge reset)
   if (reset)
      q4 <= 1'b0;
   else
      q4 <= d4;

always @(posedge clk5 or posedge reset)
   if (reset)
      q5 <= 1'b0;
   else
      q5 <= d5;

always @(posedge clk6 or posedge reset)
   if (reset)
      q6 <= 1'b0;
   else
      q6 <= d6;
```

```
always @(posedge clk7 or posedge reset)
   if (reset)
      q7 <= 1'b0;
   else
      q7 <= d7;

always @(posedge clk8 or posedge reset)
   if (reset)
      q8 <= 1'b0;
   else
      q8 <= d8;

always @(posedge clk9 or posedge reset)
   if (reset)
      q9 <= 1'b0;
   else
      q9 <= d9;

always @(posedge clk10 or posedge reset)
   if (reset)
      q10 <= 1'b0;
   else
      q10 <= d10;

always @(posedge clk11 or posedge reset)
   if (reset)
      q11 <= 1'b0;
   else
      q11 <= d11;

always @(posedge clk12 or posedge reset)
   if (reset)
      q12 <= 1'b0;
   else
      q12 <= d12

always @(posedge clk13 or posedge reset)
   if (reset)
      q13 <= 1'b0;
   else
      q13 <= d13;

always @(posedge clk14 or posedge reset)
   if (reset)
      q14 <= 1'b0;
   else
      q14 <= d14;
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge clk15 or posedge reset)
   if (reset)
      q15 <= 1'b0;
   else
      q15 <= d15;

always @(posedge clk16 or posedge reset)
   if (reset)
      q16 <= 1'b0;
   else
      q16 <= d16;

always @(posedge clk17 or posedge reset)
   if (reset)
      q17 <= 1'b0;
   else
      q17 <= d17;

always @(posedge clk18 or posedge reset)
   if (reset)
      q18 <= 1'b0;
   else
      q18 <= d18;

always @(posedge clk19 or posedge reset)
   if (reset)
      q19 <= 1'b0;
   else
      q19 <= d19;

always @(posedge clk20 or posedge reset)
   if (reset)
      q20 <= 1'b0;
   else
      q20 <= d20;

endmodule
```

VHDL Syntax and Example

attribute syn_global_buffers of *object* **:** *objectType* **is** *maximum***;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity top is
   port (clk : in std_logic_vector(19 downto 0);
         d : in std_logic_vector(19 downto 0);
         q : out std_logic_vector(19 downto 0);
         reset : in std_logic );
end top;

architecture behave of top is
attribute syn_global_buffers : integer;
attribute syn_global_buffers of behave : architecture is 10;
begin
   process (clk, reset)
   begin
      for i in 0 to 19 loop
         if (reset = '1') then
            q(i) <= '0';
         elsif clk(i) = '1' and clk(i)' event then
            q(i) <= d(i);
         end if;
      end loop;
   end process;
end behave;
```


Copyright © 2013 Synopsys, Inc.

syn_hier

Attribute. Lets you control the amount of hierarchical transformation across boundaries on module or component instances during optimization.

During synthesis, the synthesis tool dissolves as much hierarchy as possible to allow efficient logic optimization across hierarchical boundaries while maintaining optimal run times. The tool then rebuilds the hierarchy as close as possible to the original source to preserve the topology of the design. Use the syn_hier attribute to address specific needs to maintain the original design hierarchy during optimization. This attribute gives you manual control over flattening/preserving instances, modules, or architectures in the design.

Constraint File Syntax and Example

define_attribute {*object*} syn_hier {*value*}

where *object* is a view, and *value* can be any of the values described in *syn_hier Values*, on page 1345. Note however, if you are defining syn_hier globally, it is recommended that you use the SCOPE collection to apply syn_hier on all views instead. For example:

```
define_scope_collection all_views {find -hier -view {*}}
define_attribute {$all_views} {syn_hier} {fixed}
```

Check the attribute values to determine where to attach the attribute. Here is an example:

```
define_attribute {v:fifo} syn_hier {hard}
```

Make sure to specify the attribute on the view (**v:** object type). See *syn_hier in the SCOPE Window*, on page 1346 for details.

Verilog Syntax and Examples

object /* **synthesis syn_hier = "***value***"** */ **;**

where *object* can be a module declaration and *value* can be any of the values described in *syn_hier Values*, on page 1345. Check the attribute values to determine where to attach the attribute.

Copyright © 2013 Synopsys, Inc.

This is the Verilog syntax:

```
module fifo(out, in) /* synthesis syn_hier = "hard" */;

// Other code
```

VHDL Syntax and Examples

attribute syn_hier of *object* **: architecture is "***value***" ;**

where *object* is an architecture name and *value* can be any of the values described in *syn_hier Values*, on page 1345. Check the attribute values to determine the level at which to attach the attribute.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives. This is the VHDL syntax:

```
architecture struct of cpu is

attribute syn_hier : string;
attribute syn_hier of struct: architecture is "hard";

-- Other code
```

syn_hier Values

The following table shows the values you can use for syn_hier. For additional information about using this attribute in HDL Analyst, see *Controlling Hierarchy Flattening*, on page 442 and *Preserving Hierarchy*, on page 442 in the *User Guide*.

soft (default)	The synthesis tool determines the best optimization across hierarchical boundaries. This attribute affects only the design unit in which it is specified.
firm	Preserves the interface of the design unit. However, when there is cell packing across the boundary, it changes the interface and does not guarantee the exact RTL interface. This attribute affects only the design unit in which it is specified.
hard	Preserves the interface of the design unit and prevents most optimizations across the hierarchy. However, the boundary optimization for constant propagation and gated clock conversion are performed. This attribute affects only the specified design units.

Copyright © 2013 Synopsys, Inc.

fixed	Preserves the interface of the design unit with no exceptions. Fixed prevents all optimizations performed across hierarchical boundaries and retains the port interfaces as well. For more information, see *Using syn_hier fixed*, on page 1347.
remove	Removes the level of hierarchy for the design unit in which it is specified. The hierarchy at lower levels is unaffected. This only affects synthesis optimization. The hierarchy is reconstructed in the netlist and Technology view schematics.
macro	Preserves the interface and contents of the design with no exceptions. This value can only be set on structural netlists. (In the constraint file, or using the SCOPE editor, set syn_hier to macro on the view (the **v:** object type).
flatten	Flattens the hierarchy of all levels below, but not the one where it is specified. This only affects synthesis optimization. The hierarchy is reconstructed in the netlist and Technology view schematics. To create a completely flattened netlist, use the syn_netlist_hierarchy attribute (*syn_netlist_hierarchy*, on page 1413), set to false. You can use flatten in combination with other syn_hier values; the effects are described in *Using syn_hier flatten with Other Values*, on page 1349. If you apply syn_hier to a compile point, flatten is the only valid attribute value. All other values only apply to the current level of hierarchy. The compile point hierarchy is determined by the type of compile point specified, so a syn_hier value other than flatten is redundant and is ignored.

syn_hier in the SCOPE Window

If you use the SCOPE window to specify the syn_hier attribute, do not drag and drop the object into the SCOPE spreadsheet. Instead, first select syn_hier in the Attribute column, and then use the pull-down menu in the Object column to select the object. This is because you must set the attribute on a view (v:). If you drag and drop an object, you might not get a view object. Selecting the attribute first ensures that only the appropriate objects are listed in the Object column.

Copyright © 2013 Synopsys, Inc.

Using syn_hier fixed

When you use the fixed value with syn_hier, hierarchical boundaries are preserved with no exceptions. For example, optimizations such as constant propagation and gated or generated clock conversions are not performed across these boundaries.

Note: It is recommended that you do not use syn_hier with the fixed value on modules that have ports driven by tri-state gates. For details, see *When Using Tri-states*, on page 1347.

When Using Tri-states

It is advised that you avoid using syn_hier="fixed" with tri-states. However, if you do, here is how the software handles the following conditions:

- Tri-states driving output ports

  If a module with syn_hier="fixed" includes tri-state gates that drive a primary output port, then the synthesis software retains a tri-state buffer so that the P&R tool can pack the tri-state into an output port.

  For example, the software can infer a Xilinx BUFT so that the ISE P&R tool packs the tri-state into an OBUFT output port.

- Tri-states driving internal logic

  If a module with syn_hier="fixed" includes tri-state gates that drive internal logic, then the synthesis software converts the tri-state gate to a MUX and optimizes within the module accordingly.

In the following code example, myreg has syn_hier set to fixed.

```
module top(
   clk1,en1, data1,
   q1, q2
   );
input clk1, en1;
input data1;
output q1, q2;
wire cwire, rwire;
wire clk_gt;
assign clk_gt = en1 & clk1;
```


Copyright © 2013 Synopsys, Inc.

```
// Register module
myreg U_reg (
   .datain(data1),
   .rst(1'b1),
   .clk(clk_gt),
   .en(1'b0),
   .dout(rwire),
   .cout(cwire)
   );
assign q1 = rwire;
assign q2 = cwire;
endmodule

module myreg (
   datain,
   rst,
   clk,
   en,
   dout,
   cout
   ) /* synthesis syn_hier = "fixed" */;
input clk, rst, datain, en;
output dout;
output cout;
reg dreg;
assign cout = en & datain;

always @(posedge clk or posedge rst)
   begin
      if (rst)
         dreg <= 'b0;
      else
         dreg <= datain;
   end
assign dout = dreg;
endmodule
```

The HDL Analyst views show that myreg preserves its hierarchical boundaries without exceptions and prevents constant propagation and gated clock conversions optimizations.

Using syn_hier flatten with Other Values

You can combine flatten with other syn_hier values as shown below:

flatten,soft	Same as flatten.
flatten,firm	Flattens all lower levels of the design but preserves the interface of the design unit in which it is specified. This option also allows optimization of cell packing across the boundary.
flatten,remove	Flattens all lower levels of the design, including the one on which it is specified.

Copyright © 2013 Synopsys, Inc.

If you use flatten in combination with another option, the tool flattens as directed until encountering another syn_hier attribute at a lower level. The lower level syn_hier attribute then takes precedence over the higher level one.

These example demonstrate the use of the flatten and remove values to flatten the current level of the hierarchy and all levels below it (unless you have defined another syn_hier attribute at a lower level).

Verilog
```
module top1 (Q, CLK, RST, LD, CE, D)
/* synthesis syn_hier = "flatten,remove" */;

// Other code
```

VHDL
```
architecture struct of cpu is

attribute syn_hier : string;
attribute syn_hier of struct: architecture is "flatten,remove";

-- Other code
```

Example of syn_hier hard

Here is an example of two versions of a design: one with syn_hier set on modules mem and data; the other shows what happens to those modules with the automatic flattening that occurs during synthesis.

With syn_hier="hard"

Without syn_hier="hard"
This is the default.

Copyright © 2013 Synopsys, Inc.

syn_highrel_ioconnector

Attribute; Altera and Xilinx

Adds I/O connectors that interface to modules with distributed TMR or DWC, which allows you to access the signals within its boundaries ensuring their inputs and outputs are not a single-point of failure.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute syn_highrel_ioconnector = {1|0}

For example:

```
define_attribute {v:work.ipconn_dtmr}
   {syn_highrel_ioconnector} {1}
```

Verilog Syntax and Example

module **/* syn_highrel_ioconnector="1" */;**

For example:

```
module ipconn_dtmr (in1, in2, in3, out)
   /* synthesis syn_highrel_ioconnector="1" */
```

VHDL Syntax and Example

attribute syn_highrel_ioconnector : string;

attribute syn_highrel_ioconnector of *architectureName* **: architecture is "1";**

syn_insert_buffer (Microsemi)

Attribute; Microsemi Axcelerator, IGLOO/IGLOOe/IGLOO+, ProASIC^{PLUS}, ProASIC3/3E/3L The synthesis tool inserts a clock buffer according to the vendor-specific values you specify. The attribute can be applied on instances.

Constraint File Syntax and Example

define_attribute {*object***} syn_insert_buffer {***value***}**

Where:

- *object* can be an instance.
- *value* can be the following:

Microsemi	For supported Microsemi devices, specify the value of the global clock buffer. For the Axcelerator family, the following attribute values are supported when adding a buffer to: • Pads: CLKBUF, HCLKBUF • Nets: CLKINT, HCLKINT Note: HCLKBUF and HCLKINT are supported only for nets driving clock pins of sequential primitives.

Verilog Example (Microsemi)

object **/* synthesis syn_insert_buffer = "***value***" */ ;**

In the following example, the attribute is attached to LDPRE, SEL, RST, LDCOMP, and CLK.

```
module prep2_2 (DATA0, DATA1, DATA2, LDPRE, SEL, RST, CLK, LDCOMP);
output [7:0] DATA0;
input [7:0] DATA1, DATA2;
input LDPRE, SEL, RST, CLK
   /* synthesis syn_insert_buffer = "GL25" */, LDCOMP;
wire [7:0] DATA0_internal;
prep2_1 inst1 (CLK, RST, SEL, LDCOMP, LDPRE, DATA1, DATA2,
   DATA0_internal);
prep2_1 inst2 (CLK, RST, SEL, LDCOMP, LDPRE, DATA0_internal,
   DATA2, DATA0);
endmodule
```


Copyright © 2013 Synopsys, Inc.

```
module prep2_1 (CLK, RST, SEL, LDCOMP, LDPRE, DATA1, DATA2, DATA0);
input CLK, RST, SEL, LDCOMP, LDPRE ;
input [7:0] DATA1, DATA2 ;
output [7:0] DATA0 ;
reg [7:0] DATA0;
reg [7:0] highreg_output, lowreg_output; // internal registers
wire compare_output = (DATA0 == lowreg_output);  // comparator
wire [7:0]  mux_output = SEL ? DATA1 : highreg_output;

// mux registers
always @ (posedge CLK  or posedge RST)
begin
   if (RST) begin
      highreg_output = 0;
      lowreg_output = 0;
   end else begin
      if (LDPRE)
         highreg_output = DATA2;
      if (LDCOMP)
         lowreg_output  = DATA2;
   end
end


// counter
always @(posedge CLK or posedge RST)
begin
   if (RST)
      DATA0 = 0;
   else if (compare_output)  // load
      DATA0 = mux_output;
   else
      DATA0 = DATA0 + 1;
end

endmodule
```

VHDL Example (Microsemi)

attribute syn_insert_buffer of *object* **:** *string* ;

See *VHDL Attribute and Directive Syntax*, on page 963 for alternate methods for specifying VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity prep2_1 is
   port (clk : in bit;
         rst : in bit;
         sel : in bit;
         ldcomp : in bit;
         ldpre : in bit;
         data1,data2 : in std_logic_vector(7 downto 0);
         data0 : out std_logic_vector(7 downto 0) );
end prep2_1;

architecture behave of prep2_1 is
signal equal: bit;
signal mux_output: std_logic_vector(7 downto 0);
signal lowreg_output: std_logic_vector(7 downto 0);
signal highreg_output: std_logic_vector(7 downto 0);
signal data0_i: std_logic_vector(7 downto 0);
begin
   compare: process(data0_i, lowreg_output)
   begin
      if data0_i = lowreg_output then
         equal <= '1';
      else
         equal <= '0';
      end if;
   end process compare;

   mux: process(sel, data1, highreg_output)
   begin
      case sel is
         when '0' =>
            mux_output <= highreg_output;
         when '1' =>
            mux_output <= data1;
      end case;
   end process mux;

   registers: process (rst,clk)
   begin
      if ( rst = '1') then
         highreg_output <= "00000000";
         lowreg_output <= "00000000";
      elsif clk = '1' and clk'event then
```


Copyright © 2013 Synopsys, Inc.

```
            if ldpre = '1' then
               highreg_output <= data2;
            end if;
            if ldcomp = '1' then
               lowreg_output <= data2;
            end if;
         end if;
      end process registers;

      counter: process (rst,clk)
      begin
         if  rst = '1' then
            data0_i <= "00000000";
         elsif clk = '1' and clk'event then
            if equal = '1' then
               data0_i <= mux_output;
            elsif equal = '0' then
               data0_i  <= data0_i + "00000001";
            end if;
         end if;
      end process counter;
   data0 <= data0_i;
   end behave;

   library ieee;
   use ieee.std_logic_1164.all;

   entity prep2_2 is
      port (CLK : in bit;
            RST : in bit;
            SEL : in bit;
            LDCOMP : in bit;
            LDPRE : in bit;
            DATA1,DATA2 : in std_logic_vector(7 downto 0);
            DATA0 : out std_logic_vector(7 downto 0) );
   attribute syn_insert_buffer : string;
   attribute syn_insert_buffer of clk : signal is "GL25";
   end prep2_2;

   architecture behave of prep2_2 is
   component prep2_1
      port (clk : in bit;
            rst : in bit;
            sel : in bit;
```

```
            ldcomp : in bit;
            ldpre : in bit;
            data1,data2 : in std_logic_vector(7 downto 0);
            data0 : out std_logic_vector(7 downto 0) );
end component;

signal data0_internal : std_logic_vector (7 downto 0);

begin
inst1: prep2_1 port map(clk => CLK, rst => RST, sel => SEL,
   ldcomp => LDCOMP, ldpre => LDPRE, data1 => DATA1,
   data2 => DATA2, data0 => data0_internal );
inst2: prep2_1 port map(clk => CLK, rst => RST, sel => SEL,
   ldcomp => LDCOMP, ldpre => LDPRE, data1 => data0_internal,
   data2 => DATA2, data0 => DATA0 );
end behave;
```


Copyright © 2013 Synopsys, Inc.

syn_insert_buffer (Lattice)

Attribute; Lattice iCE65/iCE40.

Inserts global buffers on a non-clock pin. For a design that might have a non-clock high fanout net, such as reset or common enable that needs global routing, use syn_insert_buffer to insert a global buffer for that port.

Constraint File Syntax and Example

define_attribute {*object*} syn_insert_buffer {*value*}

Where:

- *object* can be a port or net.
- *value* can be the following:
  - For ports specify the value SB_GB_IO
  - For nets specify the value SB_GB

Verilog Syntax and Example

Object /* synthesis syn_insert_buffer = "{*value*}" */;

Example – syn_insert_buffer Attribute

```
module test
   (CLK, din1, din2, din3, din4,Q1, Q2, reset, gt1, gt2);

input gt1, gt2;
input CLK;
input reset /* synthesis syn_insert_buffer = "SB_GB_IO" */;
input din1;
input din2 /* synthesis syn_insert_buffer = "SB_GB_IO" */;
input din3 /* synthesis syn_insert_buffer = "SB_GB_IO" */;
input din4;
output reg Q1, Q2;

wire gt11 /* synthesis syn_insert_buffer = "SB_GB" syn_keep = 1 */;

assign gt11 = gt1;

wire int_clk_glob;
wire int_clk_core;
```

```
wire int_clk_glob_gt;
wire int_clk_core_gt;

reg  reg_1, reg_2, reg_3, reg_4;

assign int_clk_glob_gt = CLK & gt11;
assign int_clk_core_gt = CLK & gt2;

always @(posedge int_clk_core_gt or negedge reset)
begin
   if (!reset)
   reg_1 <= 0;
   else
   begin
   reg_1 <= din1;
   reg_2 <= din2;
   Q1 <= reg_1 + reg_2;
   end
end

always @(posedge int_clk_glob_gt)
begin
   reg_3 <= din3;
   reg_4 <= din4;
   Q2 <= reg_3 + reg_4;
end

endmodule
```

This code inserts SB_GB_IO buffers for the reset, din2, and din3 ports. Otherwise these ports will use the SB_IO buffer and infer the SB_GB buffer on the gt11 net.

VHDL Syntax

attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of *object* **:** *object_type* **is "{***value***}";**

Copyright © 2013 Synopsys, Inc.

syn_insert_buffer (Xilinx)

Attribute; Xilinx. The synthesis tool inserts a clock buffer according to the vendor-specific values you specify.

Constraint File Syntax and Example

define_attribute {*object*} syn_insert_buffer {*value*}

Where:

- *object* can be
  - an instance for BUFGMUX inferencing
  - a net or port for BUFR, BUFH, and BUFG inferencing
- *value* can be the following:

BUFGMUX	BUFGMUX inference is disabled by default. You can override this behavior by setting one of the following options for the mux with the syn_insert_buffer attribute. When you specify: • none, then the software infers only the LUT without the BUFG. • bufgmux, then the software infers the BUFGMUX. If the syn_insert_buffer attribute is not specified, then the default behavior of the software is to infer the LUT driving the BUFG unless you specify a value of none or bufgmux.

BUFG	BUFG inferencing is available for *Virtex-4* and newer devices. To infer BUFG: • Apply the syn_insert_buffer attribute on a port or net. • Set the attribute value to BUFG.
BUFH	BUFH inferencing is available for *Virtex-4* and newer devices. BUFH inference is disabled by default. To specifically infer BUFH: • Apply the syn_insert_buffer attribute on a port or net. • Set the attribute value to BUFH.
BUFR	BUFR inferencing is available for *Virtex-4* and newer devices. BUFR inference is disabled by default. To specifically infer BUFR: • Apply the syn_insert_buffer attribute on a port or net. • Set the attribute value to BUFR.

Effect of Using syn_insert_buffer with BUFGMUX

The Verilog and VHDL examples below show that syn_insert_buffer infers the BUFGMUX.

Example 1 – BUFGMUX Inferencing in Verilog

```
// Attribute attached to clk_mux.
`define SIZE 16

module cr_top (
input rst,
input [`SIZE:0] in1,in2,
output reg [`SIZE:0] out,
input clk1,
input clk2,
input sel_clk );
wire  clk_mux/*synthesis syn_insert_buffer="BUFGMUX"*/;
assign clk_mux = sel_clk ? clk1 : clk2;

always @(posedge clk_mux or posedge rst)
begin
   if(rst)
   begin
      out <= 0;
   end
   else
```


Copyright © 2013 Synopsys, Inc.

```
        begin
            out <= in1&in2 ;
        end
    end
    endmodule
```

The following figure shows how the syn_insert_buffer infers the BUFGMUX.

Example 2 – BUFGMUX Inferencing in VHDL

-- Attribute attached to clk_out.

```
library ieee;
use ieee.std_logic_1164.all;

entity Top is
port( clk1: in std_logic;
```

```
      clk2: in std_logic;
      sel_clk: in std_logic;
      in1,in2: in std_logic_vector (2 downto 0);
      outp: out std_logic_vector (2 downto 0) );
end Top;

architecture behav of Top is
signal clk_out: std_logic;
attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk_out : signal is "BUFGMUX";
begin
    process1: process(sel_clk,clk1,clk2)
    begin
      case sel_clk is
         when '0' => clk_out <= clk1;
         when '1' => clk_out <= clk2;
      end case;
   end process process1;
   process2: process(clk_out)
   begin
      if rising_edge (clk_out) then
         outp <= in1 and in2;
      end if;
   end process process2;
end behav;
```

The following figure shows how the syn_insert_buffer infers the BUFGMUX.

Copyright © 2013 Synopsys, Inc.

Effect of Using syn_insert_buffer with BUFG

The following examples show designs using BUFG, where syn_insert_buffer is set on a net.

Example 1 – BUFG Applied to a Net in HDL

```
ifdef synthesis
module bufr_drom (clk,addr,q);

`else

module bufr_drom_rtl (clk,addr,q);

`endif

parameter addr_width = 6;
```

```
parameter data_width = 4;

input clk/* synthesis syn_insert_buffer = "bufg"*/;

input [addr_width - 1 : 0] addr;

output [data_width - 1 : 0] q;
wire [data_width - 1 : 0] q;

reg [data_width - 1 : 0] mem [(2**addr_width) - 1  : 0];
reg [addr_width - 1 : 0] reg_addr;

always@(posedge clk)
reg_addr <= addr;

initial $readmemb("mem1.dat", mem);

assign q = mem[reg_addr];

endmodule
```

The RTL and Technology views for the BUFG are shown below.

Copyright © 2013 Synopsys, Inc.

Example 2 – BUFG Applied to a Port in Constraint File

```
`ifdef synthesis
module bufr_srl (clk,sren,srin,srout);
`else
module bufr_srl_rtl (clk,sren,srin,srout);

`endif

input clk;
input sren;
input srin;
output [1:0] srout;

reg [31:0] sr0;
```

```
reg [31:0] sr1;
always@(posedge clk)
begin
   if (sren == 1)
      begin
         sr1 <= {sr1[30:0], srin};
         sr0 <= {srin,sr0[31:1]};
      end
end

assign srout[1] = sr1[31];
assign srout[0] = sr0[0];

endmodule
```

The syn_insert_buffer attribute is applied on p:clk in the constraint file.

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	☑	<any>	p:clk	syn_insert_buffer	BUFG	string	Applies a global clock...
2							
3							

The RTL and Technology views for the BUFG are shown below.

Copyright © 2013 Synopsys, Inc.

srout[1:0]
[31]
[0]
[30:0]
srin
clk
D[31:0]
Q[31:0]
E
sr1[31:0]
[31:0]
[31:1]
sren
sr0[31:0]
IBUF
srin_ibuf
OBUF
srout_obuf[1]
srout[1:0]
BUFG
IBUF
clk_ibuf
ck_keep
SRLC32E
FDE
OBUF
srout_obuf[0]
clk_c local_fanout = 1
is_gated_clock = 1
is_clock = 1
syn_priority_applied = 1
.has_nback_att = 1
.cdb_cp = 0 = 205886_28_196365_422777_453046_233119
.cdb = 1002 = object = 0 = 1
syn_insert_buffer = BUFG
.netpropapplied = 1
syn_noprune = 1
.orig_net_name = clk
.syn_port_driven = 1

Effect of Using syn_insert_buffer with BUFR

The following examples show designs using BUFR, where syn_insert_buffer is set on a net, port, and entity respectively.

Example 1 – BUFR Applied to Net

```
module bufr_basic (
input rst,
input [14:0] in1,
input [14:0] in2,
output reg [29:0] out2,
```


Copyright © 2013 Synopsys, Inc.

```
input clk1 ,
input clk2/* synthesis syn_insert_buffer = "BUFR" */ );

reg [14:0] reg_in2;
reg [14:0] reg_in1;
reg [29:0] temp;

always @(posedge clk1)
begin
   if(rst)
      begin
         reg_in1 <= 0;
         reg_in2 <= 0;
         temp <= 0;
      end
   else
      begin
         reg_in1 <= in1 ;
         reg_in2 <= in2 ;
         temp <= reg_in1 + reg_in2 ;
      end
end

always @(posedge clk2)
begin
   if(rst)
      out2 <= 0;
   else
      out2 <= temp ;
end
endmodule
```

The following figure shows how the syn_insert_buffer infers the BUFR in the RTL and Technology views.

Example 2 – BUFR Applied to Port

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity bufr_ff_vhd is
port (
   clk : in std_logic;
   clk1 : in std_logic ;
   en : in std_logic;
   rst : in std_logic;
   ip1  : in std_logic;
   out1 : out std_logic_vector(1 downto 0));
end entity bufr_ff_vhd;

architecture bufr_arch of bufr_ff_vhd is
signal clk1_en : std_logic;

attribute syn_insert_buffer : string ;
```


Copyright © 2013 Synopsys, Inc.

```
attribute syn_insert_buffer of clk1_en : signal is "BUFR";

begin
clk1_en <= clk1  and en;

ff_reg :process (clk , rst)
   begin
   if(rst = '1') then
      out1(0) <= '0';
      elsif rising_edge(clk) then
      out1(0) <=  ip1;
   end if;
end process ff_reg;

ff_reg1 :process (clk1_en , rst)
   begin
      if(rst = '1') then
         out1(1) <= '0';
         elsif rising_edge(clk1_en) then
         out1(1) <=  ip1;
   end if;
end process ff_reg1;

end architecture bufr_arch;
```

The following figure shows how the syn_insert_buffer infers the BUFR in the RTL and Technology views.

Example 3 – Defining BUFR in Entity Declaration

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity bufr_ff_vhd is
```


Copyright © 2013 Synopsys, Inc.

```
port (
   clk : in std_logic;
   clk1 : in std_logic ;
   en : in std_logic;
   rst : in std_logic;
   ip1  : in std_logic;
   out1 : out std_logic_vector(1 downto 0));

attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk : signal is "BUFR";
end entity bufr_ff_vhd;

architecture bufr_arch of bufr_ff_vhd is
signal clk1_en : std_logic;

attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk1_en : signal is "BUFR";

begin

clk1_en <= clk1  and en;

ff_reg :process (clk , rst)
   begin
      if(rst = '1') then
         out1(0) <= '0';
         elsif rising_edge(clk) then
         out1(0) <=  ip1;
      end if;
   end process ff_reg;

ff_reg1 :process (clk1_en , rst)
   begin
      if(rst = '1') then
         out1(1) <= '0';
         elsif rising_edge(clk1_en) then
         out1(1) <=  ip1;
      end if;
   end process ff_reg1;

end architecture bufr_arch;
```

The following figure shows how the syn_insert_buffer infers the BUFR in the RTL and Technology views.

Copyright © 2013 Synopsys, Inc.

syn_insert_pad

Attribute

Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.

Vendor	Technology
Lattice	iCE40, iCE65
Xilinx	Virtex-7 and older families Spartan-6 and older families

syn_insert_pad Values

Value	Description	Default	Global	Object
0	Removes an IBUF/OBUF from a port or net	None	No	Port, net
1	Replaces a previously removed IBUF/OBUF on a port or net.	None	No	Port, net

Description

The syn_insert_pad attribute is used when the Disable I/O Insertion option is not enabled (when buffers are automatically inserted) to allow users to selectively remove an individual buffer from a port or net or to replace a previously removed buffer.

- Setting the attribute to 0 on a port or net removes the I/O buffer (or prevents an I/O buffer from being automatically inserted).
- Setting the attribute to 1 on a port or net replaces a previously removed I/O buffer.

The syn_insert_pad attribute can only be applied through a constraint file.

syn_insert_pad Syntax

FDC	**define_attribute {*object*} syn_insert_pad {1|0}**	FDC Example

FDC Example

The following figure shows the attribute applied to the RST port using the SCOPE window:

Enable	Object Type	Object	Attribute	Value	Value Type
☑	<any>	p:RST	syn_insert_pad	0	

Effect of Using syn_insert_pad

Original design before applying syn_insert_pad (or after applying syn_insert_pad with a value of 1 to replace a previously removed buffer).

Copyright © 2013 Synopsys, Inc.

Technology view after applying syn_insert_pad with a value of 0 to remove the original buffer from the RST input.

Copyright © 2013 Synopsys, Inc.

syn_isclock

Directive. Used with the syn_black_box directive and specifies an input port on a black box as a clock. Use the syn_isclock directive to specify that an input port on a black box is a clock, even though its name does not correspond to one of the recognized names. Using this directive connects it to a clock buffer if appropriate. The data type is Boolean.

The syn_isclock directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Verilog Syntax and Examples

object **/* synthesis syn_isclock = 1 */ ;**

where *object* is an input port on a black box.

```
module ram4 (myclk,out,opcode,a,b) /* synthesis syn_black_box */;
output [7:0] out;
input myclk /* synthesis syn_isclock = 1 */;
input [2:0] opcode;
input [7:0] a, b;

//Other code
```

VHDL Syntax and Examples

attribute syn_isclock of *object*: *objectType* **is true ;**

where *object* is a black-box input port.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library synplify;
```

```
entity ram4 is
   port (myclk : in bit;
         opcode : in bit_vector(2 downto 0);
         a, b : in bit_vector(7 downto 0);
         rambus : out bit_vector(7 downto 0) );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;

-- Other code
```


Copyright © 2013 Synopsys, Inc.

syn_keep

Directive

Preserves the specified net intact during optimization and synthesis.

Technology	Default Value	Global	Object
All	-	No	Net

Description

With this directive, the tool preserves the net withouth optimizing it away by placing a temporary keep buffer primitive on the net as a placeholder. You can view this buffer in the schematic views (see *Effect of Using syn_keep*, on page 1386 for an example). The buffer is not part of the final netlist, so no extra logic is generated. There are various situations where this directive is useful:

- To preserve a net that would otherwise be removed as a result of optimization. You might want to preserve the net for simulation results or to obtain a different synthesis implementation.
- To prevent duplicate cells from being merged during optimization. You apply the directive to the nets connected to the input of the cells you want to preserve.
- As a placeholder to apply the -through option of the define_multicycle_path or define_false_path timing constraint. This allows you to specify a unique path as a multiple-cycle or false path. Apply the constraint to the keep buffer.
- To prevent the absorption of a register into a macro. If you apply syn_keep to a reg or signal that will become a sequential object, the tool keeps the register and does not absorb it into a macro.
- To prevent the inferencing of altmult_add and altmult_accum (Altera Stratix). Add the syn_keep directive between the adder and the multiplier of a Multiply/Accumulate (MAC) block to prevent this inferencing while preserving the inference of plain multipliers.

Copyright © 2013 Synopsys, Inc.

syn_keep with Multiple Nets in Verilog

In the following statement, syn_keep only applies to the last variable in the wire declaration, which is net c:

```
wire a,b,c /* synthesis syn_keep=1 */;
```

To apply syn_keep to all the nets, use one of the following methods:

- Declare each individual net separately as shown below.

  ```
  wire a /* synthesis syn_keep=1 */;
  wire b /* synthesis syn_keep=1 */;
  wire c /* synthesis syn_keep=1 */;
  ```

- Use Verilog 2001 parenthetical comments, to declare the syn_keep attribute as a single line statement.

  ```
  (* syn_keep=1 *) wire a,b,c;
  ```

  For more information, see *Attribute Examples Using Verilog 2001 Parenthetical Comments*, on page 760.

syn_keep and SystemVerilog Data Types

The SystemVerilog data types behave like logic or reg, and SystemVerilog allows them to be assigned either inside or outside an always block. If you want to use syn_keep to preserve a net with a SystemVerilog data type, like bit, byte, longint or shortint for example, you must make sure that continuous assigns are made inside an always block, not outside.

The following table shows examples of SystemVerilog datatype assignments:

Copyright © 2013 Synopsys, Inc.

Assignment in always block, syn_keep works	`assign keep1_wireand_out;`
	`assign keep2_wireand_out;`
	`always @(*) begin`
	`  keep1_bitand_out;`
	`  keep2_bitand_out;`
	`  keep1_byteand_out;`
	`  keep2_byteand_out;`
	`  keep1_longintand_out;`
	`  keep2_longintand_out;`
	`  keep1_shortintand_out;`
	`  keep2_shortintand_out;`

Assignment outside always block, syn_keep does not work	`assign keep1_wireand_out;`
	`assign keep2_wireand_out;`
	`assign keep1_bitand_out;`
	`assign keep2_bitand_out;`
	`assign keep1_byteand_out;`
	`assign keep2_byteand_out;`
	`assign keep1_longintand_out;`
	`assign keep2_longintand_out;`
	`assign keep1_shortintand_out;`
	`assign keep2_shortintand_out;`

For information about supported SystemVerilog data types, see *Data Types*, on page 769.

Comparison of syn_keep, syn_preserve, and syn_noprune

The following comparison differentiates between syn_keep, syn_preserve, and syn_noprune:

- syn_keep works only on nets and combinational logic. It ensures that the wire is kept during synthesis, and that no optimizations cross the wire. This directive is usually used to prevent unwanted optimizations and to ensure that manually created replications are preserved. When applied to a register, the register is preserved and not absorbed into a macro.
- syn_preserve ensures that registers are not optimized away.
- syn_noprune ensures that a black box is not optimized away when its outputs are unused (i.e., when its outputs do not drive any logic).

Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Example

object **/* synthesis syn_keep = 1 */ ;**

where *object* is a wire or reg declaration. Make sure that there is a space between the object name and the beginning of the comment slash (/).

Here is the source code used to produce the results shown in *Effect of Using syn_keep*, on page 1386.

```
module example2(out1, out2, clk, in1, in2);
output out1, out2;
input clk;
input in1, in2;
wire and_out;
wire keep1 /* synthesis syn_keep=1 */;
wire keep2 /* synthesis syn_keep=1 */;
reg out1, out2;
assign and_out=in1&in2;
assign keep1=and_out;
assign keep2=and_out;

always @(posedge clk)begin;
   out1<=keep1;
   out2<=keep2;
end
endmodule
```

VHDL Syntax and Example

attribute syn_keep of *object* **:** *objectType* **is true ;**

where *object* is a single or multiple-bit signal. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Here is the source code used to produce the schematics shown in *Effect of Using syn_keep*, on page 1386.

```
entity example2 is
   port (in1, in2 : in bit;
         clk : in bit;
         out1, out2 : out bit );
end example2;
```


Copyright © 2013 Synopsys, Inc.

```
architecture rt1 of example2 is
attribute syn_keep : boolean;
signal and_out, keep1, keep2: bit;
attribute syn_keep of keep1, keep2 : signal is true;
begin
and_out <= in1 and in2;
keep1 <= and_out;
keep2 <= and_out;
   process(clk)
   begin
      if (clk'event and clk = '1') then
         out1 <= keep1;
         out2 <= keep2;
      end if;
   end process;
end rt1;
```

Effect of Using syn_keep

When you use syn_keep on duplicate logic, the tool retains it instead of optimizing it away. The following figure shows the Technology view for two versions of a design.

In the first, syn_keep is set on the nets connected to the inputs of the registers out1 and out2, to prevent sharing. The second figure shows the same design without syn_keep. Setting syn_keep on the input wires for the registers ensures that the design has duplicate registered outputs for out1 and out2. If you do not apply syn_keep to keep1 and keep2, the software optimizes out1 and out2, and only has one register.

With syn_keep

Without syn_keep

Copyright © 2013 Synopsys, Inc.

syn_loc

Attribute: Altera, Microsemi, Xilinx. Specifies pin locations for I/O pins and cores, and forward-annotates this information to the place-and-route tool. This attribute can only be specified in a constraint file.

Use this attribute in place of the Xilinx location attribute xc_loc. Use the QSF2SDC utility (*Altera qsf2sdc Utility*, on page 994) to set this attribute when it translates Altera QSF set_location_assignment and set_instance_assignment constraints.

In the syntax, the *pinNumbers* specification is a comma-separated list of pin or placement numbers. Refer to the vendor databook for valid values.

Constraint File Syntax and Examples

define_attribute {*portName*} syn_loc {*pinNumbers*}

The following are examples of using this attribute:

Altera	`define_attribute {DATA0[7:0]} syn_loc {14,12,11,5,21,18,16,15}`
Microsemi	`define_attribute {CR_DIN[3:0]} syn_loc {M7, Y6, B6, D10}`
Xilinx	`define_attribute {DATA0[4:0]} syn_loc {P14,P12,P11,P5,P21}`

You can also specify locations for individual bus bits with this attribute:

Microsemi

```
define_attribute {CR_DIN[3]} syn_loc {M7}
define_attribute {CR_DIN[2]} syn_loc {Y6}
define_attribute {CR_DIN[1]} syn_loc {B6}
define_attribute {CR_DIN[0]} syn_loc {D10}
```

Specify the b: prefix and the bit slice:

```
define_attribute {b:CR_DIN[0]} syn_loc {D10}
define_attribute {b:CR_DIN[1]} syn_loc {B6}
define_attribute {b:CR_DIN[2]} syn_loc {Y6}
define_attribute {b:CR_DIN[3]} syn_loc {M7}
```

Copyright © 2013 Synopsys, Inc.

Altera	Xilinx

`define_attribute {DATA0[7]} syn_loc {14}`
`define_attribute {DATA0[6]} syn_loc {12}`
`define_attribute {DATA0[5]} syn_loc {11}`
`define_attribute {DATA0[4]} syn_loc {5}`
`define_attribute {DATA0[3]} syn_loc {21}`
`define_attribute {DATA0[2]} syn_loc {18}`
`define_attribute {DATA0[1]} syn_loc {16}`
`define_attribute {DATA0[0]} syn_loc {15}`
	Specify the individual bits:
	`define_attribute {valid[0]} syn_loc {R6}`
	`define_attribute {valid[1]} syn_loc {T2}`
	`define_attribute {valid[2]} syn_loc {R5}`
	`define_attribute {valid[3]} syn_loc {R1}`
	
	Specify the b: prefix and the bit slice:
	`define_attribute {b:valid[0]} syn_loc {R6}`
	`define_attribute {b:valid[1]} syn_loc {T2}`
	`define_attribute {b:valid[2]} syn_loc {R5}`
	`define_attribute {b:valid[3]} syn_loc {R1}`

Copyright © 2013 Synopsys, Inc.

syn_looplimit

Directive, VHDL only

Specifies a loop iteration limit for while loops in the design when the loop index is a variable, not a constant. If your design requires a variable loop index, use the syn_looplimit directive to specify a limit for the compiler. If you do not, you can get a "while loop not terminating" compiler error. The limit cannot be an expression. The higher the value you set, the longer the runtime.

Verilog applications use the loop_limit directive (see *loop_limit*, on page 1207).

VHDL Syntax and Example

attribute syn_looplimit : *integer*;
attribute syn_looplimit of *labelName* **: label is** *value*;

The following is an example where the loop limit is set to 5000:

```
library IEEE;
use std.textio.all;
use ieee.std_logic_textio.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity initram is
   port (rAddr, wAddr, dataIn  : in integer;
         clk: in bit;
         we : in bit;
         dataOut  : out integer );
end;

architecture rtl of initram is
subtype smallint is integer range 0 to 3000;
type intAry is array (0 to 3000) of smallint;
function load( name : string) return intAry is
attribute syn_looplimit : integer;
attribute syn_looplimit of myloop: label is 5000;

variable t : intAry ;
variable data : smallint ;
variable dataLine : line ;
variable i : natural ;
file dataFile : text open READ_MODE is name ;
```

```
begin
   myloop: while ( not endfile(dataFile) ) loop
      readline(dataFile,dataLine);
      read(dataLine,data);
      t(i) := data;
      i := i + 1;
   end loop myloop;

return t;
end load;

-----------------------------------------------------

signal ram : intAry := load("data.txt");
signal rAddr_reg : integer ;
begin
   process (clk) begin
      if (clk'event and clk='1') then
         rAddr_reg <= rAddr;
         if(we = '1') then
            ram(wAddr) <= dataIn;
         end if;
      end if;
end process;

dataOut <= ram(rAddr_reg);
end RTL ;
```

The data.txt file in the example is a large data file with each entry representing an iteration for the loop.

Copyright © 2013 Synopsys, Inc.

syn_macro

Attribute

Determines whether instantiated macros are optimized and included in the final output netlist.

Vendor	Devices	Synthesis Support
Xilinx	All	Synplify Pro, Synplify Premier

syn_macro = 1 | 0

Value	Description	Default
0	The Synplify Pro tool does not optimize instantiated primitives whose associated edn, ngo, or ngc files have been added to the project, although it uses the timing and resource utilization information from these files. It does not write out the contents of the macros to the output netlist. The working of syn_macro in the Synplify Premier tool is influenced by the setting of the Enhanced Optimization option: • If the Enhanced Optimization option is disabled, the syn_macro attribute functions just as in the Synplify Pro tool. • If the option is enabled, the tool first optimizes instantiated primitives whose associated edn, ngo, or ngc files have been added to the project. It then includes the modified contents of these primitives in the output netlist.	X
1	The tool preserves the contents of the macro and the macro boundary. This is true in both Synplify Pro and Synplify Premier synthesis, even if the Synplify Premier Enhanced Optimization option is enabled. The tool treats the macro as a white box and only uses the timing and connection information from it. The tool writes out the contents of the macro to the output netlist. For Synplify Pro synthesis, if the macro has an associated ngc file, the contents are never written to the output netlist.	

For a comparison of this behavior with that of syn_user_instance, see *syn_macro Versus syn_user_instance*, on page 1393.

Description

The syn_macro attribute determines whether the macros are optimized and whether the contents of instantiated macros are included in the final output netlist. For Synplify Pro synthesis, the contents of a macro with an associated ngc file are never written to the output netlist.

You can also use this attribute with IP cores (`edn`, `ngo`, or `ngc`). The attribute can only be set on an instantiated macro or IP core in a constraint file, and you must set this attribute on a view, not an instance.

For details on using the syn_macro attribute with the Xilinx IP flow for secure and non-secure cores, see *Including Xilinx Cores for Logic and Physical Synthesis*, on page 1049 in the *User Guide*.

syn_macro Constraint File Syntax

You can only set this attribute in the constraint file, where you set it on the view (`v:`) for the macro or IP core:

define_attribute {v:*object*} syn_macro {1|0}

This example prevents optimizations for the instantiated fifo macro and does not write the contents of the macro to the output netlist.

```
define_attribute {v:fifo} syn_macro {1}
```

If you do not set it on the view, the attribute will not work. The following setting will not work, because it is set on an instance:

```
define_attribute {i:comp_IP1} {syn_macro} {1}
```

syn_macro Versus syn_user_instance

The syn_macro and syn_user_instance attributes (*syn_user_instance*, on page 1585) can have similar results. In Synplify Pro synthesis, there is no conflict as only the syn_macro attribute is available. For Synplify Premier synthesis, where both syn_macro and syn_user_instance are available, use them as follows:

- Use syn_macro to disable optimizations to the core and to prevent the contents of the core from being written to the output netlist.
- Use syn_user_instance to disable optimizations to the core, but still include the contents of the core in the final netlist.

Copyright © 2013 Synopsys, Inc.

If both the syn_macro and syn_user_instance attributes are applied to the same object, the syn_user_instance attribute takes precedence.

The Synplify Premier Enhanced Optimization option setting can also affect the working of the two attributes, as summarized in the table below. The Synplify Pro behavior for syn_macro is the same as the Synplify Premier behavior with Enhanced Optimization disabled.

Attribute	Synplify Pro, or Synplify Premier with Enhanced Optimization Off	Synplify Premier with Enhanced Optimization On
syn_macro = 0	Macro optimization: NO Written to netlist: NO	Macro optimization: YES Written to netlist: YES
syn_macro = 1	Macro optimization: NO Written to netlist: NO	Macro optimization: NO Written to netlist: NO
syn_user_instance = 0	Macro optimization: NO Written to netlist: NO	Macro optimization: YES Written to netlist: YES
syn_user_instance = 1	Macro optimization: NO Written to netlist: YES	Macro optimization: NO Written to netlist: YES

Copyright © 2013 Synopsys, Inc.

syn_map_dffrs

*Attribute; Xilinx Virtex-6 and Spartan-*6. This attribute allows the synthesis software to handle registers with both asynchronous set and asynchronous reset. This attribute is applicable for Xilinx Virtex-6 and Spartan-6 devices only, for which registers with asynchronous set and reset are not supported.

If this attribute is set to 0 when the design contains registers with asynchronous set and reset, the synthesis software generates an error message. You must set the attribute syn_map_dffrs to 1 for the top-level design.

When you set this attribute to 1, instead of generating an error, the software takes either the set or reset and maps it to logic which might affect the quality of results. The default value is 1.

Constraint File Syntax and Example

This attribute is applied globally on the top-level design.

define_global_attribute {syn_map_dffrs} {1}

Verilog Syntax and Example

object **/* synthesis syn_map_dffrs =** 1 ***/ ;**

```
module dff_aload(q, d, ad, clk, load)
   /*synthesis syn_map_dffrs = 1*/;
input d, ad, clk, load;
output q;
reg q;

always @(posedge clk or posedge load)
begin
   if (load)
      q = ad;
   else
      q = d;
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute syn_map_dffrs of *object* **:** *objectType* **is** 1 **;**

```
library ieee ;
use ieee.std_logic_1164.all ;
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY clatches IS
   PORT (d,c,r,s : IN std_logic;
   qrs: OUT std_logic);
attribute syn_map_dffrs : integer;
END clatches;

ARCHITECTURE behave OF clatches IS
   SIGNAL qrs_i: std_logic;
   attribute syn_map_dffrs : boolean;
   attribute syn_map_dffrs of behave : architecture is true;
BEGIN
   qrs_i <= '0' WHEN r = '1' ELSE
   '1' WHEN s = '1' ELSE
      d  WHEN c = '1' ELSE
      qrs_i;
   qrs <= qrs_i;
END BEHAVE;
```

Copyright © 2013 Synopsys, Inc.

syn_maxfan

Attribute

Overrides the default (global) fanout guide for an individual input port, net, or register output.

Vendor	Technology	Default
Altera	All	None
Lattice	ECP3, ECP2S, ECP2M, ECP2 ECP/EC XP2/XP SC/SCM, MachXO, ORCA	None
Microsemi	All	None
Xilinx	All except XC4000	None

syn_maxfan Value

value	Integer for the maximum fanout

Description

syn_maxfan overrides the global fanout for an individual input port, net, or register output. You set the default Fanout Guide for a design through the Device panel on the Implementation Options dialog box or with the set_option -fanout_limit command or -fanout_guide in the project file. Use the syn_maxfan attribute to specify a different (local) value for individual I/Os.

Generally, syn_maxfan and the default fanout guide are suggested guidelines only, but in certain cases they function as hard limits.

- When they are guidelines, the synthesis tool takes them into account, but does not always respect them absolutely. The synthesis tool does not respect the syn_maxfan limit if the limit imposes constraints that interfere with optimization.
- The attribute value functions as a hard limit when it is attached to nets, ports, primitive instances, and registers in Xilinx, Altera, and Microsemi ProASIC^{PLUS} designs. See *Setting Fanout Limits,* on page 444 of the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

You can apply the syn_maxfan attribute to the following objects:

- Registers or instances. For Xilinx, Altera, and Microsemi designs with the Synplify Pro and Synplify Premier tools, you can also apply it to a module or entity. If you attach the attribute to a lower-level module or entity that is subsequently optimized during synthesis, the synthesis tool moves the syn_maxfan attribute up to the next higher level. If you do not want syn_maxfan moved up during optimization, set the syn_hier attribute for the entity or module to hard. This prevents the module or entity from being flattened when the design is optimized.
- Ports or nets. If you apply the attribute to a net, the synthesis tool creates a KEEPBUF component and attaches the attribute to it to prevent the net itself from being optimized away during synthesis.

The syn_maxfan attribute is often used along with the syn_noclockbuf attribute on an input port that you do not want buffered. There are a limited number of clock buffers in a design, so if you want to save these special clock buffer resources for other clock inputs, put the syn_noclockbuf attribute on the clock signal. If timing for that clock signal is not critical, you can turn off buffering completely to save area. To turn off buffering, set the maximum fanout to a very high number; for example, 1000. Note, do not use the syn_maxfan attribute with fast synthesis.

Similarly, you use syn_maxfan with the syn_replicate attribute in certain technologies to control replication.

syn_maxfan Syntax

Global	Object Type
No	Registers, instances, ports, nets

FDC	**define_attribute {***object***} syn_maxfan {***integer***}**	FDC Example
Verilog	*object* **/* synthesis syn_maxfan = "***value***" */ ;**	Verilog Example
VHDL	**attribute syn_maxfan of** *object* **:** *objectType* **is "***value***" ;**	VHDL Example

FDC Example

define_attribute {*object***} syn_maxfan {***integer***}**

Enable	Object Type	Object	**Attribute**	Value	Value Type	Description
✔	<any>	<Global>	syn_maxfan	1	integer	Overrides the defaul...

Verilog Example

object /* **synthesis syn_maxfan = "***value***"** */ **;**

For example:

```
module syn_maxfan (clk,rst,a,b,c);
input clk,rst;
input [7:0] a,b;
output reg [7:0] c;

reg d/* synthesis syn_maxfan=3 */;

always @ (posedge clk)
   begin
      if(rst)
         d <= 0;
      else
         d <= ~d;
   end

always @ (posedge d)
   begin
      c <= a^b;
   end

endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Example

attribute syn_maxfan of *object* **:** *objectType* **is "***value***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity maxfan is
   port ( a : in std_logic_vector(7 downto 0);
      b : in std_logic_vector(7 downto 0);
      rst : in std_logic;
      clk : in std_logic;
      c : out std_logic_vector(7 downto 0) );
end maxfan;

architecture rtl of maxfan is
signal d : std_logic;

attribute syn_maxfan : integer;
attribute syn_maxfan of d : signal is 3;

begin

process (clk)
   begin
   if (clk'event and clk = '1') then
      if (rst = '1') then
      d <= '0';
      else
      d <= not d;
   end if;
   end if;
end process;

process (d)
   begin

   if (d'event and d = '1') then

   c <= a and b;
   end if;
   end process;

end rtl;
```

Effect of Using syn_maxfan

Before applying syn_maxfan:

After applying syn_maxfan:

After applying attribute syn_maxfan, the register “d” replicated three times (shown in red) because its actual fanout is 8, but we have restricted it to 3.

Copyright © 2013 Synopsys, Inc.

Verilog	reg d/* synthesis syn_maxfan=3 */;
VHDL	attribute syn_maxfan of d : signal is 3;

syn_max_memsize_reg

Attribute

Achronix Speedster22iHD

The syn_max_memsize_reg attribute allows you to specify the maximum number of registers that can be mapped to an inferred RAM above the limit for which the tool errors out. The default value is 128.

You can apply the syn_max_memsize_reg attribute globally in the top-level design to change the default threshold to a new value above the limit for which the tool generates an error message.

Constraint File Syntax

define_global_attribute syn_max_memsize_reg { *integer* }

Note: If the syn_ramstyle attribute is set to "registers" for an inferred RAM, then the syn_max_memsize_reg attribute has no effect on the RAM and the tool always maps the RAM to registers.

Copyright © 2013 Synopsys, Inc.

syn_multstyle

Attribute

Determines how multipliers are implemented.

Vendor	Device	Values
Altera	Stratix and Cyclone families	block_mult | logic | lpm_mult
Lattice	ECP3/ECP2S/ECP2M/ECP2 ECP/EC XP2/XP MACH XO, ORCA	block_mult | logic
Microsemi	SmartFusion2 Axcelerator RTAX2000D Axcelerator RTAX4000D	dsp | logic
Xilinx	Virtex-II and newer families	block_mult | logic

syn_multstyle Values

Value	Description	Default
block_mult	*Altera, Lattice, Xilinx* Implements the multipliers as dedicated hardware blocks: • Altera: DSP MACs • Lattice: DSP blocks • Xilinx: Block mults	X
logic	Implements the multipliers as logic.	-
dsp	*Microsemi* Implements the multipliers as DSP blocks.	X
lpm_mult	*Altera* Implements the multipliers as LPMs.	-

Copyright © 2013 Synopsys, Inc.

Altera Stratix and Cyclone families	The default implementation uses DSP MAC blocks. This corresponds to the block_mult value. You can override this behavior by specifying lpm_mult or logic. When you specify: • lpm_mult, the implementation uses LPMs. If you want an lpm_mult implementation, you must explicitly apply the lpm_mult value; if you do not specify this, the software implements the multipliers as alt_mult_add. • logic, it uses logic.
Xilinx Virtex-II and later families	The default implementation uses block multipliers (this corresponds to the attribute value block_mult). You can override this behavior by specifying a value of logic.
Lattice	This attribute only applies to families that use DSP blocks on the device. You can use the block_mult value to implement dedicated hardware DSP blocks. To override this behavior, specify a value of logic.

Description

This attribute specifies whether the multipliers are implemented as dedicated hardware blocks or as logic. The implementation varies with the technology, as shown in the preceding table.

syn_multstyle Syntax

Global Attribute	Object
Yes	Module or instance

The following shows the attribute syntax when specified in different files:

Copyright © 2013 Synopsys, Inc.

FDC	`define_attribute {instance} syn_multstyle {block_mult | logic | dsp | lpm_mult}`
	Global attribute:
	`define_global_attribute syn_multstyle {block_mult | logic | dsp | lpm_mult}`
		SCOPE Example

Verilog	`input net /* synthesis syn_multstyle = "block_mult | logic | dsp | lpm_mult" */;`	Verilog Example
VHDL	`attribute syn_multstyle of instance : signal is "block_mult | logic | dsp | lpm_mult";`	VHDL Example

See VHDL Attribute and Directive Syntax, on page 963 for different ways to specify VHDL attributes and directives.

SCOPE Example

This SCOPE example specifies that the multipliers be globally implemented as logic:

This example specifies that multipliers be implemented as logic.

```
define_attribute {temp[15:0]} syn_multstyle {logic}
```

Verilog Example

```
module mult(a,b,c,r,en);
input [7:0] a,b;
output [15:0] r;
input [15:0] c;
input en;
wire [15:0] temp /* synthesis syn_multstyle="logic" */;
assign temp = a*b;
assign r = en ? temp: c;
endmodule
```

VHDL Example

```
library ieee ;
use ieee.std_logic_1164.all ;
USE ieee.numeric_std.all;

entity mult is
   port (clk : in std_logic ;
      a : in std_logic_vector(7 downto 0) ;
      b : in std_logic_vector(7 downto 0) ;
      c : out std_logic_vector(15 downto 0))
end mult ;

architecture rtl of mult is
signal mult_i : std_logic_vector(15 downto 0) ;
attribute syn_multstyle : string ;
attribute syn_multstyle of mult_i : signal is "logic" ;
begin
mult_i <= std_logic_vector(unsigned(a)*unsigned(b)) ;
   process(clk)
   begin
      if (clk'event and clk = '1') then
         c <= mult_i ;
      end if ;
   end process ;
end rtl ;
```


Copyright © 2013 Synopsys, Inc.

Effect of Using syn_multstyle in an Altera Design

The following figure shows the multiplier implemented as a DSP block in an Altera design, when the attribute is set to block_mult:

Verilog	`wire [15:0] temp /* synthesis syn_multstyle = "block_mult"*/;`
VHDL	`attribute syn_multstyle of mult_i : signal is "block_mult";`

The next figure shows the same Altera design with the multiplier implemented as LPM when the attribute is set to lpm_mult:

Verilog	`wire [15:0] temp /* synthesis syn_multstyle = "lpm_mult"*/;`
VHDL	`attribute syn_multstyle of mult_i : signal is "lpm_mult";`

The following figure shows the Altera design with the multiplier implemented as logic when the attribute is set to logic:

Verilog	`wire [15:0] temp /* synthesis syn_multstyle = "logic"*/;`
VHDL	`attribute syn_multstyle of mult_i : signal is "logic";`

Copyright © 2013 Synopsys, Inc.

The next figure shows the effect of applying the lpm_mult value globally to the Altera design. All four multipliers are implemented as LPMs.

HDL	`module mult (a,b,c,de,e,f,g,h,r1,r2 /* synthesis syn_multstyle = "lpm_mult"*/;`
FDC	`define_global_attribute syn_multstyle {lpm_mult}`

Effect of Using syn_multstyle in a Microsemi Design

In a Microsemi design, you can specify that the multipliers be implemented as logic or as dedicated DSP blocks. The following figure shows a multiplier implemented as DSP:

Verilog	`wire [15:0] temp /* synthesis syn_multstyle = "dsp"*/;`
VHDL	`attribute syn_multstyle of mult_i : signal is "dsp";`

Copyright © 2013 Synopsys, Inc.

The following figure shows the same Microsemi design with the multiplier implemented as logic when the attribute is set to logic:

Verilog	`wire [15:0] temp /* synthesis syn_multstyle = "logic"*/;`
VHDL	`attribute syn_multstyle of mult_i : signal is "logic";`

syn_netlist_hierarchy

Attribute

Determines if the generated netlist is to be hierarchical or flat.

Vendor	Technology
Achronix	Speedster families
Altera	Stratix V, Cyclone V, Arria V, and older families
Lattice	EC, SC, XP, ORCA, isp families
Microsemi	ProASIC, Igloo families
Xilinx	Virtex-7 and older families Spartan-6 and older families

syn_netlist_hierarchy Values

Value	Description	Default
1/true	Allows hierarchy generation	Default
0/false	Flattens hierarchy in the netlist	

Description

A global attribute that controls the generation of hierarchy in the EDIF or VQM output netlist when assigned to the top-level module in your design. The default (1/true) allows hierarchy generation, and setting the attribute to 0/false flattens the hierarchy and produces a completely flattened output netlist.

Syntax Specification

Global	Object
Yes	Module/Architecture

Copyright © 2013 Synopsys, Inc.

FDC	**define_global_attribute syn_netlist_hierarchy {0|1}**	SCOPE Example
Verilog	*object* /* **synthesis syn_netlist_hierarchy = 0|1** */ ;	Verilog Example
VHDL	**attribute syn_netlist_hierarchy of** *object* **:** *objectType* **is true|false ;**	VHDL Example

SCOPE Example

Enable	Object Type	Object	**Attribute**	Value	Value Type	Description
☑	global	<Global>	syn_netlist_hierarchy	1	boolean	Enable hierarchy reconstruction

Verilog Example

```
module fu_add(input a,b,cin,output su,cy);
assign su = a ^ b ^ cin;
assign cy = (a & b) | ((a^b) & cin);
endmodule 4

module rca_adder#(parameter width =4)
   (input[width-1:0]A,B,input CIN,
    output[width-1:0]SU,output COUT );
wire[width-2:0]CY;
fu_add FA0(.su(SU[0]),.cy(CY[0]),.cin(CIN),.a(A[0]),.b(B[0]));
fu_add FA1(.su(SU[1]),.cy(CY[1]),.cin(CY[0]),.a(A[1]),.b(B[1]));
fu_add FA2(.su(SU[2]),.cy(CY[2]),.cin(CY[1]),.a(A[2]),.b(B[2]));
fu_add FA3(.su(SU[3]),.cy(COUT),.cin(CY[2]),.a(A[3]),.b(B[3]));
endmodule

module rp_top#(parameter width =16)
   (input[width-1:0]A1,B1,input CIN1,
    output[width- 1:0]SUM,output COUT1) /*synthesis
      syn_netlist_hierarchy=0*/;
wire[2:0]CY1;
rca_adder RA0 (.SU(SUM[3:0]),.COUT(CY1[0]),.CIN(CIN1),
   .A(A1[3:0]),.B(B1[3:0]));
rca_adder RA1(.SU(SUM[7:4]),.COUT(CY1[1]),.CIN(CY1[0]),
   .A(A1[7:4]),.B(B1[7]));
rca_adder RA2 (.SU(SUM[11:8]),.COUT(CY1[2]),.CIN(CY1[1]),
   .A(A1[11:8]),.B(B1[11:8]));
rca_adder RA3(.SU(SUM[15:12]),.COUT(COUT1),.CIN(CY1[2]),
   .A(A1[15:12]),.B(B1[15:12]));
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;

entity FULLADDER is
   port (a, b, c : in std_logic;
         sum, carry: out std_logic);
end FULLADDER;

architecture fulladder_behav of FULLADDER is
begin
   sum <= (a xor b) xor c ;
   carry <= (a and b) or (c and (a xor b));
end fulladder_behav;

library ieee;
use ieee.std_logic_1164.all;

entity FOURBITADD is
   port (a, b : in std_logic_vector(3 downto 0);
         Cin  : in std_logic;
         sum  : out std_logic_vector (3 downto 0);
         Cout, V : out std_logic );
end FOURBITADD;

architecture fouradder_structure of FOURBITADD is
signal c: std_logic_vector (4 downto 1);
component FULLADDER
   port (a, b, c: in std_logic;
         sum, carry: out std_logic);
end component;
begin
   FA0: FULLADDER
      port map (a(0), b(0), Cin, sum(0), c(1));
   FA1: FULLADDER
      port map (a(1), b(1), C(1), sum(1), c(2));
   FA2: FULLADDER
      port map (a(2), b(2), C(2), sum(2), c(3));
   FA3: FULLADDER
      port map (a(3), b(3), C(3), sum(3), c(4));
   V <= c(3) xor c(4);
   Cout <= c(4);
end fouradder_structure;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity BITADD is
   port (A, B: in std_logic_vector(15 downto 0);
         Cin : in std_logic;
         SUM : out std_logic_vector (15 downto 0);
         COUT: out std_logic );
end BITADD;

architecture adder_structure of BITADD is
attribute syn_netlist_hierarchy : boolean;
attribute syn_netlist_hierarchy of adder_structure:
   architecture is false;
signal C: std_logic_vector (4 downto 1);

component FOURBITADD
   port (a, b: in std_logic_vector(3 downto 0);
         Cin : in std_logic;
         sum : out std_logic_vector (3 downto 0);
         Cout, V: out std_logic);
end component;

begin
   F1: FOURBITADD
      port map (A(3 downto 0),B(3 downto 0),
                Cin, SUM(3 downto 0),C(1) );
   F2: FOURBITADD
      port map (A(7 downto 4),B(7 downto 4),
                C(1), SUM(7 downto 4),C(2) );
   F3: FOURBITADD
      port map (A(11 downto 8),B(11 downto 8),
                C(2), SUM(11 downto 8),C(3) );
   F4: FOURBITADD
      port map (A(15 downto 12),B(15 downto 12),
                C(3), SUM(15 downto 12),C(4) );
   COUT <= c(4);
end adder_structure;
```


Copyright © 2013 Synopsys, Inc.

Effect of Using syn_netlist_hierarchy

Without applying the attribute (default is to allow hierarchy generation) or setting the attribute to 1/true creates a hierarchical netlist.

Verilog

```
output[width-1:0]SUM,output COUT1)
/*synthesis syn_netlist_hierarchy=1*/;
```

VHDL

```
attribute syn_netlist_hierarchy of adder_structure :
architecture is true ;
```

Applying the attribute with a value of 0/false creates a flattened netlist.

Verilog
```
output[width-1:0]SUM,output COUT1)
/*synthesis syn_netlist_hierarchy=0*/;
```

VHDL
```
attribute syn_netlist_hierarchy of adder_structure :
architecture is false ;
```

syn_hier flatten and syn_netlist_hierarchy

The syn_hier=flatten attribute and the syn_netlist_hierarchy=false attributes both flatten hierarchy, but work slightly differently. Use the syn_netlist_hierarchy attribute if you want a completely flattened netlist (this attribute flattens all levels of hierarchy). When you set syn_hier=flatten, you flatten the hierarchical levels below the component on which it is set, but you do not flatten the current hierarchical level where it is set. Refer to *syn_hier*, on page 1344 for information about this attribute.

Copyright © 2013 Synopsys, Inc.

syn_noarrayports

Attribute. Specifies that the ports of a design unit be treated as individual signals (scalars), not as buses (arrays) in the output file.

Constraint File Syntax and Example

define_global_attribute syn_noarrayports {0|1}

For example:

```
define_global_attribute syn_noarrayports {1}
```

Verilog Syntax and Example

object **/* synthesis syn_noarrayports = 0 | 1 ;**

Where *object* is a module declarations. For example:

```
module adder8(cout, sum, a, b, cin)
   /* synthesis syn_noarrayports = 1 */;

// Other code
```

VHDL Syntax and Example

attribute syn_noarrayports of *object* **:** *objectType* **is true | false ;**

where *object* is an architecture name. The data type is Boolean. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

In this example, the ports of adder8 are treated as scalars during synthesis.

```
architecture adder8 of adder8 is
attribute syn_noarrayports : boolean;
attribute syn_noarrayports of adder8 : architecture is true;

-- Other code
```

syn_noclockbuf

Attribute.

Synplify Pro.

Turns off automatic clock buffer usage.

Vendor	Technology
Microsemi	all
Lattice	all, including iCE65/iCE40
Altera	all
Xilinx	all

syn_noclockbuf Values

Value	Description
0/false (Default)	Turns on clock buffering.
1/true	Turns off clock buffering.

Description

The synthesis tool uses clock buffer resources, if they exist in the target module, and puts them on the highest fanout clock nets. You can turn off automatic clock buffer usage by using the syn_noclockbuf attribute. For example, you can put a clock buffer on a lower fanout clock that has a higher frequency and a tighter timing constraint.

You can turn off automatic clock buffering for nets or specific input ports. Set the Boolean value to 1 or true to turn off automatic clock buffering.

You can attach this attribute to a port or net in any hard architecture or module whose hierarchy will not be dissolved during optimization.

Copyright © 2013 Synopsys, Inc.

Constraint File Syntax and Example

Global Support	Object
Yes	module/architecture

define_attribute {*clock_port*} syn_noclockbuf {0|1}

define_global_attribute syn_noclockbuf {0|1}

For example:

```
define_attribute {clk} syn_noclockbuf {1}
define_global_attribute syn_noclockbuf {1}
```

FDC Example

The `syn_noclockbuf` attribute can be applied in the scope window as shown:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑	global	<global>	syn_noclockbuf	1	boolean	Use normal input buffer

Verilog Syntax and Examples

object **/* synthesis syn_noclockbuf = 1 | 0 */ ;**

```
module ckbufg (d,clk,rst,set,q);
input d,rst,set;
input clk   /*synthesis syn_nocclockbuf=1*/;
output reg q;
always@(posedge clk)
begin
if(rst)
q<=0;
else if(set)
q<=1;
else
q<=d;
end
endmodule
```

VHDL Syntax and Examples

attribute syn_noclockbuf of *object* **:** *objectType* **is true | false ;**

```
library IEEE;
use IEEE.std_logic_1164.all;
entity d_ff_srss is
port ( d,clk,reset,set : in STD_LOGIC;
            q : out STD_LOGIC);
attribute syn_noclockbuf: Boolean;
attribute syn_noclockbuf of clk : signal is false;
end d_ff_srss;
architecture d_ff_srss of d_ff_srss is
begin
process(clk)
begin
if clk'event and clk='1' then
if reset='1' then
q <= '0';
elsif set='1' then
q <= '1';
else
q <= d;
end if;
end if;
end process;
end d_ff_srss;
```

Effect of Using syn_noclockbuf

The following graphic shows a design without the `syn_noclockbuf` attribute.

Copyright © 2013 Synopsys, Inc.

Verilog	input clk `/*synthesis syn_noclockbuf=0*/;`
VHDL	`attribute syn_noclockbuf: Boolean;` `attribute syn_noclockbuf of clk : signal is false;`

Global buffers are inferred

The following graphic shows a design with the `syn_noclockbuf` attribute.

Verilog	input clk `/*synthesis syn_noclockbuf=1*/;`
VHDL	`attribute syn_noclockbuf: Boolean;` `attribute syn_noclockbuf of clk : signal is true;`

No global buffers inferred

Global Support

When `syn_noclockbuf` attribute is applied globally, global buffers are inferred by default. If the `syn_noclockbuf` attribute value is set to '1', global buffers are not inferred.

Copyright © 2013 Synopsys, Inc.

HDL	module ckbufg(d1,d2,d3,d4,clk1,clk2,clk3,clk4,rst,set,q1,q2,q3,q4)`/*synthesis syn_noclockbuf=1*/;`
FDC	`define_global_attribute {syn_noclockbuf} {1}`

Before Applying attribute

After applying attribute

syn_noclockpad

Atttibute

Technology	Default Value	Global Attribute	Object
Lattice iCE65/iCE40	-	x	Clock net

When you specify the syn_noclockpad attribute on a clock net connected to an input port, the software creates SB_IO and connects it to an input port. The software infers SB_GB and drives this buffer using the SB_IO connection. The SB_GB buffer is created if the number of global buffers already used in the design satisfies the resource limitation.

Constraint File Syntax and Example

define_attribute {*clock_port*} syn_noclockpad {0|1}

define_global_attribute syn_noclockpad {0|1}

VHDL Syntax

attribute syn_noclockpad : boolean ;
attribute syn_noclockpad of *object* : *object_type* is true | false;

Verilog Syntax and Example

Object /* synthesis syn_noclockpad = "{0|1}" */;

```
module test
   (CLK, CLK2, din1, din2, din3, din4,Q1, Q2, reset);

input CLK /* synthesis syn_noclockpad = 1*/;
input CLK2;
input reset;
input [2:0] din1, din2, din3, din4;
output reg [2:0] Q1, Q2;

wire int_clk_glob;
wire int_clk_core;
reg [2:0] reg_1, reg_2, reg_3, reg_4;


always @(posedge CLK)
```


Copyright © 2013 Synopsys, Inc.

```
begin
   reg_1 <= din1;
   reg_2 <= din2;
   Q1 <= reg_1 + reg_2;
end

always @(posedge CLK2)
begin
   reg_3 <= din3;
   reg_4 <= din4;
   Q2 <= reg_3 + reg_4;
end
endmodule
```

There are two clocks in this example; the syn_noclockpad attribute has been set on CLK so it does not use the buffer SB_GB_IO, but uses SB_IO with SB_GB buffers instead.

syn_no_compile_point

Attribute. Use this attribute when the Automatic Compile Point feature is enabled. The software automatically identifies modules as compile points in the design based on its size, number of I/Os, and hierarchical levels. For details about this feature, see the *The Automatic Compile Point Flow*, on page 1109.

However, if you do NOT want the software to create a compile point for a particular view or module, then apply this attribute. This design view or module is ignored by the Automatic Compile Point software, ensuring that a compile point is not generated for it during synthesis. You must explicitly set this attribute to 1 or true.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute {*object*} syn_no_compile_point {*value*}

Where:

- *object* must be a view with the syntax v:*designViewName*
- *value* must be 1 or true

```
define_attribute {v:fifo} syn_no_compile_point {1}
```

Note that you cannot apply this attribute globally.

Verilog Syntax

object **/* synthesis syn_no_compile_point = "1" */;**

where *object* should be a module and *value* is 1.

```
module prep3 (CLK, RST, IN, OUT)
              /* synthesis syn_no_compile_point="1" */;
// Other code
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax

attribute syn_no_compile_point : boolean ;
attribute syn_no_compile_point of *object* **:** *objectType* **is true;**

where *object* is an architecture name and *value* can only be true.

```
architecture struct of cpu is
attribute syn_no_compile_point : boolean;
attribute syn_no_compile_point of struct: architecture is true;
-- Other code
```

syn_noprune

Directive. Prevents optimizations for instances and black-box modules (including technology-specific primitives) with unused output ports. During optimization, if a module does not drive any logic, it is removed by the synthesis tool. If you want to keep the instance of the module in the design, use the syn_noprune directive on the instance or module along with syn_hier set to hard.

Effects of using syn_noprune

The following figure shows the HDL Analyst view for two versions of a design: one version using syn_noprune on black-box instance U1, one version without syn_noprune.

With syn_noprune

Without syn_noprune

With syn_noprune, module U1 remains in the design. Without syn_noprune the module is optimized away. See the following HDL syntax and example sections for the source code used to generate the schematics above.

Copyright © 2013 Synopsys, Inc.

Usage Compared: syn_keep, syn_preserve, syn_noprune

To understand how the syn_keep, syn_preserve, and syn_noprune directives are used, see *Comparison of syn_keep, syn_preserve, and syn_noprune*, on page 1384.

Using syn_noprune – Example 1

The following examples show the Verilog and VHDL source code used for the previous schematics.

Verilog Syntax and Example

object **/* synthesis syn_noprune = 1 */ ;**

where object is a module declaration or an instance. The data type is Boolean.

```
module top(a1,b1,c1,d1,y1,clk);
output y1;
input a1,b1,c1,d1;
input clk;
wire x2,y2;
reg y1;
syn_noprune u1(a1,b1,c1,d1,x2,y2) /* synthesis syn_noprune=1 */;

always @(posedge clk)
   y1<= a1;

endmodule

module syn_noprune (a,b,c,d,x,y)/* synthesis syn_hier="hard" */;
output x,y;
input a,b,c,d;
endmodule
```

In this example, syn_noprune can be applied in two places, on the module declaration of syn_noprune or in the top-level instantiation. The most common place to use syn_noprune is in the declaration of the module. By placing it here, all instances of the module are protected.

```
module syn_noprune (a,b,c,d,x,y); /* synthesis syn_noprune=1 */;

// Other code
```

Here is an example of using syn_noprune on black-box instances. If your design uses multiple instances with a single module declaration, the synthesis comment must be placed before the comma (,) following the port list for each of the instances.

```
my_design my_design1(out,in,clk_in) /* synthesis syn_noprune=1 */;
my_design my_design2(out,in,clk_in) /* synthesis syn_noprune=1 */;
```

In this example, only the instance my_design2 will be removed if the output port is not mapped.

```
my_design
   my_design1 (out, in, clk_in) /* synthesis syn_noprune=1 */,
   my_design2 (out, in, clk_in),
   my_design3 (out, in, clk_in) /* synthesis syn_noprune=1 */;
```

VHDL Syntax and Example

attribute syn_noprune of *object* **:** *objectType* **is true ;**

where the data type is boolean, and *object* is an architecture, a component, or a label of an instantiated component. See *Architectures*, on page 1434, *Component Declaration*, on page 1434, *Component Instance*, on page 1435, for details of the objects. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;

entity noprune is
   port (a, b, c,d : in std_logic;
         x,y : out std_logic );
end noprune;

architecture behave of noprune is
attribute syn_hier : string;
attribute syn_hier of behave : architecture is "hard" ;
begin
   x <= a and b;
   y <= c and d;
end behave;

library ieee;
use ieee.std_logic_1164.all;
```


Copyright © 2013 Synopsys, Inc.

```
entity top is
   port (a1, b1 : in std_logic;
         c1,d1,clk : in std_logic;
         y1 :out std_logic );
end ;

architecture behave of top is
component noprune
port (a, b, c, d : in std_logic;
      x,y : out std_logic );
end component;

signal x2,y2 : std_logic;
attribute syn_noprune : boolean;
attribute syn_noprune of u1 : label is true;
begin
   u1: noprune port map(a1, b1, c1, d1, x2, y2);
   process begin
      wait until (clk = '1') and clk'event;
      y1 <= a1;
   end process;
end;
```

Architectures

The syn_noprune attribute is normally associated with the names of architectures. Once it is associated, any component instantiation of the architecture (design unit) is protected from being deleted.

```
library synplify;
architecture mydesign of rtl is

attribute syn_noprune : boolean;
attribute syn_noprune of mydesign : architecture is true;

-- Other code
```

Component Declaration

Here is an example:

```
architecture top_arch of top is
component gsr
   port (gsr : in std_logic);
end component;

attribute syn_noprune : boolean;
attribute syn_noprune of gsr: component is true;
```

See *Instantiating Black Boxes in VHDL*, on page 961, for more information.

Component Instance

The syn_noprune attribute works the same on component instances as with a component declaration.

```
architecture top_arch of top is
component gsr
   port (gsr : in bit);
end component;

attribute syn_noprune : boolean;
attribute syn_noprune of u1_gsr: label is true;
```

Using syn_noprune – Example 2

In the following Verilog code example, syn_noprune is applied on both an instance and black box module with unused outputs.

Verilog Example

```
module top
   (input a, b, c, d, e, clk,
   output o1);
reg o2_noprunereg /* synthesis syn_noprune = 1*/ ;
wire o3_wire;
assign o1 = a & b;
always @(posedge clk)
   begin
      o2_noprunereg = c & d & e;
   end
noprune_bb U1 (a, o3_wire) /* synthesis syn_noprune = 1*/ ;
endmodule
module noprune_bb ( input in1, output o1 );
endmodule
```


Copyright © 2013 Synopsys, Inc.

The following Technology views show that the instance and black box module have not been optimized away when their outputs are unused if syn_noprune is applied.

Without syn_noprune

With syn_noprune

VHDL Example

In the following VHDL code example, syn_noprune is applied on both an instance and black box module with unused outputs.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
   port (
   clk : in  std_logic;
   a, b, c, d : in std_logic;
   out_a : out std_logic);
end entity top;

architecture arch of top is
   component noprune_bb
      port(
      din : in std_logic;
      dout : out std_logic);
   end component noprune_bb;

   signal o1_noprunereg : std_logic;
   signal o2_reg : std_logic;

   attribute syn_noprune : boolean;
   attribute syn_noprune of U1: label is true;
   attribute syn_noprune of o1_noprunereg : signal is true;

begin
   process(clk)
      begin
         if rising_edge(clk) then
            o1_noprunereg <= b and  c;
            out_a <= a;
      end if;
   end process;
   U1: noprune_bb port map (d, o2_reg);
end architecture arch;
```

The following Technology views show that the instance and black box module have not been optimized away when their outputs are unused if syn_noprune is applied.

Copyright © 2013 Synopsys, Inc.

Without syn_noprune

With syn_noprune

syn_packer_effort_level

Attribute; Altera Stratix III, Stratix IV; Xilinx Virtex-5, and Virtex-6

Determines utilization by specifying a packing effort level for global placement in Synplify Premier designs. You can use it with the Physical Plus flow. You can only specify this attribute in the constraint file.

The higher the value, the more densely the components are packed. If you have a design with components occupying most of the chip, you can set a higher value so that the tool packs the components and frees up area. Conversely, if you have a congested design, set a lower value to spread out the design and reduce congestion.

Constraint File Syntax

You can only set this attribute globally:

define_global_attribute {*object*} syn_packer_effort_level {*value*}

object	
value	0 to 9 The higher the value, the greater the packing effort, and the more densely the components are packed. Set a higher value to reduce area, and a lower value to reduce congestion. Typical values for medium to high-utilization designs are 2 or 3. If you do not specify the attribute, the tool automatically picks an effort level based on the design.

For example:

```
define_global_attribute syn_packer_effort_level 5
```


Copyright © 2013 Synopsys, Inc.

syn_pad_type

Attribute; Xilinx (Spartan-3/3E, Spartan-6, Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, and Virtex-6) Microsemi (Axcelerator, IGLOO, ProASIC families). Specifies an I/O buffer standard. Refer to *I/O Standards*, on page 489 and to the vendor-specific documentation for a list of I/O buffer standards available for the selected device family. Use this attribute instead of the Xilinx I/O buffer standard attribute xc_padtype.

Constraint File Syntax

define_io_standard -default_*portType***|{***port***} -delay_type** *portType*
syn_pad_type {*io_standard***}**

-default_*portType*	*PortType* can be input, output, or bidir. Setting default_input, default_output, or default_bidir causes all ports of that type to have the same I/0 standard applied to them.
-delay_type *portType*	*PortType* can be input, output, or bidir.
syn_pad_type {*io_standard***}**	Specifies I/O standard (see following table).

Constraint File Examples

To set...	Use this syntax...
The default for all input ports to the AGP1X pad type	`define_io_standard -default_input -delay_type input syn_pad_type {AGP1X}`
All output ports to the GTL pad type	`define_io_standard -default_output -delay_type output syn_pad_type {GTL}`
All bidirectional ports to the CTT pad type	`define_io_standard -default_bidir -delay_type bidir syn_pad_type {CTT}`

Copyright © 2013 Synopsys, Inc.

The following are examples of pad types set on individual ports. Note that you cannot assign pad types to bit slices.

```
define_io_standard {in1} -delay_type input
   syn_pad_type {LVCMOS_15}

define_io_standard {out21} -delay_type output
   syn_pad_type {LVCMOS_33}

define_io_standard {bidirbit} -delay_type bidir
   syn_pad_type {LVTTL_33}
```

Verilog Syntax

*object /** **synthesis syn_pad_type =** *io_standard */* **;**

See the tables under *I/O Standards*, on page 489 for a list of the valid I/O standards.

VHDL Syntax

attribute syn_pad_type of *object*: *objectType* **is** *io_standard*;

See the tables under *I/O Standards*, on page 489 for a list of the valid I/O standards. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Copyright © 2013 Synopsys, Inc.

syn_pipeline

Attribute; Specific Altera, Lattice, Lattice iCE65/iCE40, and Xilinx architectures. Specifies that registers that are outputs of multipliers can be moved to improve frequency. Depending on the criticality of the path, the output register can be either pushed into the multiplier or pushed back to the input side. For Xilinx, you can also pipeline ROMs for the Virtex architectures if there are registers in the RTL code. See *Pipelining Virtex-II Multipliers*, on page 1445, for pipelining of Virtex-II multipliers. Note, do not use the syn_pipeline attribute with the Fast Synthesis flow. For information about using this attribute, see *Pipelining*, on page 426 in the *User Guide*.

Constraint File Syntax and Example

define_attribute {*register*} syn_pipeline {0|1}

For example:

```
define_attribute {res[15:0]} syn_pipeline {1}
```

Verilog Syntax and Example

object **/* synthesis syn_pipeline = 1 | 0 */ ;**

This example shows you how to pipeline a multiplier with syn_pipeline.

```
`define lefta 7
`define leftb 7
module mult(r, a, b, preset, clk);
output [`lefta+`leftb +1:0] r;
input [`lefta:0] a;
input [`leftb:0] b;
input clk;
input preset;
reg [`lefta:0] a_aux;
reg [`leftb:0] b_aux;
reg [`lefta+`leftb+1:0] res /* synthesis syn_pipeline = 1 */;
reg [`lefta+`leftb+1:0] res1 ;
wire [`lefta+`leftb+1:0] res_out1 = a_aux * b_aux;
wire [`lefta+`leftb+1:0] res_out2 = res ;
```

Copyright © 2013 Synopsys, Inc.

```
always @(posedge clk or posedge preset)
begin
if (preset)
   res = 16'hFFF;
else
   res = res_out1;
end

always @(posedge clk or posedge preset)
begin
if (preset)
   res1 = 16'hFFFF;
else
   res1 = res_out2;
end

always @(posedge clk)
begin
   a_aux = a;
   b_aux = b;
end

assign r = res1;
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute syn_pipeline of *object* **:** *objectType* **is true** | **false ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity onereg is port (
   r : out std_logic_vector(31 downto 0);
   a : in std_logic_vector(15 downto 0);
   b : in std_logic_vector(15 downto 0);
   clk , rst : in std_logic );
end onereg;

architecture beh of onereg is
signal temp1, temp2, temp3, wire : std_logic_vector(31 downto 0);
attribute syn_pipeline : boolean;
attribute syn_pipeline of temp1 : signal is true;
attribute syn_pipeline of temp2 : signal is true;
attribute syn_pipeline of temp3 : signal is true;
begin
process(clk,rst)
   begin
   if rst='1' then
      temp1 <= (others => '0');
      temp2 <= (others => '0');
      temp3 <= (others => '0');
   elsif clk'event and clk='1' then
      temp1 <= a * b;
      temp2 <= temp1;
      temp3 <= temp2;
   end if;
end process;

wire <= temp3;
r <= wire;
end beh;
```

Pipelining Virtex-II Multipliers

When pipelining Virtex-II multipliers (MULT18X18), the synthesis tool creates a new pipelined multiplier, MULT18X18S. This is a MULT18X18 with a row of registers at its outputs. Only registers with enable and synchronous clear are combined with a MULT18X18 to become a MULT18X18S.

The pipelining optimization is reported in the log file, along with the number of stages created. The timing model shows that it has embedded registers.

Verilog Example

```
`timescale 1ns/1ns
module test(a, b, out,rst,e,clk);
input signed[9:0]a;
input signed[9:0]b;
input clk;
input rst;
input e;
output signed[19:0]out;
wire signed[19:0]mult_out;
reg signed[19:0]out ;
assign mult_out = a*b;

always @(negedge clk)
begin
   if (rst)
      out <= 0;
   else if(e)
      out <= mult_out;
   else
      out <= out;
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity testmult is
   port (clock, reset, enable : in std_logic;
         in1, in2 : in std_logic_vector(7 downto 0);
         out1 : out std_logic_vector(15 downto 0) );
attribute syn_pipeline : boolean;
attribute syn_pipeline of out1 : signal is true;
end testmult;

architecture beh of testmult is
signal temp : std_logic_vector(15 downto 0);
begin
temp <= in1 * in2 ;
   process (clock)
   begin
      if clock'event and clock = '1' then
         if (reset = '1') then
            out1 <= (others => '0');
         elsif (enable = '1') then
            out1 <= temp;
         end if;
      end if;
   end process;
end beh;
```

syn_placer_effort_level

Attribute; Available for *Xilinx Virtex-5* and *Virtex-6* devices with the *Synplify Premier* tools. You can use this attribute with all the Synplify Premier physical synthesis flows.

By default, syn_placer_effort_level is set to HIGH. The default setting provides maximum placement effort for timing.

Set the syn_placer_effort_level attribute to DECONGEST so that Synplify global placement optimizes the design for routing at the expense of timing. You can use this attribute for designs that are unroutable. Set this attribute to DECONGEST to determine if routing improvements are possible for the design.

Set the syn_placer_effort_level attribute to LARGE_BLOCK so that Synplify global placement only places the FIFO, RAM, and DSP blocks in the design and forward-annotates them for P&R. This can provide quality of results (QoR) improvements for the design.

The attribute can be specified in the PDC file (Physical Plus). This attribute is applied globally for the design.

Constraint File Syntax and Example

The constraint file syntax for the attribute is the following:

define_global_attribute syn_placer_effort_level {DECONGEST}

For example:

```
define_global_attribute syn_placer_effort_level {DECONGEST}
```


Copyright © 2013 Synopsys, Inc.

syn_preserve

Directive. Prevents sequential optimization such as constant propagation, inverter push-through, and FSM extraction.

Use syn_preserve to keep registers for simulation purposes or to preserve the logic of registers driven by a constant 1 or 0. To preserve the associated flip-flop and prevent optimization of the signal, you can set syn_preserve on individual registers or on the module/architecture so that the directive is applied to all registers in the module. For example, assume that the input of a flip-flop is always driven to the same value, such as logic 1. The synthesis tool ties that signal to VCC and removes the flip-flop. Using syn_preserve on the registered signal prevents the removal of the flip-flop.

Another use for this attribute is to preserve a particular state machine. When you enable the symbolic FSM compiler for your entire design and state-machine optimizations are performed, you can use syn_preserve to retain a particular state machine during optimization.

Effects of using syn_preserve

The following figure shows an example where reg1 and out2 are preserved during optimization using syn_preserve.

With syn_preserve

Without syn_preserve

Without syn_preserve, reg1 and reg2 are shared because they are driven by the same source; out2 obtains the result of the AND of reg2 and NOT reg1. This is equivalent to the AND of reg1 and NOT reg1, which is a 0. As this is a constant, register out2 is also removed and output out2 is always 0.

When registers are removed during synthesis, a warning message appears in the log file. For example,

```
@W:...Register bit out2 is always 0, optimizing ...
```

See the HDL syntax and example sections below for the source code used to generate the schematics above.

Usage Compared: syn_keep, syn_preserve, syn_noprune

To understand how the syn_keep, syn_preserve, and syn_noprune directives are used, see *Comparison of syn_keep, syn_preserve, and syn_noprune*, on page 1384.

Verilog Syntax and Examples

object **/* synthesis syn_preserve = 1 */ ;**

where object is a register definition signal or a module.

In the following example, syn_preserve is used on a registered signal so that the flip-flop is not removed and optimization does not occur across the register. This is useful when you are not finished with the design but want to synthesize to find the area utilization.

```
reg foo /* synthesis syn_preserve = 1 */;
```

Following is an example of using syn_preserve for a state register.

```
reg [3:0] curstate /* synthesis syn_preserve = 1 */;
```

The following example shows the Verilog source code used for the schematics at the beginning of the syn_preserve attribute description in .

Copyright © 2013 Synopsys, Inc.

```
module syn_preserve (out1,out2,clk,in1,in2)
   /* synthesis syn_preserve=1 */;
output out1, out2;
input clk;
input in1, in2;
reg out1;
reg out2;
reg reg1;
reg reg2;

always@ (posedge clk)begin
reg1 <= in1 &in2;
reg2 <= in1&in2;
out1 <= !reg1;
out2 <= !reg1 & reg2;
end
endmodule
```

VHDL Syntax and Examples

attribute syn_preserve of *object* **:** *objectType* **is true ;**

where *object* is an output port or an internal signal that holds the value of a state register or architecture. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee, synplify;
use ieee.std_logic_1164.all;

entity simpledff is
   port (q : out std_logic_vector(7 downto 0);
         d : in std_logic_vector(7 downto 0);
         clk : in std_logic );

-- Turn on flip-flop preservation for the q output
attribute syn_preserve : boolean;
attribute syn_preserve of q : signal is true;
end simpledff;

architecture behavior of simpledff is
begin
   process(clk)
   begin
      if rising_edge(clk) then
```

```
      -- Notice the continual assignment of "11111111" to q.
            q <= (others => '1');
         end if;
      end process;
   end behavior;
```

In this example, syn_preserve is used on the signal curstate that is later used in a state machine to hold the value of the state register.

```
architecture behavior of mux is
begin
signal curstate : state_type;
attribute syn_preserve of curstate : signal is true;

-- Other code
```

The following example shows the VHDL source code used for the schematics at the beginning of the syn_preserve attribute description.

```
library ieee;
use ieee.std_logic_1164.all;

entity mod_preserve is
   port (out1 : out std_logic;
         out2 : out std_logic;
         in1,in2,clk : in std_logic );
end mod_preserve;

architecture behave of mod_preserve is
attribute syn_preserve : boolean;
attribute syn_preserve of behave: architecture is true;
signal reg1 : std_logic;
signal reg2 : std_logic;
begin
   process
   begin
      wait until clk'event and clk = '1';
      reg1 <= in1 and in2;
      reg2 <= in1 and in2;
      out1 <= not (reg1);
      out2 <= (not (reg1) and reg2) ;
   end process;
end behave;
```


Copyright © 2013 Synopsys, Inc.

syn_preserve_sr_priority

Attribute; Microsemi ACT 1 and 40MX. Globally implements hardware that forces set/reset flip-flops to honor the priority for the set or reset, as coded in the design. This attribute can increase the area of your design.

Sequential Microsemi components do not have a well defined behavior when both the set and reset signals are active at the same time, but you can have the synthesis tool automatically add hardware to force the priority that you have coded in the source code of your design.

Constraint File Syntax and Example

define_global_attribute syn_preserve_sr_priority {1}

For example:

```
define_global_attribute syn_preserve_sr_priority {1}
```

Most language models for a set/reset level-sensitive latch or D flip-flop define the behavior for this condition. Using this attribute sets the priority of set/reset as it is specified in your HDL source code. The following Verilog code implies that reset has priority over set if both are active:

```
if (reset)
   q=0;
else if (set)
   q=1;
else
   q=d;
end if;
```

Verilog Syntax and Example

object **/* synthesis syn_preserve_sr_priority = 1 */ ;**

The following example sets the syn_preserve_sr_priority attribute on the latch3 module. The value 1 indicates that the attribute is on.

```
module latch3(q,data,set,reset,clk)
   /* synthesis syn_preserve_sr_priority = 1 */;
output q;
input data, clk, set, reset ;
reg q;
```

```
always @(clk or data or set or reset)
begin
   if (reset)
      q = 0;
   else if (set)
      q = 1;
   else if (clk)
      q = data;
   end
endmodule
```

VHDL Syntax and Example

attribute syn_preserve_sr_priority of *object* **:** *objectType* **is true ;**

Where *object* is the entity. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Here is an example:

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
port (data, clk, reset, set : in std_logic;
      qrs: out std_logic );
end dff1;

architecture async_set_reset of dff1 is

-- Set the attribute to "true" for async_set_reset architecture
attribute syn_preserve_sr_priority : boolean;
attribute syn_preserve_sr_priority of async_set_reset :
   architecture is true;
begin
setreset: process (clk, reset, set)
   begin
      if reset = '1' then
         qrs <= '0';
      elsif set = '1' then
         qrs <= '1';
      elsif rising_edge(clk)then
         qrs <= data;
      end if;
   end process setreset;
end async_set_reset;
```


Copyright © 2013 Synopsys, Inc.

syn_probe

Attribute

Inserts probe points for testing and debugging the internal signals of a design.

syn_probe Values

Value	Description
1/true	Inserts a probe, and automatically derives a name for the probe port from the net name.
0/false	Disables probe generation.
portName	Inserts a probe and generates a port with the specified name. If you include empty square brackets, [ ], the probe names are automatically indexed to the net name.

Description

syn_probe works as a debugging aid, inserting probe points for testing and debugging the internal signals of a design, causing them to appear as ports at the top level. Use of this attribute implies the use of syn_keep directive.

You can specify values to name probe ports and assign pins to named ports for selected technologies (Altera and Xilinx). Pin-locking properties of probed nets will be transferred to the probe port and pad. If empty square brackets [] are used, probe names will be automatically indexed, according to the index of the bus being probed.

For Altera and Xilinx families, you can specify pin locations for the probe signals. Pin locations cannot be applied to bus slices.

The table below shows how to apply syn_probe values to nets, buses, and bus slices. It indicates what port names will appear at the top level. When the syn_probe value is 0, probe generation is disabled; when syn_probe is 1, the probe port name is derived from the net name.

Copyright © 2013 Synopsys, Inc.

Net Name	syn_probe Value	Probe Port	Comments
n:ctrl	1	ctrl_probe_1	Probe port name generated by the synthesis tool.
n:ctr	test_pt	test_pt	For string values on a net, the port name is identical to the syn_probe value.
n:aluout[2]	test_pt	test_pt	For string values on a bus slice, the port name is identical to the syn_probe value.
n:aluout[2]	test_pt[ ]	test_pt[2]	The empty square brackets [ ] indicate that port names will be indexed to net names.
n:aluout[2:0]	test_pt[ ]	test_pt[2] test_pt[1] test_pt[0]	The empty square brackets [ ] indicate that port names will be indexed to net names.
n:aluout[2:0]	test_pt	test_pt, test_pt_0, test_pt_1	If a syn_probe value without brackets is applied to a bus, the port names are adjusted.

syn_probe Syntax

Global	Object	Default
No	Net	None

The following table shows the syntax used to define this attribute in different files:

FDC	**define_attribute {n:***netName***} syn_probe {***probePortname***|1|0}**	FDC Example
Verilog	*object* /* **synthesis syn_probe = "***string***" | 1 | 0** */ ;	Verilog Example
VHDL	**attribute syn_probe of** *object* **: signal is "***string***" | 1 | 0 ;**	VHDL Example

Copyright © 2013 Synopsys, Inc.

FDC Example

The following examples insert a probe signal into a net and assign pin locations to the ports.

```
define_attribute {n:inst2.DATA0_*[7]} syn_probe {test_pt[]}
define_attribute {n:inst2.DATA0_*[7]} syn_loc
   {14,12,11,5,21,18,16,15}
```

Enable	Object Type	Object	Attribute	Value	Value Type	Description
☑	<any>	<Global>	syn_probe	1	string	Send a signal to out...

Verilog Example

The following example insert probes on bus alu_tmp [7:0] and assigns pin locations to each of the ports inserted for the probes. The example uses the pin assignments for a Xilinx Virtex device.

```
module alu(out1, opcode, clk, a, b, sel);
output [7:0] out1;
input [2:0] opcode;
input [7:0] a, b;
input clk , sel;
reg [7:0] alu_tmp /* synthesis syn_probe="alu1_probe[]"
   syn_loc="A5,A6,A7,A8,A10,A11,A13,A14" */;
reg [7:0] out1;
// Other code
always @(opcode or a or b or sel)
begin
   case (opcode)
      3'b000:alu_tmp <= a+b;
      3'b000:alu_tmp <= a-b;
      3'b000:alu_tmp <= a^b;
      3'b000:alu_tmp <= sel ? a:b;
      default:alu_tmp <= a|b;
   endcase
end

always @(posedge clk)
out1 <= alu_tmp;
endmodule
```

VHDL Example

The following example inserts probes on bus alu_tmp(7 downto 0) and assigns pin locations to each of the ports inserted for the probes. The example uses the pin assignments for a Xilinx Virtex-4 device.

```
library ieee;
use ieee.std_logic_1164.all;
entity alu is
port ( a : in std_logic_vector(7 downto 0);
      b : in std_logic_vector(7 downto 0);
   opcode : in std_logic_vector(2 downto 0);
      clk : in std_logic;
   out1 : out std_logic_vector(7 downto 0) );
end alu;
architecture rtl of alu is
signal alu_tmp : std_logic_vector (7 downto 0);


attribute syn_probe : string;
attribute syn_probe of alu_tmp : signal is "test_pt";
attribute syn_loc : string;
attribute syn_loc of alu_tmp : signal is
   "A5,A6,A7,A8,A10,A11,A13,A14";


begin
   process (clk)
      begin
         if (clk'event and clk = '1') then
         out1 <= alu_tmp;
         end if;
      end process;
   process (opcode,a,b)
      begin
         case opcode is
         when "000"   =>  alu_tmp <= a and b;
         when "001"   =>  alu_tmp <= a or b;
         when "010"   =>  alu_tmp <= a xor b;
         when "011"   =>  alu_tmp <= a nand b;
         when others  =>  alu_tmp <= a nor b;
      end case;
   end process;


end rtl;
```


Copyright © 2013 Synopsys, Inc.

Effect of Using syn_probe

Before applying syn_probe:

Verilog	reg [7:0] alu_tmp /* synthesis syn_probe="0"*/
VHDL	attribute syn_probe of alu_tmp : signal is "0";

After applying syn_probe with “1”:

Verilog	reg [7:0] alu_tmp /* synthesis syn_probe="1"*/
VHDL	attribute syn_probe of alu_tmp : signal is "1";

Copyright © 2013 Synopsys, Inc.

After applying syn_probe with “test_pt”:

Verilog	reg [7:0] alu_tmp /* synthesis syn_probe="test_pt"*/
VHDL	attribute syn_probe of alu_tmp : signal is "test_pt";

Copyright © 2013 Synopsys, Inc.

After applying syn_probe with “test_pt[]”:

Verilog	reg [7:0] alu_tmp /* synthesis syn_probe="test_pt[]"*/
VHDL	attribute syn_probe of alu_tmp : signal is "test_pt[]";

Copyright © 2013 Synopsys, Inc.

syn_radhardlevel

Attribute

Implements designs with high reliability, using radiation-resistant techniques.

Vendor	Technologies	Tool
Altera	Arria V, Cyclone-III, Cyclone-IV, Cyclone-V Stratix-III, Stratix-IV, Stratix-V	Synplify Premier
Microsemi	Anti-fuse (RT, RH and RD radhard devices) ProASIC3, ProASIC3E, ProASIC3L Axcelerator RT (attribute ignored)	Synplify Pro
Xilinx	Virtex families	Synplify Premier

The applicable syn_radhardlevel options for fault-tolerant design vary, according to the vendor and the technology family. See the following for specific details:

- syn_radhardlevel for Microsemi, on page 1462
- syn_radhardlevel for Altera and Xilinx, on page 1464

syn_radhardlevel for Microsemi

Some high reliability techniques are not available or appropriate for all Microsemi families. Use a design technique that is valid for the project. Contact Microsemi technical support for details.

You can apply syn_radhardlevel globally to the top-level module/architecture or on an individual register output signal (or inferred register in VHDL), and the tool uses the attribute value in conjunction with the Microsemi macro files supplied with the software. For more details about using this attribute, see *Specifying syn_radhardlevel in the Source Code*, on page 791 and *Working with Radhard Designs*, on page 790 in the *User Guide*.

syn_radhardlevel = none | cc | tmr | tmr_cc

none	Default, except for Microsemi Axcelerator RT devices* Uses standard design techniques, and does not insert any triple register logic. *For Microsemi Axcelerator RT devices, the tool implements TMR by default for all registers in the design, and the syn_radhardlevel attribute is ignored.
cc	Microsemi Anti-fuse Implements combinational cells with feedback as storage, rather than flip-flop or latch primitives.
tmr	Microsemi Anti-fuse, ProASIC3, ProASIC3E, ProASIC3L Uses triple module redundancy or triple voting to implement registers. Each register is implemented by three flip-flops or latches that “vote” to determine the state of the register. This option can potentially affect area and timing QoR because of the additional logic inserted, so be sure to check your area and timing goals when you use this option.
tmr_cc	Microsemi Anti-fuse Uses triple module redundancy, where each voting register is composed of combinational cells with feedback rather than flip-flop or latch primitives

syn_radhardlevel Syntax (Microsemi)

Name	Global Attribute	Object
syn_radhardlevel	No	Module, architecture, register Verilog: output signal VHDL: architecture, signal

The following table summarizes the syntax in different files:

FDC	define_attribute {*object*} syn_radhardlevel {none|cc|trmr|tmr_cc}	Constraint File Example, on page 1464
Verilog	*object* /* synthesis syn_radhardlevel = none|cc|trmr|tmr_cc */	Verilog syn_radhardlevel Example, on page 1464
VHDL	attribute syn_rw_conflict_logic : boolean; attribute syn_rw_conflict_logic of *Object* : Object Type is none|cc|trmr|tmr_cc ;	VHDL syn_radhardlevel Example, on page 1464

Copyright © 2013 Synopsys, Inc.

Constraint File Example

```
define_attribute {dataout[3:0]} syn_radhardlevel {tmr}
```

Verilog syn_radhardlevel Example

```
//Top level
module top (clk, dataout, a, b);
input clk;
input a;
input b;
output [3:0] dataout;
M1 inst_M1 (a1, M3_out1, clk, rst, M1_out);
// Other code

//Sub modules subjected to DTMR
module M1 (a1, a2, clk, rst, q)
   /* synthesis syn_radhardlevel="distributed_tmr" */;
input clk;
input signed [15:0] a1,a2;
input clk, rst;
output signed [31:0] q;
// Other code
```

VHDL syn_radhardlevel Example

See *VHDL Attribute and Directive Syntax*, on page 963 for alternate methods for specifying VHDL attributes and directives.

```
library synplify;
architecture top of top is
attribute syn_radhardlevel : string;
attribute syn_radhardlevel of top: architecture is "tmr";

-- Other code
```

syn_radhardlevel for Altera and Xilinx

For Altera and Xilinx designs, this attribute enables triple modular redundancy (TMR) for hierarchical modules in the design and implements distributed TMR. You can apply the syn_radhardlevel attribute at the module (view) level, but you cannot set it globally or on a top-level module. This attribute also enables Duplication with Comparison (DWC) that duplicates modules and compares the output results to determine if a SEU error occurs.

Copyright © 2013 Synopsys, Inc.

syn_radhardlevel = none | distributed_tmr | duplicate_with_compare

none	Default Uses standard design techniques, and does not insert any triple register logic.
distributed_tmr	Altera, Xilinx Triplicates modules and inserts majority voting logic. This option can potentially impact area and timing QoR because of the additional logic inserted, so be sure to check your area and timing goals when you use this option. See *Specifying Local TMR for RAMs, on page 356* in the *User Guide* for a procedure.
duplicate_with_compare	Altera, Xilinx Duplicates modules and compares the outputs of this circuit to determine if a SEU failure occurs. Signal nets are also duplicated to be the same as the original connectivity. However, this option limits the redundancy performed, so it has less impact on area and QoR than distributed TMR.

syn_radhardlevel Syntax (Altera, Xilinx)

Name	Global Attribute	Object
syn_radhardlevel	No	View, module, architecture

The following table summarizes the syntax in different files:

FDC	define_attribute {*object*} syn_radhardlevel {none|distributed_tmr|duplicate_with_compare}	Constraint File Distributed TMR Example, on page 1465
Verilog	*object* /* synthesis syn_radhardlevel = none|distributed_tmr|duplicate_with_compare */	Verilog Distributed TMR Example, on page 1466
VHDL	attribute syn_radhardlevel : boolean; attribute syn_radhardlevel of *Object* : Object Type is none|distributed_tmr|duplicate_with_compare ;	Constraint File Distributed TMR Example, on page 1465

Constraint File Distributed TMR Example

```
define_attribute {v:work.M1} syn_radhardlevel {distributed_tmr}
```


Copyright © 2013 Synopsys, Inc.

Verilog Distributed TMR Example

```
// Top level
module top (clk, dataout, a, b);
input clk;
input a;
input b;
output [3:0] dataout;
M1 inst_M1 (a1, M3_out1, clk, rst, M1_out);

// Other code

//Sub-modules applied with Distributed TMR
module M1 (a1, a2, clk,  rst, q)
   /* synthesis syn_radhardlevel="distributed_tmr" */;
input clk;
input signed [15:0] a1,a2;
input clk, rst;
output signed [31:0] q;

// Other code
```

VHDL Distributed TMR Example

```
library synplify;
architecture inst_rtl of inst is
attribute syn_radhardlevel : string;
attribute syn_radhardlevel of inst_rtl: architecture is
   "distributed_tmr";

-- Other code
```

syn_ramstyle

Attribute.

The syn_ramstyle attribute specifies the implementation to use for an inferred RAM. You can apply syn_ramstyle globally, to a module, or to a RAM instance. To turn off RAM inferencing, set the value to registers.

Values for syn_ramstyle

The values for syn_ramstyle vary with the technology being used. The following table lists all the valid syn_ramstyle values, some of which apply only to certain technologies. See *Values for syn_ramstyle by Technology*, on page 1469 for information about the values and their corresponding RAM implementations in different technologies. For details about using syn_ramstyle, see *Specifying RAM Implementation Styles*, on page 375 in the *User Guide*.

registers	Specifies that an inferred RAM be mapped to registers (flip-flops and logic), not technology-specific RAM resources. If your RAM resources are limited, for whatever reason, you can map additional RAMs to registers instead of RAM resources using this attribute.
block_ram	Specifies that the inferred RAM be mapped to the appropriate vendor-specific memory. It uses the dedicated memory resources in the FPGA. By default, the software uses deep block RAM configurations instead of wide configurations to get better timing results. Using the deeper RAMs reduces the output data delay timing by reducing the MUX logic at the output of the RAMs. With this option, the software also does not use the parity bit for data by default. For a more area-efficient RAM inferencing scheme, use block_ram with power (Xilinx only). For Xilinx Virtex designs, it implements Block SelectRAM+ with additional glue logic to resolve read/write address conflicts.
select_ram	Implements RAMs using the distributed RAM resources in the CLBs. For distributed RAMs, the write operation must be synchronous and the read operation asynchronous.

Copyright © 2013 Synopsys, Inc.

no_rw_check	You can use this option independently or in conjunction with a RAM type value such as M512, or with the power value for supported technologies. For Xilinx Virtex families, this option implements the Virtex Block SelectRAM+, without any glue logic. When you read and write to the same Block SelectRAM+ address, the value of the output is indeterminate. An indeterminate output can create a simulation mismatch between RTL and post-synthesis simulations. When enabled, the synthesis tool does not insert bypass logic around the RAM. If you know your design does not read and write to the same address simultaneously, use no_rw_check to eliminate bypass logic. You can use this option to selectively not insert bypass logic on the RAM for read write checks, which is only used when the Read Write Check on RAM option is turned on (enabled) on the Device tab of the Implementation Options panel. Otherwise, this option is not honored. Use this value when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic. The syn_rw_conflict_logic attribute has similar functionality; see *syn_rw_conflict_logic*, on page 1513 for details about the differences between the two. *Note:* The no_rw_check and rw_check options for the syn_ramstyle attribute are mutually exclusive and must not be used together. Whenever synthesis conflicts exist, the software uses the following order of precedence: first the syn_ramstyle attribute, the syn_rw_conflict_logic attribute, and then the Read Write Check on RAM option on the Implementation Option panel.
power	*Synplify Premier tools for Xilinx designs* When used with the block_ram or no_rw_check values, specifies that the inferred RAM be mapped to power and area-efficient Xilinx memory. The software uses wider block RAM configurations that are more area-efficient, instead of the default deeper RAM configurations.

rw_check

When enabled, the synthesis tool inserts bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. If you know your design has RAM that read and write to the same address simultaneously, use rw_check to insert bypass logic. You can use this option to selectively generate bypass logic on the RAM for read write checks, which is only used when the Read Write Check on RAM option is turned off (disabled) on the Device tab of the Implementation Options panel. Otherwise, this option is not honored.

Note: The no_rw_check and rw_check options for the syn_ramstyle atttribute are mutually exclusive and must not be used together. Whenever synthesis conflicts exist, the software uses the following order of precedence: first the syn_ramstyle attribute, the syn_rw_conflict attribute, and then the Read Write Check on RAM option on the Implementation Option panel.

Values for syn_ramstyle by Technology

The attribute is only available for certain technologies and has different values, depending on the technology. The following table lists the valid values and implementations for each supported technology. For an explanation of the values, see *Values for syn_ramstyle*, on page 1467.

Vendor	Technology	Values	Implementation
Achronix	Speedster22iHD	no_rw_check	RAM implemented without read/write address conflict (glue) logic.
		rw_check	RAM implemented with read/write address conflict (glue) logic

Copyright © 2013 Synopsys, Inc.

Vendor	Technology	Values	Implementation
Altera	Stratix Stratix GX Stratix II Stratix II GX Stratix III Stratix IV Stratix IV GT Stratix V Hardcopy II Cyclone Cyclone II Cyclone III Cyclone IV E Cyclone IV GX Cyclone V Arria GX Arria II GX Arria V		Default: Vendor-specific memories (TriMatrix memories)
		registers	Registers
		block_ram	Vendor-specific memories
		M512	M512 RAMs (Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Arria GX, Arria II GX)
		M4K	M4K RAMs (Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Cyclone, Cyclone II)
		M-RAM	M-RAMs (Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Arria GX, Arria II GX)
		M144K	M144K RAMs (Stratix III, Stratix IV, Stratix IV GT)
		M9K	M9K RAMs (Stratix III, Stratix IV, Stratix IV GT, Cyclone III, Cyclone IV E, Cyclone IV GX)
		MLAB	Memory LAB resources are configured as LUTRAM. (Stratix III, Stratix IV, Stratix IV GT, Stratix V).
		M20K	M20K RAMs (Stratix V, Cyclone V, Arria V)

Copyright © 2013 Synopsys, Inc.

Vendor	Technology	Values	Implementation
	Stratix Stratix GX Stratix II Stratix II GX	block_ram, no_rw_check/ rw_check	Block RAM implemented without/with read or write address conflict (glue) logic.
		M512, no_rw_check/ rw_check	M512 RAM implemented without/with read or write address conflict (glue) logic.
		M4K, no_rw_check/ rw_check	M4K RAM implemented without/with read or write address conflict (glue) logic.
		M-RAM, no_rw_check/ rw_check	M-RAM implemented without/with read or write address conflict (glue) logic.
	Stratix III Stratix IV	block_ram, no_rw_check/ rw_check	Block RAM implemented without/with read or write address conflict (glue) logic.
		M9K, no_rw_check/ rw_check	M9K RAM implemented without/with read or write address conflict (glue) logic.
		M144K, no_rw_check/ rw_check	M144K RAM implemented without/with read or write address conflict (glue) logic.
	Stratix V, Cyclone V, Arria V	block_ram, no_rw_check/ rw_check	Block RAM implemented without/with read or write address conflict (glue) logic.
		M20K, no_rw_check/ rw_check	M20K RAM implemented without/with read or write address conflict (glue) logic.

Copyright © 2013 Synopsys, Inc.

Vendor	Technology	Values	Implementation
Lattice	LatticeECP3/ ECP2/ECP2M/ ECP2S LatticeECP/EC LatticeSC/SCM LatticeXP2/XP MachXO ORCA families		Default: RAMs implemented as synchronous dual-port memory cells.
		registers	RAMs implemented as registers.
		block_ram	RAMs implemented with dedicated RAM resources.
		distributed	RAMs implemented as distributed RAM or PFU resources.
		no_rw_check	RAMs implemented without read/write address conflict (glue) logic.
		rw_check	RAM implemented with read/write address conflict (glue) logic.
Lattice iCE65/iCE40	iCE40 and iCE65 families		Default: vendor-specific memories.
		registers	RAMs implemented as registers.
		block_ram	RAMs implemented with dedicated RAM resources.
		no_rw_check	RAMs implemented without read/write address conflict (glue) logic.
		rw_check	RAM implemented with read/write address conflict (glue) logic.
Microsemi	ProASIC3/ ProASIC3E/ ProASIC3L ProASIC^{plus} Axcelerator		Default: Vendor-specific memories
		registers	Registers
		block_ram	Vendor-specific memories
		no_rw_check	RAM implemented without read/write address conflict (glue) logic.
		rw_check	RAM implemented with read/write address conflict (glue) logic.

Copyright © 2013 Synopsys, Inc.

Vendor	Technology	Values	Implementation
	SmartFusion Fusion IGLOO+/ IGLOO/ IGLOOe		Default: RAMs implemented as registers.
	SmartFusion2	lsram	RAM maps to the RAM1K18 block.
		uram	RAM maps to the RAM64X18 block.
		registers	RAM maps to registers.
Xilinx	Virtex families		Default: block_ram if possible; otherwise select_ram.
		registers	Registers and combinational logic
		block_ram	Block SelectRAM plus read/write address conflict logic
		no_rw_check	Block SelectRAM without read/write address conflict (glue) logic
		rw_check	RAM implemented with read/write address conflict (glue) logic.
		select_ram	Distributed RAM
		block_ram, power	Same as block_ram; power specifies that the inferred RAM be mapped to power and area-efficient Xilinx memory. The software uses wider, more area-efficient block RAMs instead of the default deeper block RAMs.
		no_rw_check, power	Same as no_rw_check; power specifies that the inferred RAM be mapped to power and area-efficient Xilinx memory. The software uses wider, more area-efficient block SelectRAM instead of the default deeper block SelectRAMs.
	Spartan-3	registers	Registers and combinational logic

Copyright © 2013 Synopsys, Inc.

Vendor	Technology	Values	Implementation
		select_ram	Distributed RAM

See also:

- For high reliability support, see *Effect of Using syn_ramstyle for High Reliability*, on page 1478.
- For read/write checks on RAM, see *Effect if Using syn_ramstyle with Read Write Check on RAM*, on page 1475.

Constraint File Syntax and Example

define_attribute {*signalName***[***bitRange***]} syn_ramstyle {***string***}**

define_global_attribute syn_ramstyle {*string***}**

If you edit a constraint file to apply syn_ramstyle, be sure to include the range of the signal with the signal name. For example:

```
define_attribute {mem[7:0]} syn_ramstyle {registers};

define_attribute {mem[7:0]} syn_ramstyle {block_ram};
```

Verilog Syntax and Example

object **/* synthesis syn_ramstyle = "***string***" */ ;**

where *object* is a register definition (ram4) signal. The data type is string.

Here is an example:

```
module ram4 (datain,dataout,clk);
output [31:0] dataout;
input clk;
input [31:0] datain;
reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="block_ram" */;
// Other code
```

VHDL Syntax and Example

attribute syn_ramstyle of *object* **:** *objectType* **is "***string***" ;**

where *object* is a signal that defines a RAM or a label of a component instance. Data type is string. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;

entity ram4 is
   port (d : in std_logic_vector(7 downto 0);
         addr : in std_logic_vector(2 downto 0);
         we : in std_logic;
         clk : in std_logic;
         ram_out : out std_logic_vector(7 downto 0) );
end ram4;

library synplify;
architecture rtl of ram4 is
type mem_type is array (127 downto 0) of std_logic_vector (7
   downto 0);
signal mem : mem_type;
-- mem is the signal that defines the RAM

attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "block_ram";

-- Other code
```

Effect if Using syn_ramstyle with Read Write Check on RAM

When syn_ramstyle is set to rw_check, the synthesis software inserts bypass logic around the RAM to prevent a read/write address collision condition. Conversely, if syn_ramstyle is set to no_rw_check, then the synthesis software does not insert any bypass logic around the RAM to prevent a read/write address collision condition.

Constraint File Syntax and Example

define_attribute {*signalName***[***bitRange***]} syn_ramstyle {***string***}**

define_global_attribute syn_ramstyle {*string***}**

Copyright © 2013 Synopsys, Inc.

When you apply syn_ramstyle, you must include the range of the signal with the signal name. For example:

```
define_attribute {mem[7:0]} syn_ramstyle {rw_check};

define_attribute {mem1[7:0]} syn_ramstyle {no_rw_check};
```

Verilog Syntax and Example

object **/* synthesis syn_ramstyle = "***string***" */ ;**

where *object* is a register definition (ram4) signal. The data type is string.

Verilog Example 1: syn_ramstyle="no_rw_check"

```
module ram4 (datain,dataout,clk);
output [31:0] dataout;
input clk;
input [31:0] datain;
reg [7:0] dataout[31:0]
   /* synthesis syn_ramstyle="no_rw_check" */;
// Other code
```

Verilog Example 2: syn_ramstyle="rw_check"

```
module ram4 (datain,dataout,clk);
output [31:0] dataout;
input clk;
input [31:0] datain;
reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="rw_check" */;
// Other code
```

VHDL Syntax and Example

attribute syn_ramstyle of *object* **: objectType is "***string***" ;**

where *object* is a signal the defines a RAM or label of a component instance. The data type is string.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

VHDL Example 1: syn_ramstyle="no_rw_check"

```
library ieee;
use ieee.std_logic_1164.all;
entity ram4 is
port (d : in std_logic_vector(7 downto 0);
addr : in std_logic_vector(2 downto 0);
we : in std_logic;
clk : in std_logic;
ram_out : out std_logic_vector(7 downto 0) );
end ram4;
library synplify;
architecture rtl of ram4 is
type mem_type is array (127 downto 0) of std_logic_vector
   (7 downto 0);
signal mem : mem_type;
-- mem is the signal that defines the RAM
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "no_rw_check";
-- Other code
```

VHDL Example 2: syn_ramstyle="rw_check"

```
library ieee;
use ieee.std_logic_1164.all;
entity ram4 is
port (d : in std_logic_vector(7 downto 0);
addr : in std_logic_vector(2 downto 0);
we : in std_logic;
clk : in std_logic;
ram_out : out std_logic_vector(7 downto 0) );
end ram4;
library synplify;
architecture rtl of ram4 is
type mem_type is array (127 downto 0) of std_logic_vector
   (7 downto 0);
signal mem : mem_type;
-- mem is the signal that defines the RAM
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "rw_check";
-- Other code
```


Copyright © 2013 Synopsys, Inc.

Effect of Using syn_ramstyle for High Reliability

Fault tolerance for Altera and Xilinx architectures can be provided for the RAM using Triple Modular Redundancy (TMR) and Error Correction Code (ECC).

- When tmr is specified, the synthesis software implements the inferred RAM primitive in triplicate for the design.
- When ecc is specified, the synthesis tool creates the ECC RAM block primitive and implements its associated glue logic (for example, address decoders/encoders or comparators) in triplicate, while keeping the RAM interface the same.

You must run synthesis with the Enhanced Optimization option when using these syn_ramstyle values. For more information, see *Using TMR or ECC for RAMs*, on page 356.

Attribute Parameters

Values for syn_ramstyle are as follows:

Value	Description
ecc	*Synplify Premier tools for Altera Stratix V and Xilinx designs* When ecc is specified, fault tolerance can be provided for the RAM using Error Correction Code (ECC). The synthesis tool creates the ECC RAM block primitive and implements its associated glue logic (for example, address decoders/encoders or comparators) in triplicate, while keeping the RAM interface the same. Note: Synthesis must be run with the Enhanced Optimization option.
tmr	*Synplify Premier tools for Altera and Xilinx designs* When tmr is specified, fault tolerance can be provided for the RAM. The software implements the inferred RAM primitive in triplicate for the design. Note: Synthesis must be run with the Enhanced Optimization option.

Constraint File Syntax and Example

define_attribute {*signalName***[***bitRange***]} syn_ramstyle {***string***}**

define_global_attribute syn_ramstyle {*string***}**

If you edit a constraint file to apply syn_ramstyle, be sure to include the range of the signal with the signal name. For example:

```
define_attribute {mem[7:0]} syn_ramstyle {ecc};
```

The following example applies tmr globally:

```
define_global_attribute syn_ramstyle {tmr};
```

Verilog Syntax and Example

object **/* synthesis syn_ramstyle = "***string***" */ ;**

where *object* is a register definition (ram4) signal. The data type is string.

Here is an example:

```
module ram4 (datain,dataout,clk);
output [31:0] dataout;
input clk;
input [31:0] datain;
reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="ecc" */;
// Other code
```

VHDL Syntax and Example

attribute syn_ramstyle of *object* : *objectType* **is** “tmr”;

where *object* is a register definition (spram) signal. The data type is string.

Here is an example:

```
entity spram is
port (-- Input & Output ports declaration);
end spram;
architecture rtl of spram is
type mem_type is array (31 downto 0) of std_logic_vector(
   7downto 0);
signal mem : mem_type;
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "tmr";
-- Other code
```


Copyright © 2013 Synopsys, Inc.

syn_reduce_controlset_size (Xilinx)

Attribute/Directive;
Xilinx Spartan-6, Virtex-5, and Virtex-6 technologies.

The syn_reduce_controlset_size attribute controls the minimum size of the unique control-set on which control-set optimizations can occur. Control sets are unique combinations of clock, clock-enable, and synchronous -reset/-set signals. Virtex-5 and Virtex-6 based designs can encounter packing problems because these architectures have more registers with the same control signals per slice than the corresponding Virtex-4 based design:

Family	Registers per Slice
Virtex-4	2
Virtex-5	4
Virtex-6 and Spartan-6	8

To help eliminate the packing problems, the control-set optimizations move some or all of the control pins for the registers to the D inputs. By default, the tool picks the right number of control-sets based on QoR, resources available, and the target technology. This attribute lets you override the default settings for the tool. Note that using this attribute can increase the data path delay slightly. Registers on the data path that are timing critical and IOB registers are excluded from this optimization.

This attribute can be applied globally or on the top-level module. You can specify syn_reduce_controlset_size as a compiler directive or as a constraint in the constraint file.

Note: Use the syn_reduce_controlset_size attribute with caution to ensure intended results. If you do not choose a value correctly for this attribute, slice utilization can increase and the design may not fit into the target technology.

Constraint File Syntax and Examples

define_global_attribute syn_reduce_controlset_size {*value*}

define_attribute {v:*object*} syn_reduce_controlset_size {*value*}

Where:

- *object* is the top-level module. This attribute can also be applied globally.
- *value* is the minimum size of the unique control-set on which control-set optimizations can occur.
  - There is no upper limit.
  - The control-set reduction optimizations can be disabled by setting syn_reduce_controlset_size to 0.

For example:

```
define_global_attribute syn_reduce_controlset_size {3}
define_attribute {v:work.test} syn_reduce_controlset_size {3}
```

Note: The syn_direct_enable, syn_direct_set, and syn_direct_reset attributes override the control set reduction optimizations.

Example1

Assuming a design with four registers: FDR1, FDR1, FDR2, FDR2, if syn_reduce_controlset_size is set to 2, then no control-set optimizations occur because all control-sets have 2 registers. However, when syn_reduce_controlset_size is set to 3 or greater all the registers are converted from FDRs to FDs.

Example 2:

Suppose you have five registers with two registers sharing reset signal reset_A and three registers sharing reset signal reset_B. If syn_reduce_controlset_size is set to 2, then no control-set optimizations will occur on either the reset_A control-set or the reset_B control-set, since each of these control-sets has the specified minimum of two registers. However, when syn_reduce_controlset_size is set to 3 or greater the two registers sharing reset_A are converted to FDs and reset_A is implemented in combinatorial logic. The three registers sharing reset_B are unaffected since the reset_B control-set has the specified minimum of three registers.

Copyright © 2013 Synopsys, Inc.

Example 2: Using syn_reduce_controlset_size=2

Example 2: Using syn_reduce_controlset_size=3

Verilog Syntax and Example

object /* **synthesis syn_reduce_controlset_size =** *value* */

Example — Verilog syn_reduce_controlset_size

VHDL Syntax and Example

attribute syn_reduce_controlset_size of *object : objectType* **is** *value***;**

Example — VHDL syn_reduce_controlset_size

Copyright © 2013 Synopsys, Inc.

syn_reduce_controlset_size

Attribute/Directive; Lattice iCE65/iCE40

The syn_reduce_controlset_size attribute controls the minimum size of the unique control-set on which control-set optimizations can occur. Control sets are unique combinations of clock, clock-enable, and synchronous reset/set signals.

To help eliminate the packing problems, the control-set optimizations move some or all of the control pins for the registers to the D inputs. This attribute lets you override the default settings for the tool. Note that using this attribute can increase the data path delay slightly. By default, the software sets the number of control-sets to 8.

This attribute can be applied globally or on the top-level module. You can specify syn_reduce_controlset_size as a compiler directive or as a constraint in the constraint file. This attribute is not applicable for asynchronous set/reset registers.

Note: Use the syn_reduce_controlset_size attribute with caution to ensure intended results. If you do not choose a value correctly for this attribute, PLB utilization can increase and the design may not fit into the target technology. The control-set optimizations are not implemented for registers with timing critical paths and IOB registers.

Constraint File Syntax and Examples

define_global_attribute syn_reduce_controlset_size {*value*}

define_attribute {v:*object*} syn_reduce_controlset_size {*value*}

Where:

- *object* is the top-level module. This attribute can also be applied globally.
- *value* is the minimum size of the unique control-set on which control-set optimizations can occur.
  - There is no upper limit.

– The control-set reduction optimizations can be disabled by setting syn_reduce_controlset_size to 0.

For example:

```
define_global_attribute syn_reduce_controlset_size {3}

define_attribute {v:work.test} syn_reduce_controlset_size {3}
```

Example 1 - Using syn_reduce_controlset_size=2

Suppose you have four registers: SB_DFFSR. with two registers driven by R1 reset and two registers driven by R2 reset. If syn_reduce_controlset_size is set to 2, then no control-set optimizations will occur since all control-sets have two registers. However, when syn_reduce_controlset_size is set to 3 or greater, then all the registers are converted to SB_DFFs.

```
module test(i1, i2, i3, i4, r1, r2, clk, o1, o2, o3, o4)
   /* synthesis syn_useioff = 0 syn_reduce_controlset_size = 2 */;
input i1, i2, i3, i4;
input clk;
input r1, r2;
output o1, o2, o3, o4;

reg reg1, reg2, reg3, reg4;

always@(posedge clk)
if (r1)
   reg1 <= 1'b0;
else
   reg1 <= i1;

always@(posedge clk)
if (r1)
   reg2 <= 1'b0;
else
   reg2 <= i2;

always@(posedge clk)
if (r2)
   reg3 <= 1'b0;
else
   reg3 <= i3;

always@(posedge clk)
if (r2)
```


Copyright © 2013 Synopsys, Inc.

```
      reg4 <= 1'b0;
   else
      reg4 <= i4;

   assign o1 = reg1;
   assign o2 = reg2;
   assign o3 = reg3;
   assign o4 = reg4;
   endmodule
```

Example 2 - Using syn_reduce_controlset_size=3

Suppose you have five registers with two registers sharing reset signal r1 and three registers sharing reset signal r2. If syn_reduce_controlset_size is set to 2, then no control-set optimizations will occur on either the r1 control-set or the r2 control-set, since each of these control-sets has the specified minimum of two registers. However, when syn_reduce_controlset_size is set to 3 the two registers sharing r1 are converted to SB_DFFs and r1 is implemented in combinatorial logic. The three registers sharing r2 are unaffected since the r2 control-set has the specified minimum of three registers.

```
   module test(i1, i2, i3, i4, i5, r1, r2, clk, o1, o2, o3, o4, o5)
      /* synthesis syn_useioff = 0 syn_reduce_controlset_size = 3
      */;input i1, i2, i3, i4, i5;
   input clk;
   input r1, r2;
   output o1, o2, o3, o4, o5;

   reg reg1, reg2, reg3, reg4, reg5;

   always@(posedge clk)
   if (r1)
      reg1 <= 1'b0;
   else
      reg1 <= i1;

   always@(posedge clk)
   if (r2)
      reg2 <= 1'b0;
   else
      reg2 <= i2;

   always@(posedge clk)
   if (r2)
      reg3 <= 1'b0;
   else
```

```
   reg3 <= i3;

always@(posedge clk)
if (r1)
   reg4 <= 1'b0;
else
   reg4 <= i4;

always@(posedge clk)
if (r2)
   reg5 <= 1'b0;
else
   reg5 <= i5;

assign o1 = reg1;
assign o2 = reg2;
assign o3 = reg3;
assign o4 = reg4;
assign o5 = reg5;

endmodule
```

VHDL Syntax and Example

attribute syn_reduce_controlset_size of *object* **:** *objectType* **is** *value***;**

```
Library ieee;
use ieee.std_logic_1164.all;

entity reg_top is
port (clk : in std_logic;
   reset : in std_logic_vector (1 downto 0);
   ina,inb : in std_logic;
   outa,outb: out std_logic);
end;

architecture rtl of reg_top is
signal a_reg,b_reg: std_logic_vector ( 3 downto 0);
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is false;
attribute syn_reduce_controlset_size : integer;
attribute syn_reduce_controlset_size of rtl : architecture is 7;

begin
process (clk,reset)
```


Copyright © 2013 Synopsys, Inc.

```
begin
   if (clk'event and clk='1') then
   if (reset(0)= '1') then
      a_reg(0)<='0';
      b_reg(0)<='0';
      else
         a_reg(0)<=ina;
         b_reg(0)<=inb;
   end if;
end if;
end process;

process (clk,reset)
begin
if (clk'event and clk='1') then
   if (reset(1)= '1') then
      a_reg(1)<='0';
      b_reg(1)<='0';
      else
         a_reg(1)<=a_reg(0);
         b_reg(1)<=b_reg(0);
   end if;
end if;
end process;

process (clk,reset)
begin
if (clk'event and clk='1') then
   if (reset(0)= '0') then
      a_reg(2)<='1';
      b_reg(2)<='1';
      else
         a_reg(2)<=a_reg(1);
         b_reg(2)<=b_reg(1);
   end if;
end if;
end process;

process (clk,reset)
begin
if (clk'event and clk='1') then
   if (reset(0)= '0') then
      a_reg(3)<='1';
      b_reg(3)<='1';
      else
         a_reg(3)<=a_reg(2);
```

Copyright © 2013 Synopsys, Inc.

```
            b_reg(3)<=b_reg(2);

      end if;
   end if;
   end process;

   outa<=a_reg(3);
   outb<=b_reg(3);
   end;
```


Copyright © 2013 Synopsys, Inc.

syn_reference_clock

Attribute

Specifies a clock frequency other than the one implied by the signal on the clock pin of the register.

Vendor	Technology	Default Value	Global	Object
Xilinx	Virtex 6, Virtex 5, older families	-	-	Register
Altera	Stratix 5, Stratix 4, older families	-	-	Register
Lattice	ECP4, ECP3, older families	-	-	Register
Microsemi	SmartFusion4, ProASIC3, older families	-	-	Register
Silicon Blue	iCE65, iCE40	-	-	Register
Achronix	Speedster families	-	-	Register

Description

syn_reference_clock is a way to change clock frequencies other than using the signal on the clock pin. For example, when flip-flops have an enable with a regular pattern, such as every second clock cycle, use syn_reference_clock to have timing analysis treat the flip-flops as if they were connected to a clock at half the frequency.

To use syn_reference_clock, define a new clock, then apply its name to the registers you want to change.

FDC	**define_attribute {*register*} syn_reference_clock {*clockName*}**	FDC Example

FDC Example

define_attribute {*register*} syn_reference_clock {*clockName*}

For example:

```
define_attribute {myreg[31:0]} syn_reference_clock {sloClock}
```

You can also use syn_reference_clock to constrain multiple-cycle paths through the enable signal. Assign the find command to a collection (clock_enable_col), then refer to the collection when applying the syn_reference_clock constraint.

The following example shows how you can apply the constraint to all registers with the enable signal en40:

```
define_scope_collection clock_enable_col {find -seq * -filter
   (@clock_enable==en40)}
define_attribute {$clock_enable_col} syn_reference_clock {clk2}
```

Enable	Object Type	Object	Attribute	Value	Value Type	Description
☑	<any>	<Global>	syn_reference_clock	1	string	Override the default ...

Note: You apply syn_reference_clock only in a constraint file; you cannot use it in source code.

Effect of using syn_reference_clock

Before applying attribute:

```
Performance Summary
*******************


Worst slack in design: 499.379

                   Requested     Estimated     Requested     Estimated                 Clock        Clock
Starting Clock     Frequency     Frequency     Period        Period        Slack       Type         Group
-----------------------------------------------------------------------------------------------------------
clk                2.0 MHz       1609.5 MHz    500.000       0.621         499.379     declared     default_clkgroup_0
ref_clk            1.0 MHz       NA            1000.000      NA            NA          declared     default_clkgroup_1
===========================================================================================================
```

After applying attribute:

Copyright © 2013 Synopsys, Inc.

```
Performance Summary
*******************


Worst slack in design: 999.379

                   Requested     Estimated     Requested     Estimated                 Clock        Clock
Starting Clock     Frequency     Frequency     Period        Period        Slack       Type         Group
-------------------------------------------------------------------------------------------------------------
clk                2.0 MHz       NA            500.000       NA            NA          declared     default_clkgroup_0
ref_clk            1.0 MHz       1609.5 MHz    1000.000      0.621         999.379     declared     default_clkgroup_1
=============================================================================================================
```

syn_rename_module

Directive; The syn_rename_module directive, which is applied through a compiler directives file (cdc), allows a particular module to be renamed. The synthesis tool automatically replicates any necessary registers during optimization when fixing fanouts, packing I/Os, or improving the quality of results. This directive is typically used to rename submodules to avoid possible naming conflicts between a generated name and the original module name.

To illustrate this directive, consider module add with instances add_inst1 and add_inst2. Including the following directive in the cdc file renames the module from add to add_new.

```
define_directive {v:work.add} syn_rename_module {add_new}
```

As a result of the renaming, instances add_inst1 and add_inst2 automatically become instances of module add_new.

.cdc File Syntax

define_directive {*moduleName***} syn_rename_module {***newModuleName***}**

The compiler directives file specifies directives to be added to the source code. During compilation, the tool passes all active cdc files to the compiler. The compiler references the object names in these files with the original RTL objects to assign the corresponding directive. For details about specifying directives with this file, see *Specifying Directives in a CDC File*, on page 248 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

syn_replicate

Attribute; Microsemi, Altera, Lattice iCE65/iCE40, Xilinx. Controls replication. The synthesis tool automatically replicates registers during the following optimization processes: fixing fanouts, packing I/Os, and improving quality of results.

You can use this attribute to disable replication either globally or on a per-register basis. When you disable replication globally, it disables I/O packing and quality-of-results optimizations. The synthesis tool uses only buffering to meet maximum fanout guidelines.

To disable I/O packing on specific registers, set the attribute to 0. Similarly, you can use it on a register between clock boundaries to prevent replication. For example, the synthesis tool replicates a register that is clocked by clk1 but whose fanin cone is driven by clk2, even though clk2 is an unrelated clock in another clock group. By setting the attribute for the register to 0, you can disable replication.

Note, do not use the syn_replicate attribute with the Fast Synthesis flow.

Constraint File Syntax and Example

define_attribute {*object*} syn_replicate {0|1}

define_global_attribute syn_replicate {0|1}

For example, to disable all replication in the design:

```
define_global_attribute syn_replicate {0}
```

Verilog Syntax and Example

object **/* synthesis syn_replicate = 1 | 0 */;**

For example:

```
module norep (Reset, Clk, Drive, OK, ADPad, IPad, ADOut);
input Reset, Clk, Drive, OK;
input [31:0] ADOut;
inout [31:0] ADPad;
output [31:0] IPad;
reg [31:0] IPad;
reg DriveA /* synthesis syn_replicate = 0 */;
assign ADPad = DriveA ? ADOut : 32'bz;

always @(posedge Clk or negedge Reset)
   if (!Reset)
      begin
         DriveA <= 0;
         IPad   <= 0;
      end
   else
      begin
         DriveA <= Drive & OK;
         IPad   <= ADPad;
      end
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute syn_replicate of *object* **:** *objectType* **is true | false ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library IEEE;
use ieee.std_logic_1164.all;

entity norep is
   port (Reset : in std_logic;
         Clk : in std_logic;
         Drive : in std_logic;
         OK : in std_logic;
         ADPad : inout std_logic_vector (31 downto 0);
         IPad : out std_logic_vector (31 downto 0);
         ADOut : in std_logic_vector (31 downto 0) );
end norep;

architecture archnorep of norep is
signal DriveA : std_logic;
attribute syn_replicate : boolean;
attribute syn_replicate of DriveA : signal is false;
begin
ADPad <= ADOut when DriveA='1' else (others => 'Z');
   process (Clk, Reset)
   begin
      if Reset='0' then
         DriveA <= '0';
         IPad <= (others => '0');
      elsif rising_edge(clk) then
         DriveA <= Drive and OK;
         IPad <= ADPad;
      end if;
   end process;
end archnorep;
```

syn_resources

Attribute; Microsemi ProASIC3, ProASIC3E, ProASIC3L, IGLOO, IGLOOe, IGLOO+, and Fusion, Altera, Xilinx. Specifies the resources used inside a black box. It is applied to Verilog black-box modules and VHDL architectures or component definitions. For Altera designs, use this attribute instead of the obsolete altera_area.

The value of the attribute is any combination of the following:

Value	Description
luts=*integer*	number of lookup tables
regs=*integer*	number of registers
blockrams=*integer*	number of RAM resources
blockmults=*integer*	number of block multipliers resources
corecells=*integer*	number of core cells for Microsemi families only.
dsp_blocks=*integer*	Supports number of DSP blocks for: • Altera Stratix families • Xilinx families with DSP and DSP48 blocks

The value listed in the area usage report is the larger of the luts or regs value.

For the Altera Cyclone and Stratix families, the total number of logic elements (LEs) includes those that use only the LUT, those that use only the register, and those that use both. Use the total number of LEs for both the luts and regs values, even if this is larger than the actual number of LUTs or actual number of registers.

The Microsemi families only support resource values of blockrams and corecells.

Constraint File Syntax and Example

define_attribute {v:*moduleName***} syn_resources**
{luts=*integer***|regs=***integer***|blockrams=***integer***|dsp_blocks=***integer***|**
blockmults=*integer***}**

Copyright © 2013 Synopsys, Inc.

define_attribute {v:*moduleName***} syn_resources {blockrams=***integer***|corecells=***integer***}**

You can apply the attribute to more than one kind of resource at a time by separating assignments by a comma (,). For example:

```
define_attribute {v:bb} syn_resources {luts=500, regs=400,
   blockrams=10,blockmults=2}

define_attribute {v:bb} syn_resources {corecells=50,blockrams=20}
```

Verilog Syntax and Example

object **/* synthesis syn_resources = "***value***" */ ;**

In Verilog, you can only attach this attribute to a module. Here is an example:

```
module bb (o,i) /* synthesis syn_black_box syn_resources =
   "luts=500,regs=463,blockrams=10,blockmults=2" */;
input i;
output o;
endmodule

module top_bb (o,i);
input i;
output o;
bb u1 (o,i);
endmodule
```

Verilog Syntax and Example (Microsemi)

object **/* synthesis syn_resources = "***value***" */ ;**

In Verilog, you can only attach this attribute to a module. Here is an example:

```
module bb (o,i) /* synthesis syn_black_box syn_resources =
   "corecells=10,blockrams=5" */;
input i;
output o;
endmodule

module top_bb (o,i);
input i;
output o;
bb u1 (o,i);
endmodule
```

VHDL Syntax and Example

attribute syn_resources of *object* **:** *objectType* **is "***string***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives. In VHDL, this attribute can be placed on either an architecture or a component declaration.

```
architecture top of top is
component decoder
   port (clk : in bit;
         a, b : in bit;
         qout : out bit_vector(7 downto 0) );
end component;

attribute syn_resources : string;
attribute syn_resources of decoder: component is
   "luts=500,regs=463,blockrams=10,blockmults=2";

-- Other code
```

VHDL Syntax and Example (Microsemi)

attribute syn_resources of *object* **:** *objectType* **is "***string***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives. In VHDL, this attribute can be placed on either an architecture or a component declaration.

```
architecture top of top is
component decoder
   port (clk : in bit;
      a, b : in bit;
      qout : out bit_vector(7 downto 0) );
end component;

attribute syn_resources : string;
attribute syn_resources of decoder: component is
   "corecells=500,blockrams=10";

-- Other code
```


Copyright © 2013 Synopsys, Inc.

syn_romstyle (Achronix)

Attribute; Allows you to implement ROM architectures using distributed ROM. Infer ROM architectures using a CASE statement in your code.

For the synthesis tool to implement a ROM, at least half of the available addresses in the CASE statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses.

You can apply the syn_romstyle attribute globally to the design by adding the attribute to the module or entity.

.sdc File Syntax and Example

The constraint file syntax for the attribute is:

define_attribute {*romPrimitive***} syn_romstyle {logic}**

define_global_attribute syn_romstyle {logic}

The following constraint file example implements the signal ROM_bit using logic.

```
define_attribute {ROM_bit[3:0]} syn_romstyle {logic}
```

Verilog Syntax and Examples

object **/* syn_romstyle = " logic" */ ;**

This example implements a ROM structure as logic, which forces an implementation using discrete logic primitives:

```
reg [8:0] z /* synthesis syn_romstyle = "logic" */;
```

VHDL Syntax and Examples

attribute syn_romstyle of *object* **:** *objectType* **is " logic" ;**

See VHDL Attribute and Directive Syntax, on page 963 for different ways to specify VHDL attributes and directives.

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```


Copyright © 2013 Synopsys, Inc.

syn_romstyle (Altera)

Attribute; Altera. You can describe a ROM structure in RTL code using a case statement. By applying the syn_romstyle attribute to the signal output value, you can control if the ROM structure is implemented as discrete logic or technology-specific RAM blocks. By default, small ROMs (less than seven address bits) are implemented as logic, and large ROMs (seven or more address bits) are implemented as RAM.

The following table shows how the syn_romstyle values are interpreted in the Altera technologies. For additional details about ROM inferencing, see *Altera ROMs*, on page 1896.

syn_romstyle Value	Implementation
logic	Discrete logic primitives
block_rom	Maps synchronous ROM to RAM blocks for all Stratix, Cyclone, and Arria device families
block_rom	Maps asynchronous ROM to M9K, M10K, M20K, or M144K RAM for all Stratix and Cyclone device families
MLAB	Maps asynchronous ROM to MLAB resources for Stratix III and newer device families. By default, asynchronous ROM is mapped to logic for Stratix III devices and based on the threshold is mapped to logic or MLAB for Stratix IV devices.

You can apply the syn_romstyle attribute globally to the design by adding the attribute to the module or entity.

Constraint File Syntax and Example

define_attribute {*romPrimitive*} syn_romstyle {logic|block_rom|MLAB|lpm_rom}

define_global_attribute syn_romstyle {logic|block_rom|MLAB|lpm_rom}

For example:

```
define_attribute {rom_dout[7:0]} syn_romstyle {logic}
```

Verilog Syntax and Examples

object **/* synthesis syn_romstyle = "logic | block_rom | MLAB | lpm_rom" */ ;**

The following syntax (syn_romstyle = block_rom) forces an RAM implementation, depending on the technology.

```
reg [3:0] z /* synthesis syn_romstyle = "block_rom" */;
```

The following example shows you how to use the syn_romstyle attribute to implement a ROM structure as logic, which forces an implementation using discrete logic primitives.

```
reg [8:0] z /* synthesis syn_romstyle = "logic" */;
```

This example sets the attribute globally:

```
module rom2(z, a) /* synthesis syn_romstyle = "block_rom" */ ;
```

VHDL Syntax and Examples

attribute syn_romstyle of *object* **:** *objectType* **is "logic | block_rom | MLAB | lpm_rom" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

The following syntax specifies the syn_romstyle attribute with a value of block_rom, which will force a RAM implementation, depending on the technology used.

```
signal z : std_logic_vector(3 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "block_rom";
```

The following example specifies a ROM implementation using discrete logic primitives.

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```

This attribute sets the syn_romstyle attribute globally for the entity:

Copyright © 2013 Synopsys, Inc.

```
entity rom6 is
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "block_rom";
   port (a: in std_logic_vector(3 downto 0);
         z: out std_logic_vector(3 downto 0) );
end rom6;
```

Inferring ROMs (Examples)

The following examples show you how to infer a ROM.

Verilog ROM Example

```
// Sparsely populated ROM
module rom2(z, a);
output [3:0] z;
input [4:0] a;
reg [3:0] z /* synthesis syn_romstyle = "block_rom" */;

always @(a) begin
   case (a)
      5'b00000: z = 4'b1011;
      5'b00001: z = 4'b0001;
      5'b00100: z = 4'b0011;
      5'b00110: z = 4'b0010;
      5'b00111: z = 4'b1110;
      5'b01001: z = 4'b0111;
      5'b01010: z = 4'b0101;
      5'b01101: z = 4'b0100;
      5'b10000: z = 4'b1100;
      5'b10001: z = 4'b1101;
      5'b10010: z = 4'b1111;
      5'b10011: z = 4'b1110;
      5'b11000: z = 4'b1010;
      5'b11010: z = 4'b1011;
      5'b11110: z = 4'b1001;
      5'b11111: z = 4'b1000;
      default: z = 4'b0000;
   endcase
end
endmodule
```

Copyright © 2013 Synopsys, Inc.

Verilog Example for Asynchronous ROM

This example shows an asynchronous ROM that can infer a M9K, M20K, or M144K RAM for Altera Stratix and Cyclone families when the following constraint is set:

define_global_attribute {syn_romstyle} {block_rom}

```
//65X10 SPROM async read
module test (addr,clk,rst,dataout);

parameter addr_width = 6;
parameter data_width = 10;
parameter depth = 64;

input clk,rst;
input [addr_width - 1 : 0] addr;

output [data_width - 1 : 0] dataout;

reg [addr_width - 1 : 0] addr_reg;
reg [data_width - 1 : 0] dataout;
wire [data_width - 1 : 0] out_reg;
reg [data_width - 1 : 0] mem [depth - 1  : 0] ;
initial $readmemb("mem64x10.ini", mem);

assign out_reg = mem[addr_reg];

always@(posedge clk)
if(rst)
   addr_reg <= 6'b0;
else
   addr_reg <= addr;

always@(posedge clk)
if(rst)
   dataout <= 10'b0;
else
   dataout <= out_reg;
endmodule
```

VHDL ROM Example

The following example infers ROMs in Altera technologies. However, for Cyclone and Stratix families, you must register the input or output; otherwise the software does not infer a ROM.

Copyright © 2013 Synopsys, Inc.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity rom6 is
   port (a: in std_logic_vector(3 downto 0);
         z: out std_logic_vector(3 downto 0) );
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "block_rom";
end rom6;

architecture rtl of rom6 is
begin
   process(a)
   begin
      case a is
         when "0000" => z <= "1010";
         when "0001" => z <= "0100";
         when "0010" => z <= "0111";
         when "0011" => z <= "0010";
         when "0100" => z <= "0101";
         when "0101" => z <= "1001";
         when "0110" => z <= "1011";
         when "0111" => z <= "0011";
         when "1000" => z <= "1101";
         when "1001" => z <= "0001";
         when "1010" => z <= "1111";
         when "1011" => z <= "0110";
         when "1100" => z <= "1000";
         when "1101" => z <= "1110";
         when "1110" => z <= "1100";
         when others => z <= "0000";
      end case;
   end process;
end rtl;
```

syn_romstyle (Lattice)

Attribute; LatticeECP3/ECP2S/ECP2M/ECP2, LatticeECP/EC, LatticeXP2/XP, LatticeSC/SCM, and MachXO. Allows you to implement ROM architectures using dedicated or distributed ROM. Infer ROM architectures using a case statement in your code.

For the synthesis tool to implement a ROM, at least half of the available addresses in the case statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses.

You can apply the syn_romstyle attribute globally to the design by adding the attribute to the module or entity.

Constraint File Syntax and Example

The constraint file syntax for the attribute is:

define_attribute {*rom_primitive*} syn_romstyle {block_rom|distributed|logic}

define_global_attribute syn_romstyle {block_rom|distributed|logic}

The following constraint file example implements the signal ROM_bit using Block ROM.

```
define_attribute {ROM_bit[3:0]} syn_romstyle {block_rom}
```

Verilog Syntax and Examples

object **/* syn_romstyle = "block_rom | distributed | logic" */ ;**

This example implements a ROM structure as select_rom, which forces a distributed ROM implementation.

```
reg [3:0] z /* synthesis syn_romstyle = "distributed" */;
```

This example implements a ROM structure as logic, which forces an implementation using discrete logic primitives:

```
reg [8:0] z /* synthesis syn_romstyle = "logic" */;
```

This example implements a ROM structure as block RAM.

Copyright © 2013 Synopsys, Inc.

```
reg [8:0] z /* synthesis syn_romstyle = "block_rom" */;
```

This example sets the attribute globally for the module:

```
module rom2(z, a) /* synthesis syn_romstyle = "block_rom" */ ;
```

VHDL Syntax and Examples

attribute syn_romstyle of *object* **:** *object_type* **is "block_rom | distributed | logic" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Dedicated ROM	`signal z : std_logic_vector(3 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is`
	`"distributed";`

Logic	`signal z : std_logic_vector(8 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is "logic";`

Block RAM	`signal z : std_logic_vector(8 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is`
	`"block_rom";`

Copyright © 2013 Synopsys, Inc.

syn_romstyle Attribute (Lattice iCE65/iCE40)

Attribute; Lattice iCE65/iCE40. Allows you to implement ROM architectures using distributed ROM. Infer ROM architectures using a case statement in your code.

For the synthesis tool to implement a ROM, at least half of the available addresses in the case statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses.

You can apply the syn_romstyle attribute globally to the design by adding the attribute to the module or entity.

Constraint File Syntax and Example

The constraint file syntax for the attribute is:

define_attribute {*romPrimitive*} syn_romstyle {logic|block_rom}

define_global_attribute syn_romstyle {logic|block_rom}

The following constraint file example implements the signal ROM_bit using logic.

```
define_attribute {ROM_bit[3:0]} syn_romstyle {logic}
```

Verilog Syntax and Examples

object **/* syn_romstyle = " logic" */ ;**

This example implements a ROM structure as logic, which forces an implementation using discrete logic primitives:

```
reg [8:0] z /* synthesis syn_romstyle = "logic" */;
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Examples

attribute syn_romstyle of *object* **:** *objectType* **is " logic" ;**

See VHDL Attribute and Directive Syntax, on page 963 for different ways to specify VHDL attributes and directives.

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```

syn_romstyle Attribute (Xilinx)

Attribute; Xilinx. Allows you to implement ROM architectures using distributed ROM. Infer ROM architectures using a case statement in your code.

For the synthesis tool to implement a ROM, at least half of the available addresses in the case statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses.

You can apply the syn_romstyle attribute globally to the design by adding the attribute to the module or entity.

Constraint File Syntax and Example

The constraint file syntax for the attribute is:

define_attribute {*romPrimitive*} syn_romstyle {select_rom|logic|block_rom}

define_global_attribute syn_romstyle {select_rom|logic|block_rom}

The following constraint file example implements the signal ROM_bit using Xilinx SelectROM.

```
define_attribute {ROM_bit[3:0]} syn_romstyle {select_rom}
```

Verilog Syntax and Examples

object **/* syn_romstyle = "select_rom | logic | block_rom" */ ;**

This example implements a ROM structure as select_rom, which forces a distributed ROM implementation.

```
reg [3:0] z /* synthesis syn_romstyle = "select_rom" */;
```

This example implements a ROM structure as logic, which forces an implementation using discrete logic primitives:

```
reg [8:0] z /* synthesis syn_romstyle = "logic" */;
```

This example implements a ROM structure as block RAM.

```
reg [8:0] z /* synthesis syn_romstyle = "block_rom" */;
```


Copyright © 2013 Synopsys, Inc.

This example sets the attribute globally for the module:

```
module rom2(z, a) /* synthesis syn_romstyle = "block_rom" */ ;
```

VHDL Syntax and Examples

attribute syn_romstyle of *object* **:** *objectType* **is "select_rom | logic | block_rom" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Xilinx Select ROM	`signal z : std_logic_vector(3 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is`
	`"select_rom";`

Logic	`signal z : std_logic_vector(8 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is "logic";`

Xilinx block RAM
See *ROM Mapping*, on page 2084 for details
	`signal z : std_logic_vector(8 downto 0);`
	`attribute syn_romstyle : string;`
	`attribute syn_romstyle of z : signal is`
	`"block_rom";`

Copyright © 2013 Synopsys, Inc.

syn_rw_conflict_logic

Attribute/Directive

Infers an appropriate RAM implementation, and controls the insertion of glue logic if the inferred RAM is block RAM.

Vendor	Technologies
Altera	Stratix IV and earlier devices (Use syn_ramstyle=blockram for other Altera devices.)
Xilinx	Virtex-5, Virtex-6, Spartan-6, and earlier devices

syn_rw_conflict = 1 | 0

1	Default	Infers an appropriate RAM implementation and generates glue logic if a block RAM is inferred.
0		Infers an appropriate RAM implementation, but does not infer glue logic for any inferred block RAM. This setting could result in a simulation mismatch if you try to read and write to the same address.

Description

Instead of automatically inferring block RAM, the syn_rw_conflict_logic allows the synthesis tool to choose other RAM implementations as appropriate. If the tool does implement block RAM, this attribute can ensure that the tool does not insert bypass logic around the RAM to prevent a simulation mismatch. Use this attribute only when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic.

When you read and write to the same block RAM address, the value of the output is indeterminate. An indeterminate output can create a simulation mismatch between RTL and post-synthesis simulations.

By default, if block RAM is inferred, the synthesis tools automatically apply glue logic around the RAM to prevent this simulation mismatch. Glue logic creation only applies to block RAM, not for any other RAM implementations.

This attribute can be applied globally or on individual modules or instances. You can specify syn_rw_conflict_logic as a compiler directive or as a constraint in the constraint file.

Copyright © 2013 Synopsys, Inc.

syn_rw_conflict_logic and syn_ramstyle

This attribute is similar in functionality to the syn_ramstyle attribute and its no_rw_check value, which is described in *syn_ramstyle*, on page 1467. The following table summarizes the usage differences between syn_ramstyle and syn_rw_conflict_logic. If there are two conflicting attributes specified, the syn_ramstyle attribute takes precedence over syn_rw_conflict_logic.

Use syn_rw_conflict_logic...	If you know your design does not read and write to the same address simultaneously and you want to allow the tool the freedom to select the RAM implementation.
Use syn_ramstyle with no_rw_check...	If you know your design does not read and write to the same address simultaneously and you want a block RAM implementation. The syn_ramstyle attribute specifies the kind of RAM, and no_rw_check eliminates bypass logic inserted around block RAMs to prevent simulation mismatches.

syn_rw_conflict_logic Syntax

Name	Global Attribute	Object
syn_rw_conflict_logic	Yes	Module or RAM instance

The following table summarizes the syntax in different files:

FDC	define_attribute {*object*} syn_rw_conflict_logic {0|1} define_global_attribute syn_rw_conflict_logic {0|1}	SCOPE Example, on page 1514
Verilog	*object* /* synthesis syn_rw_conflict_logic = 0*/	Example — Verilog syn_rw_conflict_logic
VHDL	attribute syn_rw_conflict_logic : boolean; attribute syn_rw_conflict_logic of *Object* : Object Type is *value* ;	Example — VHDL syn_rw_conflict_logic

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value
1	☑		<global>	syn_rw_conflict_logic	0

Effect of Using syn_rw_conflict_logic

The following figure shows the attribute set to 1. The design shows a block RAM with bypass logic inserted:

Verilog	reg[d_width-1:0]mem[mem_dep-1:0]/*synthesis syn_rw_conflict_logic=1*/;
VHDL	attribute syn_rw_conflict_logic of mem : signal is true ;

The next figure shows the attribute set to 0. The tool does not insert bypass logic with this value:

Copyright © 2013 Synopsys, Inc.

Verilog	reg[d_width-1:0]mem[mem_dep-1:0]/*synthesis syn_rw_conflict_logic=0*/;
VHDL	attribute syn_rw_conflict_logic of mem : signal is false ;

syn_safe_case

Directive.

This directive enables/disables the safe case option. When enabled, the high reliability safe case option turns off sequential optimizations for counters, FSM, and sequential logic to increase the reliability of the circuit. If you set this directive on a module or architecture, the module or architecture is treated as safe and all case statements within it are implemented as safe.

Verilog Syntax and Example

module **/* syn_safe_case =1*/**

For example:

```
module top (input a, output b) /* synthesis syn_safe_case =1 */
```

VHDL Syntax and Example

For VHDL, this is a boolean directive:

attribute syn_safe_case of *architectureName* **: architecture is TRUE;**

For example:

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (a input std_logic;
      b: out std_logic);
end test;

architecture rtl of test is
attribute syn_safe_case: boolean;
attribute syn_safe_case of rtl : architecture is "TRUE";
```


Copyright © 2013 Synopsys, Inc.

syn_sharing

Directive. Enables or disables the sharing of operator resources inside a view during synthesis. Resource sharing reduces area. Values for syn_sharing are on or off.

The syn_sharing directive specifies a compiler-specific optimization, and does not affect the mapper. This means that the mapper can still perform resource sharing optimizations to improve timing, even if syn_sharing is disabled.

You can apply it globally to the whole design using the Resource Sharing switch (Project->Implementation Options->Options, which is on by default. If you disable this global switch, you can use the syn_sharing directive to turn on resource sharing for specific modules or architectures. See *Sharing Resources*, on page 448 in the *User Guide* for details about using resource sharing.

Verilog Syntax and Example

object **/* synthesis syn_sharing = on | off */ ;**

where *object* is a module definition.

```
module my_design(out,in,clk_in) /* synthesis syn_sharing=off */;
// Other code
```

VHDL Syntax and Example

attribute syn_sharing of *object* **:** *objectType* **is " true | false " ;**

where *object* is an architecture name. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity alu is
   port (a, b : in std_logic_vector (7 downto 0);
         opcode: in std_logic_vector (1 downto 0);
         clk: in std_logic;
         result: out std_logic_vector (7 downto 0) );
end alu;

architecture behave of alu is
-- Turn on resource sharing for the architecture.
attribute syn_sharing of behave : architecture is "on";
begin
-- Behavioral source code for the design goes here.
end behave;
```


Copyright © 2013 Synopsys, Inc.

syn_slow

Attribute.

Increases transition time of a specific output port or ports.

Vendor	Devices
Xilinx	Virtex6, Spatan6, older devices

syn_slow Values

Value	Description
1 | True (Default)	Increases the transition time.

Description

The transition time of the output driver can be programmed as either fast or slow. The syn_slow attribute increases the transition time which reduces the noise level in the circuit.

You can speed up the transition time with the syn_fast attribute.

The synthesis tool provides attributes, passed into the XNF/EDIF net list for Xilinx placement and routing, that affect the input setup times and output transition times for your I/Os. The syn_slow timing attribute gets passed to the Xilinx I/O Block parameter as SLOW in the XNF/EDIF netlist.

syn_slow Syntax

Global Support	Object
No	Output Port

Copyright © 2013 Synopsys, Inc.

The following table summarizes the syntax in different files:

FDC	define_attribute {*outputPort*} syn_slow {1}	FDC Example
Verilog	object /* synthesis xc_slow = 1 */ ;	Verilog Example
VHDL	attribute xc_slow of *object* : *objectType* is true ;	VHDL Example

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	☑		p:DATA0[7:0]	xc_fast	1		

Verilog Example

```
module counter (CLK, RST, DATA0);
input CLK, RST;
output [7:0] DATA0 /* synthesis syn_slow = 1 */;
reg [7:0] DATA0 ;
always @(posedge CLK or posedge RST)
begin
      if (RST)
         DATA0 = 0;
      else
         DATA0 = DATA0 + 1;
end
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity count is
port(clk : in std_logic;
      rst : in std_logic;
   data0 : out std_logic_vector(7 downto 0)
   );

end count;
```


Copyright © 2013 Synopsys, Inc.

```
architecture behave of count is
   signal data0_i: std_logic_vector(7 downto 0);
begin
process (rst,clk)
   begin
      if (clk'event and clk = '1') then
            if (rst = '1') then
            data0_i <= "00000000";
      else
         data0_i  <= data0_i + "00000001";
         end if;
         end if;
   end process;

   data0 <= data0_i;
end behave;
```

Effect of Using syn_slow

The following table shows a design without the `syn_slow` attribute.

Verilog

VHDL

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell counter (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT))
           (port CLK (direction INPUT)
 )
           (port RST (direction INPUT))
         )
```

The following table shows a design with the `syn_slow` attribute.

Verilog
```
output [7:0] DATA0 /* synthesis syn_slow = 1 */;
```

VHDL
```
attribute syn_slow of data0 : signal is true;
```

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
     (view verilog (viewType NETLIST)
       (interface
         (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT)
         (property xc_slow (integer 1))
(
```

When place and route is run, a line is added for each output port bit attached to the syn_slow attribute in the following files:

```
<projectdirectory>\rev_1\pr_1\counter.pad(257):P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMOS25*|2|12
|SLOW|||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\counter_pad.txt(258):|P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMOS25*|2|1
2|SLOW||||UNLOCATED |NO|NONE |
<projectdirectory>\rev_1\pr_1\syn_slow.edf(193):(property SLOW (string ""))
```


Copyright © 2013 Synopsys, Inc.

syn_srlstyle

Attribute; *Xilinx (Virtex families), Altera (Stratix families), Lattice (LatticeECP2S/ECP2M/ECP2, LatticeECP/EC, LatticeXP2/XP, Lattice SC/SCM, MachXO, and ORCA families*. Determines how to implement the sequential shift (seqShift or altshift_tap) components. See *Inferring Shift Registers*, on page 405 in the *User Guide* for more information.

Constraint File Syntax

define_attribute {*object*} syn_srlstyle
{registers|select_srl|noextractff_srl|altshift_tap,*groupName*}

define_global_attribute syn_srlstyle
{registers|select_srl|noextractff_srl|altshift_tap,*groupName*}

The syn_srlstyle values are:

Technology	Value	Implements...
Xilinx	registers	seqShift register components as registers.
	select_srl	seqShift components using 16-bit shift register lookup table primitives (SRL16) with flip-flops inserted ahead of the output buffers for improved timing.
	noextractff_srl	seqShift components using 16-bit shift register lookup table primitives (SRL16) without output flip-flops.
	If the syn_srlstyle attribute is not specified, then the default behavior depends on timing at the output of the SRL. For timing results with very positive slack, syn_srlstyle=noextractff_srl behavior is invoked. Otherwise, the software implements syn_srlstyle=select_srl behavior.	
Lattice	registers	seqShift register components as registers. Lattice devices only support the value of registers.

Copyright © 2013 Synopsys, Inc.

Technology	Value	Implements...
Altera Stratix families	altshift_tap *groupName*	shift registers using the Altera Stratix altshift_tap (AST) megafunction. If you specify an optional *groupName*, the software ensures that shift registers with different group names are not packed together. In the Synplify Pro/Synplify Premier tool, you can also use other attributes to control packing, as described in *Inferring Shift Registers*, on page 405 in the *User Guide*.
Altera Stratix and Cyclone device families	block_ram	shift registers are mapped to M9K, M20K, or M144K memories.
Altera Stratix III and later device families	MLAB	shift registers are mapped to MLAB resources.
	registers	seqShift or altshift_tap register components as registers.

Here is a constraint file example:

```
define_attribute {regBank[15:0]} syn_srlstyle {registers}
```

In the HDL file, you must apply the syn_srlstyle attribute on the final stage of the shift register. In the following example, apply the syn_srlstyle attribute on register pll_status_ck245_s. The constraint is not honored if it is placed on other registers in the shifting chain.

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
   port (pll_status, lbdr_clk : in std_logic;
   pll_status_ck245_s: out std_logic );
attribute syn_srlstyle : string;
attribute syn_srlstyle of pll_status_ck245_s : signal is
"registers";
end test;

architecture behave of test is
signal pll_status_ck245_r : std_logic;
signal pll_status_ck245_r1 : std_logic;

begin
```


Copyright © 2013 Synopsys, Inc.

```
resynchro_ck245_reg: process(lbdr_clk)
BEGIN
if_clk: IF lbdr_clk'EVENT AND lbdr_clk = '1' THEN
   pll_status_ck245_r <= pll_status;
   pll_status_ck245_r1 <= pll_status_ck245_r;
   pll_status_ck245_s <= pll_status_ck245_r1;
END IF if_clk;
END PROCESS resynchro_ck245_reg;

end behave;
```

Verilog Syntax and Example

object **/* synthesis syn_srlstyle =**
"select_srl | registers | noextractff_srl|altshift_tap|block_ram|MLAB" */ ;

In the above syntax, *object* is a register declaration.

This example implements seqShift components as SRL16 without output flip-flops.

```
module test_srl(clk, enable, dataIn, result, addr);
input clk, enable;
input [3:0] dataIn;
input [3:0] addr;
output [3:0] result;
reg [3:0] regBank[15:0]
   /* synthesis syn_srlstyle="noextractff_srl" */;
integer i;

always @(posedge clk) begin
   if (enable == 1) begin
      for (i=15; i>0; i=i-1) begin
         regBank[i] <= regBank[i-1];
      end
   regBank[0] <= dataIn;
   end
end
assign result = regBank[addr];
endmodule
```

The following example implements a seqShift for 16x256 bits wide and serial in and serial out register using syn_srlstyle set to MLAB.

```
// shift left register with 16X256 bits width and serial in and
serial out
module test(clock, arst, sr_en, shiftin, shiftout);

parameter sh_len=16;
parameter sh_width=256;
parameter ARESET_VALUE = {(sh_width){1'b0}};

input clock,arst,sr_en;
input [sh_width-1:0] shiftin;
output [sh_width-1:0] shiftout;
integer i;

reg [sh_width-1:0] sreg [sh_len-1:0] /* synthesis
syn_srlstyle="MLAB" */;

always @(posedge clock or posedge arst)
   begin /* synthesis loop_limit 17000 */
      if(arst)
      begin
         for(i = 0;i <= sh_len-1;i = i+1)
            sreg[i] <= ARESET_VALUE ;
      end
   else
      begin /* synthesis loop_limit 17000 */
      if(sr_en)
         begin
            sreg[0] <= shiftin;
            for(i=sh_len-1;i>0;i=i-1)
            sreg[i] <= sreg[i-1];
         end
      end
   end

assign shiftout = sreg[sh_len-1];
endmodule
```

Resource report in the `srr` file contains:

Memory ALUTs: 256

VHDL Syntax and Example

attribute syn_srlstyle of *object* **: signal is**
"select_srl | registers | noextractff_srl|altshift_tap|block_ram|MLAB" ;

Copyright © 2013 Synopsys, Inc.

In the above syntax, *object* is a register. See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

The example below implements seqShift components as SRL16 primitives without inserting timing improvement flip-flops between the SRL outputs and the output buffers.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity d_p is
   port (clk : in std_logic;
         data_out : out std_logic_vector(127 downto 0));
end d_p;

architecture rtl of d_p is
type dataAryType is array(3 downto 0) of
   std_logic_vector(127 downto 0);
signal h_data_pip_i : dataAryType;
attribute syn_srlstyle : string;
attribute syn_srlstyle of h_data_pip_i : signal
is "noextractff_srl";
begin
   process (Clk)
   begin
      if (Clk'Event And Clk = '1') then
         h_data_pip_i <= (h_data_pip_i(2 DOWNTO 0)) &
         h_data_pip_i(3);
      end if;
   end process;
data_out <= h_data_pip_i(0);
end rtl;
```

syn_state_machine

Directive. Enables/disables state-machine optimization on individual state registers in the design. When you disable the FSM Compiler, state-machines are not automatically extracted. To extract some state machines, use this directive with a value of 1 on just those individual state-registers to be extracted. Conversely, when the FSM Compiler is enabled and there are state machines in your design that you do not want extracted, use syn_state_machine with a value of 0 to override extraction on just those individual state registers.

Also, when the FSM Compiler is enabled, all state machines are usually detected during synthesis. However, on occasion there are cases in which certain state machines are not detected. You can use this directive to declare those undetected registers as state machines.

The following figure shows an example of two implementations of a state machine: one with the syn_state_machine directive enabled, the other with the directive disabled.

Copyright © 2013 Synopsys, Inc.

See the following HDL syntax and example sections for the source code used to generate the schematics above. See also:

- *syn_encoding*, on page 1305 for information on overriding default encoding styles for state machines.
- For VHDL designs, *syn_encoding Compared to syn_enum_encoding*, on page 1317 for usage information about these two directives.

Verilog Syntax and Examples

object **/* synthesis syn_state_machine = 0 | 1 */ ;**

where *object* is a state register. Data type is Boolean: 0 does not extract an FSM, 1 extracts an FSM.

Following is an example of syn_state_machine applied to register OUT.

```
module prep3 (CLK, RST, IN, OUT);
input CLK, RST;
input [7:0] IN;
output [7:0] OUT;
reg [7:0] OUT;
reg [7:0] current_state /* synthesis syn_state_machine=1 */;

// Other code
```

Here is the source code used for the example in the previous figure.

```
module FSM1 (clk, in1, rst, out1);
input        clk, rst, in1;
output [2:0] out1;

`define s0 3'b000
`define s1 3'b001
`define s2 3'b010
`define s3 3'bxxx

reg [2:0] out1;
reg [2:0] state /* synthesis syn_state_machine = 1 */;
reg [2:0] next_state;

always @(posedge clk or posedge rst)
   if (rst) state <= `s0;
   else     state <= next_state;
```

```
// Combined Next State and Output Logic
always @(state or in1)
   case (state)
      `s0 : begin
         out1 <= 3'b000;
         if (in1) next_state <= `s1;
         else next_state <= `s0;
      end
      `s1 : begin
         out1 <= 3'b001;
         if (in1) next_state <= `s2;
         else next_state <= `s1;
      end
      `s2 : begin
         out1 <= 3'b010;
         if (in1) next_state <= `s3;
         else next_state <= `s2;
      end
      default : begin
         out1 <= 3'bxxx;
         next_state <= `s0;
      end
   endcase
endmodule
```

VHDL Syntax and Examples

attribute syn_state_machine of *object* **:** *objectType* **is true|false ;**

where ***object*** is a signal that holds the value of the state machine. For example:

```
attribute syn_state_machine of current_state: signal is true;
```

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Copyright © 2013 Synopsys, Inc.

Following is the source code used for the example in the previous figure.

```
library ieee;
use ieee.std_logic_1164.all;

entity FSM1 is
   port (clk,rst,in1 : in std_logic;
         out1 : out std_logic_vector (2 downto 0) );
end FSM1;

architecture behave of FSM1 is
type state_values is ( s0, s1, s2,s3 );
signal state, next_state: state_values;
attribute syn_state_machine : boolean;
attribute syn_state_machine of state : signal is false;

begin
   process (clk, rst)
   begin
      if rst = '1' then
         state <= s0;
      elsif rising_edge(clk) then
         state <= next_state;
      end if;
   end process;

   process (state, in1) begin
      case state is
         when s0 =>
            out1 <= "000";
            if in1 = '1' then next_state <= s1;
               else next_state <= s0;
            end if;
         when s1 =>
            out1 <= "001";
            if in1 = '1' then next_state <= s2;
               else next_state <= s1;
            end if;
         when s2 =>
            out1 <= "010";
            if in1 = '1' then next_state <= s3;
               else next_state <= s2;
            end if;
         when others =>
            out1 <= "XXX"; next_state <= s0;
      end case;
   end process;
end behave;
```

syn_tco<*n*>

Directive. Used with the syn_black_box directive; supplies the clock to output timing-delay through a black box.

The syn_tco<n> directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Constraint File Syntax and Example

The syn_tco<*n*> directive can be entered as an attribute using the Attribute panel of the SCOPE editor. The information in the object, attribute, and value fields must be manually entered. This is the constraint file syntax for the directive:

define_attribute {v:*blackboxModule***} syn_tco***n* **{**[**!**]*clock***->***bundle***=***value***}**

For details about the syntax, see the following table:

v:	Constraint file syntax that indicates that the directive is attached to the view.
blackboxModule	The symbol name of the black-box.
n	A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.
!	The optional exclamation mark indicates that the clock is active on its falling (negative) edge.
clock	The name of the clock signal.
bundle	A bundle is a collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. To assign values to bundles, use the following syntax. The values are in ns. [**!**]*clock***->***bundle***=***value*
value	Clock to output delay value in ns.

Constraint file example:

```
define_attribute {v:RCV_CORE} syn_tco1 {CLK-> R_DATA_OUT[63:0]=20}
define_attribute {v:RCV_CORE) syn_tco2 {CLK-> DATA_VALID=30}
```


Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Example

object /* **syn_tco***n* **= "[!]***clock* **->** *bundle* **=** *value***"** */ **;**

See *Constraint File Syntax and Example*, on page 1533 for syntax explanations. The following example defines syn_tco<*n*> and other black-box constraints:

```
module ram32x4(z,d,addr,we,clk);
/* synthesis syn_black_box syn_tco1="clk->z[3:0]=4.0"
   syn_tpd1="addr[3:0]->z[3:0]=8.0"
   syn_tsu1="addr[3:0]->clk=2.0"
   syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

VHDL Syntax and Examples

attribute syn_tco*n* **of** *object* **:** *objectType* **is "[!]***clock* **->** *bundle* **=** *value***" ;**

In VHDL, there are ten predefined instances of each of these directives in the synplify library: syn_tpd1, syn_tpd2, syn_tpd3, … syn_tpd10. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10. For example:

```
attribute syn_tco11 : string;
attribute syn_tco12 : string;
```

See *Constraint File Syntax and Example*, on page 1533 for other syntax explanations.

See *VHDL Attribute and Directive Syntax*, on page 963 for alternate methods for specifying VHDL attributes and directives.

The following example defines syn_tco<*n*> and other black-box constraints:

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
component Dpram10240x8
```

```
   port (
-- Port A
      ClkA, EnA, WeA: in  std_logic;
      AddrA : in  std_logic_vector(13 downto 0);
      DinA  : in  std_logic_vector(7 downto 0);
      DoutA : out std_logic_vector(7 downto 0);
-- Port B
      ClkB, EnB: in  std_logic;
      AddrB : in  std_logic_vector(13 downto 0);
      DoutB : out std_logic_vector(7 downto 0) );
end component;

attribute syn_black_box : boolean;
attribute syn_tsu1      : string;
attribute syn_tsu2      : string;
attribute syn_tco1      : string;
attribute syn_tco2      : string;
attribute syn_isclock   : boolean;
attribute syn_black_box of Dpram10240x8 : component is true;
attribute syn_tsu1 of Dpram10240x8 : component is
   "EnA,WeA,AddrA,DinA -> ClkA = 3.0";
attribute syn_tco1 of Dpram10240x8 : component is
   "ClkA -> DoutA[7:0] = 6.0";
attribute syn_tsu2 of Dpram10240x8 : component is
   "EnB,AddrB -> ClkB = 3.0";
attribute syn_tco2 of Dpram10240x8 : component is
   "ClkB -> DoutB[7:0] = 13.0";

-- Other code
```

Verilog-Style Syntax in VHDL for Black Box Timing

In addition to the syntax used in the code above, you can also use the following Verilog-style syntax to specify black-box timing constraints:

```
attribute syn_tco1 of inputfifo_coregen : component is
   "rd_clk->dout[48:0]=3.0";
```


Copyright © 2013 Synopsys, Inc.

syn_tpd<*n*>

Directive. Used with the syn_black_box directive; supplies information on timing propagation for combinational delay through a black box.

The syn_tpd<n> directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Constraint File Syntax and Example

You can enter the syn_tpd<*n*> directive as an attribute using the Attribute panel of the SCOPE editor. The information in the object, attribute, and value fields must be manually entered. This is the constraint file syntax:

define_attribute {v:*blackboxModule***} syn_tpd***n* **{***bundle***->***bundle***=***value***}**

For details about the syntax, see the following table:

v:	Constraint file syntax that indicates that the directive is attached to the view.
blackboxModule	The symbol name of the black-box.
n	A numerical suffix that lets you specify different input to output timing delays for multiple signals/bundles.
bundle	A bundle is a collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. The values are in ns. **"***bundle***->***bundle***=***value***"**
value	Input to output delay value in ns.

Constraint file example:

```
define_attribute {v:MEM} syn_tpd1 {MEM_RD->DATA_OUT[63:0]=20}
```

Verilog Syntax and Example

*object /** **syn_tpd***n* **= "***bundle* **->** *bundle* **=** *value***"** **/* **;**

See *Constraint File Syntax and Example*, on page 1536 for an explanation of the syntax. This is an example of syn_tpd<*n*> along with some of the other black-box timing constraints:

```
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
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Examples

attribute syn_tpd*n* **of** *object* **:** *objectType* **is "***bundle* **->** *bundle* **=** *value***" ;**

In VHDL, there are 10 predefined instances of each of these directives in the synplify library, for example: syn_tpd1, syn_tpd2, syn_tpd3, … syn_tpd10. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10. For example:

```
attribute syn_tpd11 : string;
attribute syn_tpd11 of bitreg : component is
   "di0,di1 -> do0,do1 = 2.0";
attribute syn_tpd12 : string;
attribute syn_tpd12 of bitreg : component is
   "di2,di3 -> do2,do3 = 1.8";
```

See *Constraint File Syntax and Example*, on page 1536 for an explanation of the syntax.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

The following is an example of assigning syn_tpd<*n*> along with some of the black box constraints. See *Verilog-Style Syntax in VHDL for Black Box Timing*, on page 1535 for another way.

```
-- A USE clause for the Synplify Attributes package was included
-- earlier to make the timing constraint definitions visible here.
architecture top of top is
component rcf16x4z
   port (ad0, ad1, ad2, ad3 : in std_logic;
         di0, di1, di2, di3 : in std_logic;
         clk, wren, wpe : in std_logic;
         tri : in std_logic;
         do0, do1, do2, do3 : out std_logic );
end component;

attribute syn_tpd1 of rcf16x4z : component is
   "ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is
   "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcf16x4z : component is
   "ad0,ad1,ad2,ad3 -> clk = 1.2";
attribute syn_tsu2 of rcf16x4z : component is
   "wren,wpe -> clk = 0.0";
-- Other code
```

syn_tristate

Directive. Specifies that an output port, on a module defined as a black box, is a tristate. Use this directive to eliminate multiple driver errors if the output of a black box has more than one driver. A multiple driver error is issued unless you use this directive to specify that the outputs are tristate.

Verilog Syntax and Examples

object **/* synthesis syn_tristate = 1 */ ;**

where *object* can be black-box output ports. For example:

```
module BUFE(O, I, E); /* synthesis syn_black_box */
   output O /* synthesis syn_tristate = 1 */;

// Other code
```


Copyright © 2013 Synopsys, Inc.

syn_tristatetomux

Attribute; Xilinx (Virtex and Spartan families). Analyzes each net in the design that is driven by tristate drivers and converts the tristate drivers to multiplexers. The software converts the tristates to muxes when the number of tristates drivers on the net is less than the value of the integer argument. All drivers to be converted to multiplexers must be tristate drivers, otherwise the conversion fails and the software issues a warning.

The synthesis process cannot precisely emulate a tristate condition with a mux. Therefore, during tristate-to-mux conversion, the default behavior of the output when all the enables are disabled is zero. There is no user control over this.

For Xilinx designs, all the internal tristates along a critical path are converted to muxes. This is done to avoid probable routing problems and to improve performance. You cannot turn off the conversion, but you can use this attribute to relax the constraints.

The conversion of tristates to muxes varies with the technology, as shown in the following table:

Technology	syn_tristatetomux Behavior
Virtex-4 Virtex-5 Virtex-6 Spartan-3/3E Spartan-6	All tristates are converted to logic, regardless of the attribute value.
Virtex-II Virtex-II Pro	If you do not specify the attribute, the software converts all tristates to logic. If you specify the attribute, the software only coverts those tristate-driven nets that meet the threshold criteria. If the threshold is set to 0, the software does not convert any tristates.
Spartan-II Spartan-IIE	If you do not specify the attribute, the software only converts tristates on the critical path to logic. It does not change tristates on non-critical paths. If you specify the attribute, the software only coverts those tristate-driven nets that meet the threshold criteria. If the threshold is set to 0, the software does not convert any tristates.

Copyright © 2013 Synopsys, Inc.

Constraint File Syntax and Example

define_attribute {*object*} syn_tristatetomux {*integer*}

For example:

```
define_attribute {v:tristate2} syn_tristatetomux {4}
```

Verilog Syntax and Example

object **/* synthesis syn_tristatetomux =** *integer* ***/ ;**

For example:

```
module tristate2 (input0, input1, input2, input3, enable, qout)
   /* synthesis syn_tristatetomux = 4 */;
input [7:0] input0, input1, input2, input3;
input [3:0] enable;
output [7:0] qout;
wire [7:0] temp;
assign temp = (enable[0]) ? input0 : 8'hZZ;
assign temp = (enable[1]) ? input1 : 8'hZZ;
assign temp = (enable[2]) ? input2 : 8'hZZ;
assign temp = (enable[3]) ? input3 : 8'hZZ;
assign qout = temp & input0;
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute syn_tristatetomux of *object* **:** *objectType* **is** *integer* **;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

Here is a VHDL example of multiple tristate drivers on a bus.

```
library ieee;
library synplify;

use ieee.std_logic_1164.all;
entity tristate2 is
   port (input3,
         input2,
         input1,
         input0: in std_logic_vector (7 downto 0);
         enable : in std_logic_vector (3 downto 0);
         qout : out std_logic_vector (7 downto 0) );
end tristate2;

architecture multiple_drivers of tristate2 is
attribute syn_tristatetomux : integer;
attribute syn_tristatetomux of multiple_drivers :
   architecture is 4;
signal temp : std_logic_vector (7 downto 0);

begin
   temp <= input3 when enable(3) = '1'
      else "ZZZZZZZZ" ;

   temp <= input2 when enable(2) = '1'
      else "ZZZZZZZZ" ;

   temp <= input1 when enable(1) = '1'
      else "ZZZZZZZZ" ;

   temp <= input0 when enable(0) = '1'
      else "ZZZZZZZZ" ;
   qout <= temp and input0;
end multiple_drivers;
```

syn_tsu<*n*>

Directive. Used with the syn_black_box directive; supplies information on timing setup delay required for input pins (relative to the clock) in the black box.

The syn_tsu<n> directive is one of several directives that you can use with the syn_black_box directive to define timing for a black box. See *syn_black_box*, on page 1235 for a list of the associated directives.

Constraint File Syntax and Example

The syn_tsu<*n*> directive can be entered as an attribute using the Attribute panel of the SCOPE editor. The information in the object, attribute, and value fields must be manually entered. The constraint file syntax for the directive is:

define_attribute {v:*blackboxModule***} syn_tsu***n* **{***bundle***->**[**!**]*clock***=***value***}**

For details about the syntax, see the following table:

v:	Constraint file syntax that indicates that the directive is attached to the view.
blackboxModule	The symbol name of the black-box.
n	A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.
bundle	A collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. The values are in ns. This is the syntax to define a bundle: *bundle***->[!]***clock***=***value***
!	The optional exclamation mark indicates that the clock is active on its falling (negative) edge.
clock	The name of the clock signal.
value	Input to clock setup delay value in ns.

Constraint file example:

```
define_attribute {v:RTRV_MOD} syn_tsu4 {RTRV_DATA[63:0]->!CLK=20}
```


Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Example

object /* **syn_tsu***n* **= "***bundle* **->** [**!**]*clock* **=** *value***"** */* **;**

For syntax explanations, see *Constraint File Syntax and Example*, on page 1543.

This is an example that defines syn_tsu<*n*> along with some of the other black-box constraints:

```
module ram32x4(z,d,addr,we,clk);
/* synthesis syn_black_box syn_tpd1="addr[3:0]->z[3:0]=8.0"
   syn_tsu1="addr[3:0]->clk=2.0" syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

VHDL Syntax and Examples

attribute syn_tsu*n* **of** *object* **:** *objectType* **is "***bundle* **->** [**!**]*clock* **=** *value***" ;**

In VHDL, there are 10 predefined instances of each of these directives in the synplify library, for example: syn_tsu1, syn_tsu2, syn_tsu3, … syn_tsu10. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10:

```
attribute syn_tsu11 : string;
attribute syn_tsu11 of bitreg : component is
   "di0,di1 -> clk = 2.0";
attribute syn_tsu12 : string;
attribute syn_tsu12 of bitreg : component is
   "di2,di3 -> clk = 1.8";
```

For other syntax explanations, see *Constraint File Syntax and Example*, on page 1543.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives. For this directive, you can also use the following Verilog-style syntax to specify it, as described in *Verilog-Style Syntax in VHDL for Black Box Timing*, on page 1535.

The following is an example of assigning syn_tsu<*n*> along with some of the other black-box constraints:

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
component rcf16x4z
   port (ad0, ad1, ad2, ad3 : in std_logic;
         di0, di1, di2, di3 : in std_logic;
         clk, wren, wpe : in std_logic;
         tri : in std_logic;
         do0, do1, do2, do3 : out std_logic );
end component;

attribute syn_tco1 of rcf16x4z : component is
   "ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is
   "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcf16x4z : component is
   "ad0,ad1,ad2,ad3 -> clk = 1.2";
attribute syn_tsu2 of rcf16x4z : component is
   "wren,wpe -> clk = 0.0";
-- Other code
```


Copyright © 2013 Synopsys, Inc.

syn_unconnected_inputs

Attribute; You can specify that input pins of an instance be left floating (unassigned in the RTL).

If an instance has a floating input pin, the synthesis software does not tie it to GND in the output netlist. However when the input pin is driven by a wire that subsequently is unassigned, then the synthesis tool ties it to GND. Connecting floating input pins to GND causes routing problems in the ACE place-and-route tool. Use the syn_unconnected_inputs attribute to ensure that input pins with undriven wires are left floating. This attribute can only be specified in the constraint file.

Constraint File Syntax and Example

Specify the syntax for this attribute in the constraint file as follows:

define_attribute {*InstanceName***} syn_unconnected_inputs {***string***}**

Verilog Example

```
define_attribute {i:U2.UI} syn_unconnected_inputs
{din2[1],din3,data_in,din2[0])
```

The following example shows an instance which you can drag-and-drop from the RTL view into the SCOPE user interface.

The following examples of Verilog source code are defined for this attribute.

```
module test (din1, din2, din3, data_in, dout, clk) /*synthesis
syn_black_box*/;

input [1:0] din1;
input [1:0] din2;
input [2:0] data_in;
input din3;
input clk;

output [1:0] dout;

endmodule


module test1 (din1, din2, din3, data_in, dout, clk);

input [1:0] din1;
input [1:0] din2;
input din3;
input [2:0] data_in;
input clk;

output [1:0] dout;
```


Copyright © 2013 Synopsys, Inc.

```
//Following wires are left floating
wire [1:0] d2_wire;
wire [2:0] data_wire;
wire d3_wire;

test UI (
      .din1(din1),
      .din2(d2_wire),
      .din3(d3_wire),
      .data_in(data_wire),
      .dout(dout),
      .clk(clk)
      );

endmodule

module top (din1, din2, din3, data_in, dout, clk);

input [1:0] din1;
input [1:0] din2;
input [2:0] data_in;
input din3;
input clk;

output [1:0] dout;

test1 U2 (
      .din1(din1),
      .din2(din2),
      .din3(din3),
      .data_in(data_in),
      .dout(dout),
      .clk(clk)
      );

endmodule
```

VHDL Syntax and Example

```
define_attribute {i:UI} syn_unconnected_inputs {din2, din3,
data_in}
```

The following examples of VHDL source code are defined for this attribute.

```
Library ieee;
use ieee.std_logic_1164.all;

entity top is
port (
      din1: in std_logic_vector(1 downto 0);
      din2: in std_logic_vector(1 downto 0);
      data_in: in std_logic_vector(2 downto 0);
      din3: in std_logic;
      clk: in std_logic;
      dout: out std_logic_vector(1 downto 0)

   );
end top;



architecture top_a of top is

component test is
port (
      din1: in std_logic_vector(1 downto 0);
      din2: in std_logic_vector(1 downto 0);
      data_in: in std_logic_vector(2 downto 0);
      din3: in std_logic;
      clk: in std_logic;
      dout: out std_logic_vector(1 downto 0)
   );
end component;
```


Copyright © 2013 Synopsys, Inc.

```
-------------------------
-- Wires left floating
-------------------------

signal d2_wire: std_logic_vector(1 downto 0);
signal d3_wire: std_logic;
signal data_wire: std_logic_vector(2 downto 0);

begin

UI: test port map (
            din1 => din1,
            din2 => d2_wire,
            data_in => data_wire,
            din3 => d3_wire,
            clk => clk,
            dout => dout);
end top_a;
```

syn_unique_inst_module

Directive; The syn_unique_inst_module directive, which is applied through a compiler directives file (`cdc`), allows an instance's module to be renamed while leaving the other instances of that module intact (pointing to the original module). This directive can be used in a hierarchical design flow where subprojects are created for an instance-based export. In this case, the instance must uniquely link to a particular module which is achieved by renaming the instance's module while leaving the other instances of the same module intact (pointing to the original module).

To illustrate this directive, consider module add with instances add_inst1 and add_inst2. The following directive creates new module add_new with instance add_inst1 referencing this module and instance add_inst2 remaining an instance of the original module (add).

```
define_directive {i:add_inst1} syn_unique_inst_module {add_new}
```

As a result of the renaming, instance add_inst1 becomes an instance of module add_new, but add_inst2 remains an instance of module add.

.cdc File Syntax

define_directive {*instanceName***} syn_unique_inst_module {***newModuleName***}**

The compiler directives file specifies directives to be added to the source code. During compilation, the tool passes all active `cdc` files to the compiler. The compiler references the object names in these files with the original RTL objects to assign the corresponding directive. For details about specifying directives with this file, see *Specifying Directives in a CDC File*, on page 248 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

syn_use_carry_chain (Lattice)

Attribute; Lattice ispXPLD. Used to turn on or off carry chain implementation for adders.

Constraint File Syntax and Example

The constraint file syntax for the attribute is:

syn_use_carry_chain {0|1}

For example:

```
syn_use_carry_chain=1
```

syn_use_carry_chain (Lattice)

Attribute; Use the syn_use_carry_chain attribute to turn on or off carry chain implementation for arithmetic and combinational operators depending on their input or output widths. You can apply this attribute globally or on a particular instance. Use any of the following operators: such as an adder, addmux, subtractor, accumulator, counter, or wide AND/OR gate and set the syn_use_carry_chain attribute value in the constraint file. This value controls whether or not carry chain is inferred.

Constraint File Syntax and Example

The constraint file syntax for the attribute is:

define_ attribute {*instanceName***} {syn_use_carry_chain} {***integerValue***}**

define_global_attribute {syn_use_carry_chain} {*integerValue***}**

Where:

- *instanceName* can be specified for an operator, such as an adder, addmux, subtractor, accumulator, counter, or wide AND/OR gate.
- *integerNalue* can be specified as follows:
  - To disable carry chain (SB_CARRY) usage, the value for this attribute should be greater than the input or output width of the logical or arithmetic operator.

  Also, for example:

  - For adders, carry chain inferencing occurs when the output width for the adder is less than or equal to the threshold value applied.
  - For comparators with only one output such as the > operator, the threshold value should be set according to input widths of the comparator.

Alternatively, you can set this attribute from the SCOPE Attribute drop-down menu.

Copyright © 2013 Synopsys, Inc.

Examples of syn_use_carry_chain Applied on an Instance

For details, see the following examples.

Example 1 (Applied on a Instance)

This code example contains two adders dout_reg1 and dout_reg. Also, the following attribute is set in the constraint file:

```
define_attribute {i:dout_reg1_1[16:0]} {syn_use_carry_chain} {18}
```

The syn_use_carry_chain attribute defines the carry chain inference limit for dout_reg1 to 18, which is more than its output width of 17. Therefore, this particular adder will not be mapped using SB_CARRY. To infer SB_CARRY, set the carry chain limit to 17 or less. However, the other adder instance will infer the standard carry chain.

```
module test (dout, din1, din2, din3, clk);

output [16:0] dout;
input [15:0] din1, din2, din3;
input clk;

reg [15:0] din1_reg, din2_reg;

reg [16:0] dout_reg, dout_reg1;
wire [16:0] data1;

assign data1 = dout_reg & din2_reg;
assign dout = dout_reg;

always @(posedge clk)
   begin
      din1_reg <= din1;
      din2_reg <= din2;
      dout_reg1 <= din1_reg + data1;
      dout_reg <= dout_reg1 + din3;
   end
endmodule
```

Example 2 (Applied on a Instance)

This code example contains two comparators; note that comparators must take into account their input width. Here the two comparators have an input width of 8; for which the ngr instance comparator is set to following constraint:

```
define_attribute {i:ngr} {syn_use_carry_chain} {100}
```

In this case, the syn_use_carry_chain value is greater than the input width of the comparator. Therefore, the software does not infer SB_CARRY for instance ngr. However, instance ngr2 infers the standard carry chain.

```
module test (a, b, c, d, gr, ngr, gr2,ngr2);

parameter size =8;
input[size-1:0]a,c,d;
input[size-1:0]b;
output gr, gr2;
output ngr, ngr2;

reg gr, gr2,ngr2;
reg ngr;

always@(a or b)
   begin
      if(a >= b)
         begin
            gr = 1'b1;
         ngr = 1'b0;
         end
      else
         begin
            gr = 1'b0;
            ngr = 1'b1;
         end
   end

always@(c or d)
   begin
      if(c >= d)
         begin
            gr2 = 1'b1;
            ngr2 = 1'b0;
         end
      else
         begin
```


Copyright © 2013 Synopsys, Inc.

```
            gr2 = 1'b0;
            ngr2 = 1'b1;
         end
   end

endmodule
```

Examples of syn_use_carry_chain Applied Globally

For details, see the following examples.

Example 1 (Applied Globally)

```
module adder_12 (
a_in,
b_in,
add_out,
c_out
);

input [11:0] a_in;
input [11:0] b_in;

output [11:0] add_out;
output c_out;

assign {c_out,add_out} = a_in + b_in;

endmodule
```

For this example, the output for the adder has 13 bits (this includes `c_out`). To infer the carry chain, set this attribute to the following:

```
define_global_attribute {syn_use_carry_chain} {13}
```

Therefore, any value that is less than or equal to 13 will infer carry chain. Carry chain will be disabled when you set the attribute value greater than 13; so you can set this value to 14 in this case.

Example 2 (Applied Globally)

```
module comp ( a, b, y );
input [7:0] a,b;
output y;

assign y = ( a > b )? 1'b1 : 1'b0;

endmodule
```

In this example, the output width is 1 bit. For comparators such as >, you must take into account the input width of 8. To infer carry chain, set this attribute to the following:

```
define_global_attribute {syn_use_carry_chain} {8}
```

Therefore, any value that is less than or equal to 8 will infer carry chain. Carry chain will be disabled when you set the attribute value greater than 8; so you can set this value to 9 in this case.

Copyright © 2013 Synopsys, Inc.

syn_use_xilinx_placement

Attribute; Use the syn_use_xilinx_placement attribute so that Xilinx global placement is used for placing the design in the Physical Plus flow. By default, all the physical synthesis flow uses Synplify global placement. For details about this flow, see *Xilinx Physical Plus*, on page 64. This attribute is available for Xilinx Virtex-5 and Virtex-6 devices with the Synplify Premier tools.

To enable this attribute, set syn_use_xilinx_placement to 1. The attribute is applied globally for the design. You can only specify this attribute in the constraint file.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_global_attribute syn_use_xilinx_placement {1}

For example:

```
define_global_attribute syn_use_xilinx_placement {1}
```

syn_useenables

Attribute

Controls the use of clock-enable registers within a design.

Vendor	Technology
Achronix	Speedster families
Altera	Stratix V, Cyclone V, Arria V, and older families
Lattice	EC, SC, XP, ORCA, isp families
Microsemi	30, 40, and 50 series, ACT, Axcelerator, ex, and G5 families
Xilinx	Virtex-7 and older families Spartan-3 and Spartan-6 families

syn_useenables Values

Default	Global	Object Type
1/true	No	Register

Value	Description
1/true	Infers registers with clock-enable pins
0/false	Uses external logic to generate the clock-enable function for the register

Description

By default, the synthesis tool uses registers with clock enable pins where applicable. Setting the syn_useenables attribute to 0 on a register creates external clock-enable logic to allow the tool to infer a register that does not require a clock-enable.

By eliminating the need for a clock-enable, designs can be mapped into less complex registers that can be more easily packed into RAMs or DSPs. The trade-off is that while conserving complex registers, the additional external clock-enable logic can increase the overall logic-unit count.

Copyright © 2013 Synopsys, Inc.

Syntax Specification

FDC	**define attribute {***register*|*signal***} syn_useenables {0|1}**	SCOPE Example
Verilog	*object* **/* synthesis syn_useenables = "0|1" */;**	Verilog Example
VHDL	**attribute syn_useenables of** *object* **:** *objectType* **is "true|false";**	VHDL Example

SCOPE Example

Enable	Object Type	Object	Attribute	**Value**	Value Type	Description
☑	register	i:q[1:0]	syn_useenables	0	boolean	Generate with clock enable pin

Verilog Example

```
module useenables(d,clk,q,en);
input [1:0] d;
input en,clk;
output [1:0] q;
reg [1:0] q /* synthesis syn_useenables = 0 */;

always @(posedge clk)
   if (en)
      q<=d;
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;

entity syn_useenables is
   port (d : in std_logic_vector(1 downto 0);
         en,clk : in std_logic;
         q : out std_logic_vector(1 downto 0) );
attribute syn_useenables: boolean;
attribute syn_useenables of q: signal is false;
end;

architecture syn_ue of syn_useenables is
begin
   process (clk) begin
      if (clk = '1' and clk'event) then
         if (en='1') then
            q <= d;
         end if;
      end if;
   end process;
end architecture;
```


Copyright © 2013 Synopsys, Inc.

Effect of Using syn_useenables

Without applying the attribute (default is to use registers with clock-enable pins) or setting the attribute to 1/true uses registers with clock-enable pins (FDEs in the below schematic).

Applying the attribute with a value of 0/false uses registers without clock-enable pins (FDs in the below schematic) and creates external clock-enable logic.

Copyright © 2013 Synopsys, Inc.

syn_useioff (Altera)

Attribute; Altera (Arria families, Stratix families, Cyclone families). Overrides the default behavior to pack registers into I/O pad cells based on timing requirements for the target Altera families. When syn_useioff is set to 1 (true), the software tries to pack the registers into the I/O pad cells; when set to 0, the software does not pack the registers.

With Altera Arria and Stratix families, when this attribute is enabled, registers are not packed into Multiply/Accumulate (MAC) blocks.

You can place this attribute on an individual register or port, or apply it globally. When applied globally, the synthesis tool packs all input, output, and I/O registers into I/O pad cells. When applied to a register, the synthesis tool packs the register into the pad cell, and when applied to a port, packs all registers attached to the port into the pad cell.

NOTE: The syn_useioff attribute can be set on a:

- top-level port (see Example 1)
- register driving the top-level port (see Example 2)
- lower-level port, only if the register is specified as part of the port declaration (see Example 3)

Example 1

This attribute is applied on a top-level port.

```
module good2_top (
            input clk,
            input d,
            output q/* synthesis syn_useioff=1 */
            );
   good_2 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule

   module good_2 (
            input clk,
            input d,
```

```
            output q
            );

   reg temp;
   reg qreg;

   assign q = qreg;
   always@(posedge clk)begin
      temp <= d;
      qreg <= temp;
   end

   endmodule
```

NOTE: The following condition applies if syn_useioff is applied on a top-level port or on registers which drive top-level ports. The registers can be included in a lower-level module.

Example 2

This attribute is applied on a register driving the top-level port.

```
module good1_top (
            input clk,
            input d,
            output q
            );
   good_1 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule

   module good_1 (
            input clk,
            input d,
            output q
            );

   reg temp;
   reg qreg/* synthesis syn_useioff=1 */;

   assign q = qreg;
   always@(posedge clk)begin
```


Copyright © 2013 Synopsys, Inc.

```
   temp <= d;
   qreg <= temp;
end

endmodule
```

NOTE: The following condition applies if syn_useioff is applied on a top-level port or on registers which drive top-level ports. The registers can be included in a lower-level module.

Example 3

This attribute is applied on a lower-level port, for which the register is defined as part of the port declaration. However, the attribute is not applied if the register driving the port is declared independently.

```
module good_top (
         input clk,
         input d,
         output q
         );
good U (
         .clk(clk),
         .d(d),
         .q(q)
         );

endmodule

module good (
         input clk,
         input d,
         output reg q/* synthesis syn_useioff=1 */
         );

reg temp;
//reg qreg;

//assign q = qreg;
always@(posedge clk)begin
   temp <= d;
   q    <= temp;
end

endmodule
```

NOTE: The following conditions apply:

- If syn_useioff is applied on a lower-level port, for which a register is defined as part of the port definition, the port should be driven within a clocked block.
- If syn_useioff is applied on a lower-level port, for which a register is defined independently and is not driven within a clocked block, then this attribute will not work.

When using the syn_useioff attribute to pack registers into I/O pad cells, the order of precedence is registers, followed by ports, and then global as outlined below:

1. Registers (highest priority)

syn_useioff = 1	Pack register into the I/O pad cell regardless of the port or global specification
syn_useioff = 0	Do not pack register into I/O pad cell regardless of the port or global specification

2. Ports

syn_useioff = 1	Pack registers into the I/O pad cell regardless of global specification
syn_useioff = 0	Do not pack registers into I/O pad cell regardless of global specification

3. Global (lowest priority)

syn_useioff = 1	Pack registers into the I/O pad cells
syn_useioff = 0	Do not pack registers into I/O pad cells

Constraint File Syntax and Examples

define_attribute {*portName*} syn_useioff {1 | 0}

define_global_attribute syn_useioff {1 | 0}

For example:

```
define_attribute {z[3:0]} syn_useioff {1}
```


Copyright © 2013 Synopsys, Inc.

```
define_global_attribute syn_useioff {1}
```

Verilog Syntax and Examples

object **/* synthesis syn_useioff = 1 | 0 */ ;**

To use this attribute globally, apply it to the module.

```
module test (a, b, clk, rst, d) /* synthesis syn_useioff = 1 */;
```

To use this attribute on individual ports, apply it to individual port declarations.

```
module test (a, b, clk, rst, d);
input a;
input b /* synthesis syn_useioff = 1 */;
```

VHDL Syntax and Examples

attribute syn_useioff of *object* **:** *objectType* **is true | false ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

To use syn_useioff globally, apply it to the architecture.

```
Entity test is port (
   .
   .
   .
   );
end test;

architecture archtest of test is
signal temp : std_logic;
signal temp1 : std_logic;
signal temp2 : std_logic;
signal temp3 : std_logic;

attribute syn_useioff : boolean;
attribute syn_useioff of archtest : architecture is true;
```

To use syn_useioff on individual ports, apply it to the ports in the entity section.

```
entity test is port (
   a : in std_logic;
      .
      .
      .);

attribute syn_useioff : boolean;
attribute syn_useioff of a : signal is true;
end test,
```


Copyright © 2013 Synopsys, Inc.

syn_useioff (Lattice)

Attribute; Lattice (LatticeECP3/ECP2S/ECP2M/ECP2, LatticeECP/EC, LatticeXP2/XP, LatticeSC/SCM, MachXO, and ORCA). Overrides the default behavior to pack registers into I/O pad cells based on timing requirements for the target Lattice families above. When syn_useioff is set to 1 (true), the software tries to pack the registers into the I/O pad cells; when set to 0, the software does not pack the registers.

You can place this attribute on an individual register or port, or apply it globally. When applied globally, the synthesis tool packs all input, output, and I/O registers into I/O pad cells. When applied to a register, the synthesis tool packs the register into the pad cell, and when applied to a port, packs all registers attached to the port into the pad cell.

NOTE: The syn_useioff attribute can be set on a:

- top-level port (see Example 1)
- register driving the top-level port (see Example 2)
- lower-level port, only if the register is specified as part of the port declaration (see Example 3)

Example 1

This attribute is applied on a top-level port.

```
module good2_top (
            input clk,
            input d,
            output q/* synthesis syn_useioff=1 */
            );
   good_2 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule

   module good_2 (
            input clk,
            input d,
            output q
            );
```

```
reg temp;
reg qreg;

assign q = qreg;
always@(posedge clk)begin
   temp <= d;
   qreg <= temp;
end

endmodule
```

NOTE: The following condition applies if syn_useioff is applied on a top-level port or on registers which drive top-level ports. The registers can be included in a lower-level module.

Example 2

This attribute is applied on a register driving the top-level port.

```
module good1_top (
            input clk,
            input d,
            output q
            );
   good_1 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule

   module good_1 (
            input clk,
            input d,
            output q
            );

   reg temp;
   reg qreg/* synthesis syn_useioff=1 */;

   assign q = qreg;
   always@(posedge clk)begin
```


Copyright © 2013 Synopsys, Inc.

```
   temp <= d;
   qreg <= temp;
end

endmodule
```

NOTE: The following condition applies if syn_useioff is applied on a top-level port or on registers which drive top-level ports. The registers can be included in a lower-level module.

Example 3

This attribute is applied on a lower-level port, for which the register is defined as part of the port declaration. However, the attribute is not applied if the register driving the port is declared independently.

```
module good_top (
         input clk,
         input d,
         output q
         );
good U (
         .clk(clk),
         .d(d),
         .q(q)
         );

endmodule

module good (
         input clk,
         input d,
         output reg q/* synthesis syn_useioff=1 */
         );

reg temp;
//reg qreg;

//assign q = qreg;
always@(posedge clk)begin
   temp <= d;
   q    <= temp;
end

endmodule
```

NOTE: The following conditions apply:

- If syn_useioff is applied on a lower-level port, for which a register is defined as part of the port definition, the port should be driven within a clocked block.
- If syn_useioff is applied on a lower-level port, for which a register is defined independently and is not driven within a clocked block, then this attribute will not work.

When using the syn_useioff attribute to pack registers into I/O pad cells, the order of precedence is registers, followed by ports, and then global as outlined below:

1. Registers (highest priority)

syn_useioff = 1	Pack register into the I/O pad cell regardless of the port or global specifications
syn_useioff = 0	Do not pack register into I/O pad cell regardless of the port or the global specification

2. Ports

syn_useioff = 1	Pack registers into the I/O pad cell regardless of global specifications
syn_useioff = 0	Do not pack registers into I/O pad cell regardless of global specification

3. Global (lowest priority)

syn_useioff = 1	Pack registers into the I/O pad cells
syn_useioff = 0	Do not pack registers into I/O pad cells

Constraint File Syntax and Example

define_attribute {*portName*} syn_useioff {1 | 0}

define_global_attribute syn_useioff {1 | 0}

For example:

```
define_attribute {z[3:0]} syn_useioff {1}
```


Copyright © 2013 Synopsys, Inc.

```
define_global_attribute syn_useioff {1}
```

Verilog Syntax and Examples

object **synthesis syn_useioff = {1 | 0} */ ;**

To use this attribute globally, apply it to the module.

```
module test (a, b, clk, rst, d) /* synthesis syn_useioff = 1 */;
```

To use this attribute on individual ports, apply it to individual port declarations.

```
module test (a, b, clk, rst, d);
input a;
input b /* synthesis syn_useioff = 1 */;
```

VHDL Syntax and Examples

attribute syn_useioff of *object* **:** *objectType* **is true | false ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

To use this attribute globally, apply it to the architecture:

```
architecture archtest of test is
signal temp : std_logic;
signal temp1 : std_logic;
signal temp2 : std_logic;
signal temp3 : std_logic;

attribute syn_useioff : boolean;
attribute syn_useioff of archtest : architecture is true;
```

To use this attribute on individual ports, apply it to the ports in the entity section.

```
entity test is
   port (a : in std_logic;
         .
         .
         .
         );

attribute syn_useioff : boolean;
attribute syn_useioff of a : signal is true;
end test;
```


Copyright © 2013 Synopsys, Inc.

syn_useioff (Lattice iCE65/iCE40)

Attribute; This attribute specifies whether or not to pack registers with the I/Os. When syn_useioff is set to 1, the software tries to pack the registers with the I/Os; when set to 0, the software does not pack the registers. The output netlist is not modified when registers are packed, however, this information is forward-annotated to the `edif` file.

The syn_useioff attribute can be set on a port, bit port, or the top-level view.

If you do not specify the syn_useioff attribute on the port or from the top-level view, the synthesis software will perform timing-driven I/O register packing on the port. The output netlist is not modified when registers are packed, however, this information is also forward-annotated to the `edif` file.

Note: The Merge Registers into I/O option is no longer supported. You can globally specify syn_useioff on the top-level view instead.

Constraint File Syntax and Examples

define_attribute {*port*} syn_useioff {1|0}

define_global_attribute syn_useioff {1|0}

Verilog Syntax and Examples

object **/* synthesis syn_useioff = 1 | 0 */ ;**

This syn_useioff attribute is applied on a port.

```
module good2_top (
            input clk,
            input d,
            output q/* synthesis syn_useioff=1 */
            );
   good_2 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule
```

```
module good_2 (
         input clk,
         input d,
         output q
         );

reg temp;
reg qreg;

assign q = qreg;
always@(posedge clk)begin
   temp <= d;
   qreg <= temp;
end

endmodule
```

This example sets the attribute globally, from the top-level module:

```
module test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

VHDL Syntax and Examples

attribute syn_useioff of *object* **:** *objectType* **is true | false ;**

To use this attribute globally, apply it to the architecture of the top-level module:

```
architecture rtl of top is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```

To use this attribute on individual ports, apply it to the ports in the entity section.

```
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;
```


Copyright © 2013 Synopsys, Inc.

Forward-annotating I/O Register Packing

When registers are packed into the I/O pad cells, the attribute is specified in the `edif` file with the following properties: Input FF, Output FF, and Output Enable FF. Therefore, if logic FF is packed into the input FF of an I/O pad, then the I/O pad instance is specified as IO_FF_IN. Similarly, for output and output enable registers, the I/O pad instance is specified as IO_FF_OUT and IO_FF_OE respectively.

When using the syn_useioff attribute to pack registers into I/O pad cells, the order of precedence is ports and then global as outlined below:

1. Ports

syn_useioff = 1	Pack registers into the I/O pad cell regardless of global specification
syn_useioff = 0	Do not pack registers into I/O pad cell regardless of global specification

2. Global (lowest priority)

syn_useioff = 1	Pack registers into the I/O pad cells
syn_useioff = 0	Do not pack registers into I/O pad cells

Copyright © 2013 Synopsys, Inc.

syn_useioff (Xilinx)

Attribute; Xilinx (Virtex and Spartan-II families). Overrides the default behavior to pack registers into I/O pad cells based on timing requirements. When syn_useioff is set to 1 (true), the software tries to pack the registers into the I/O pad cells; when set to 0, the software does not pack the registers.

The synthesis software writes out the syn_useioff attribute to the `edf` file as IOB=FALSE/TRUE statements when you set the attribute on a top-level register or port.

NOTE: The syn_useioff attribute can be set on a:

- top-level port (see Example 1)
- register driving the top-level port (see Example 2)
- lower-level port, only if the register is specified as part of the port declaration (see Example 3)

Example 1

This attribute is applied on a top-level port.

```
module good2_top (
            input clk,
            input d,
            output q/* synthesis syn_useioff=1 */
            );
   good_2 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule

   module good_2 (
            input clk,
            input d,
            output q
            );

   reg temp;
   reg qreg;
```


Copyright © 2013 Synopsys, Inc.

```
assign q = qreg;
always@(posedge clk)begin
   temp <= d;
   qreg <= temp;
end

endmodule
```

NOTE: The following condition applies if syn_useioff is applied on a top-level port or on registers which drive top-level ports. The registers can be included in a lower-level module.

Example 2

This attribute is applied on a register driving the top-level port.

```
module good1_top (
            input clk,
            input d,
            output q
            );
   good_1 U (
            .clk(clk),
            .d(d),
            .q(q)
            );

   endmodule

   module good_1 (
            input clk,
            input d,
            output q
            );

   reg temp;
   reg qreg/* synthesis syn_useioff=1 */;

   assign q = qreg;
   always@(posedge clk)begin
      temp <= d;
      qreg <= temp;
   end

   endmodule
```

NOTE: The following condition applies if syn_useioff is applied on a top-level port or on registers which drive top-level ports. The registers can be included in a lower-level module.

Example 3

This attribute is applied on a lower-level port, for which the register is defined as part of the port declaration. However, the attribute is not applied if the register driving the port is declared independently.

```
module good_top (
         input clk,
         input d,
         output q
         );
good U (
         .clk(clk),
         .d(d),
         .q(q)
         );

endmodule

module good (
         input clk,
         input d,
         output reg q/* synthesis syn_useioff=1 */
         );

reg temp;
//reg qreg;

//assign q = qreg;
always@(posedge clk)begin
   temp <= d;
   q    <= temp;
end

endmodule
```

NOTE: The following conditions apply:

- If syn_useioff is applied on a lower-level port, for which a register is defined as part of the port definition, the port should be driven within a clocked block.

Copyright © 2013 Synopsys, Inc.

- If syn_useioff is applied on a lower-level port, for which a register is defined independently and is not driven within a clocked block, then this attribute will not work.

When you set the syn_useioff attribute to true globally, all I/O registers have IOB=TRUE statements unless overridden by a syn_useioff on an individual register or port to prevent packing. If you globally set the attribute to false, the software does not write out any constraints to the `ncf` file. Any global settings are overridden by attributes on individual registers or ports.

If the IOB=FALSE/TRUE is set by the synthesis tool because of timing requirements, the attribute is written out into the `.ncf` file.

The following table summarizes the behavior:

To...	**Set** syn_useioff **to...**
Pack registers	1, globally (all I/O registers are packed) or on individual registers
Explicitly prevent register packing on individual registers	0, on individual registers
Manually set register packing in the P&R tool	0, globally (no IOB statements are written). Setting 1 on individual registers passes IOB statement to P&R tool.
Leave register packing to the synthesis tool	No global syn_useioff

When using the syn_useioff attribute to pack registers into I/O pad cells, the order of precedence is registers, followed by ports, and then global as outlined below:

1. Registers (highest priority)

syn_useioff = 1	Pack register into the I/O pad cell regardless of the port or global specification
syn_useioff = 0	Do not pack register into I/O pad cell regardless of the port or the global specification

2. Ports

syn_useioff = 1	Pack registers into the I/O pad cell regardless of global specification
syn_useioff = 0	Do not pack registers into I/O pad cell regardless of global specification

3. Global (lowest priority)

syn_useioff = 1	Pack registers into the I/O pad cells
syn_useioff = 0	Do not pack registers into I/O pad cells

Constraint File Syntax and Examples

define_attribute {*port*} syn_useioff {1 | 0}

define_global_attribute syn_useioff {1 | 0}

For example, to embed flip-flops locally in IOBs:

```
define_attribute {p:q[3:0]} syn_useioff {1}
```

To embed flip-flops globally:

```
define_global_attribute syn_useioff {1}
```


Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Examples

object **/* synthesis syn_useioff = 1 | 0 */ ;**

This example sets the attribute on a port:

```
module test(d, clk, q);
input [3:0] d;
input clk;
output [3:0] q /* synthesis syn_useioff = 1 */;
reg q;
endmodule
```

This example sets the attribute globally, from the top-level module:

```
module test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

VHDL Syntax and Examples

attribute syn_useioff of *object* **:** *objectType* **is true | false ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

To use this attribute globally, apply it to the architecture of the top-level module:

```
architecture rtl of top is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```

To use this attribute on individual ports, apply it to the ports in the entity section.

```
entity test is
   port (d : in std_logic_vector(3 downto 0);
         clk : in std_logic;
         q : out std_logic_vector(3 downto 0) );
attribute syn_useioff : boolean;
attribute syn_useioff of q : signal is true;
end test;
```

syn_user_instance

Attribute; Xilinx. Prevents optimizations on user-instantiated instances. This attribute can be set globally, or on a view/module, or instance. The syn_user_instance attribute is only supported in the Synplify Premier tool.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute {*object*} syn_user_instance {1|0}

define_global_attribute syn_user_instance {1|0}

Where:

- *object* can be a view/module or a user instantiated primitive.
- *value* can be the following:
  - 1—Do not optimize the specified instance, but write this instance to the output netlist. If you specify this attribute globally, any user-instantiated instance of the design is not touched.
  - 0—Freely optimize the view/module, instance, or globally on the design, and write out its content to the output netlist. This is the default.

    Note that the view/module or instance is not optimized for basic logic synthesis, but the software does write out its content to the output netlist.

For example:

```
define_global_attribute syn_user_instance {1}
define_attribute {v:work.prep2_2} syn_user_instance {0}
define_attribute {i:LUTinst0} syn_user_instance {1}
```

You can use the syn_user_instance attribute to selectively disable optimizations for instantiated macros/modules when the global Enhanced Optimization switch is enabled.

For more information about the syn_macro attribute and how it differs from syn_user_instance, see *syn_macro Versus syn_user_instance*, on page 1393.

Copyright © 2013 Synopsys, Inc.

Verilog Syntax and Example

For Verilog designs, you can set the syn_user_instance attribute on a view or an instance. Use this attribute on instantiated primitives such as LUTs, FFs, or BRAMs.

The following Verilog example sets syn_user_instance on a view.

```
module LutRestructure(
   input din,
   input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
   input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
   output out );

LutRestructure_mod inst0(
   .din(din),.in_lut0(in_lut0),.in_lut1(in_lut1),
      .in_lut2(in_lut2),.in_lut3(in_lut3),.in_lut4(in_lut4),
      .in_lut5(in_lut5),.in2_lut0(in2_lut0),.in2_lut1(in2_lut1),
      .in2_lut2(in2_lut2),.in2_ut3(in2_lut3),.out(out) );
endmodule

module LutRestructure_mod(
   input din,
   input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
   input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
   output out ) /*synthesis syn_user_instance = 1*/;
   wire out_l;

LUT6  LUTinst0(
   .O(out_l),.I5(in_lut5),.I2(in_lut4),.I0(in_lut3),
      .I4(in_lut2),.I3(in_lut1),.I1(in_lut0) );
defparam LUTinst0.INIT = 64'h8000000000000000;

LUT5  LUTinst1(
   .O(out),.I2(in2_lut0),.I0(din),.I4(in2_lut1),
      .I1(in2_lut2),.I3(out_l) );
defparam LUTinst1.INIT = 32'h80000000;
endmodule
```

The following Verilog example sets syn_user_instance on an instance.

```
module LutRestructure(
   input din,
   input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
   input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
   output out );
```

```
LutRestructure_mod inst0(
   .din(din),.in_lut0(in_lut0),.in_lut1(in_lut1),
      .in_lut2(in_lut2),.in_lut3(in_lut3),.in_lut4(in_lut4),
      .in_lut5(in_lut5),.in2_lut0(in2_lut0),.in2_lut1(in2_lut1),
      .in2_lut2(in2_lut2),.in2_lut3(in2_lut3),.out(out) );
endmodule

module LutRestructure_mod(
   input din,
   input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
   input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
   output out );
wire out_l;

LUT6 LUTinst0(
   .O(out_l),.I5(in_lut5),.I2(in_lut4),.I0(in_lut3),.I4(in_lut2),
      .I3(in_lut1),.I1(in_lut0) )
      /*synthesis syn_user_instance = 1*/;
defparam LUTinst0.INIT = 64'h8000000000000000;

LUT5 LUTinst1(
   .O(out),.I2(in2_lut0),.I0(din),.I4(in2_lut1),.I1(in2_lut2),
      .I3(out_l) ) /*synthesis syn_user_instance = 1*/;
defparam LUTinst1.INIT = 32'h80000000;
endmodule
```

VHDL Syntax and Example

For VHDL designs, you can set the syn_user_instance attribute on a view or an instance. Use this attribute on instantiated primitives such as LUTs, FFs, or BRAMs.

The following VHDL example sets syn_user_instance on a view.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity LutRestructure is
   port (out_c : out std_logic;
         in2_lut1_c : in std_logic;
         in2_lut0_c : in std_logic;
         in2_lut2_c : in std_logic;
```


Copyright © 2013 Synopsys, Inc.

```
            din_c : in std_logic;
            in_lut5_c : in std_logic;
            in_lut2_c : in std_logic;
            in_lut1_c : in std_logic;
            in_lut4_c : in std_logic;
            in_lut0_c : in std_logic;
            in_lut3_c : in std_logic );
end LutRestructure;

architecture beh of LutRestructure is
signal OUT_L : std_logic ;
signal GND : std_logic ;
signal VCC : std_logic ;
attribute syn_user_instance : boolean;
attribute syn_user_instance of beh : architecture is true;

begin
LUTINST0: LUT6
generic map (INIT => X"8000000000000000")
port map (
   I0 => in_lut3_c,
   I1 => in_lut0_c,
   I2 => in_lut4_c,
   I3 => in_lut1_c,
   I4 => in_lut2_c,
   I5 => in_lut5_c,
   O => OUT_L );
LUTINST1: LUT5
generic map (INIT => X"80000000")
port map (
   I0 => din_c,
   I1 => in2_lut2_c,
   I2 => in2_lut0_c,
   I3 => OUT_L,
   I4 => in2_lut1_c,
   O => out_c );
GND <= '0';
VCC <= '1';
end beh;
```

The following VHDL example sets syn_user_instance on an instance.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity LutRestructure is
   port (out_c : out std_logic;
         in2_lut1_c : in std_logic;
         in2_lut0_c : in std_logic;
         in2_lut2_c : in std_logic;
         din_c : in std_logic;
         in_lut5_c : in std_logic;
         in_lut2_c : in std_logic;
         in_lut1_c : in std_logic;
         in_lut4_c : in std_logic;
         in_lut0_c : in std_logic;
         in_lut3_c : in std_logic);
end LutRestructure;

architecture beh of LutRestructure is
signal OUT_L : std_logic ;
signal GND : std_logic ;
signal VCC : std_logic ;
attribute syn_user_instance : boolean;
attribute syn_user_instance of LUTINST0 : label is true;
attribute syn_user_instance of LUTINST1 : label is true;

begin
LUTINST0: LUT6
generic map (INIT => X"8000000000000000")
port map (
   I0 => in_lut3_c,
   I1 => in_lut0_c,
   I2 => in_lut4_c,
   I3 => in_lut1_c,
   I4 => in_lut2_c,
   I5 => in_lut5_c,
   O => OUT_L );
LUTINST1: LUT5
generic map (INIT => X"80000000")
port map (
   I0 => din_c,
   I1 => in2_lut2_c,
   I2 => in2_lut0_c,
```


Copyright © 2013 Synopsys, Inc.

```
   I3 => OUT_L,
   I4 => in2_lut1_c,
   O => out_c );
GND <= '0';
VCC <= '1';
end beh;
```

syn_vote_loops

Attribute; Altera and Xilinx. Use this attribute when a loop exists in the module for which distributed TMR is applied. By default, to restore the state after effects of an SEU fault, the loop in each of the three instances must insert a voter. If you have used a mitigation mechanism to avoid accumulation of faults in a sequential feedback path and do not want the overhead of voter logic, you can turn off adding voters for your design by setting the syn_vote_loops attribute to false. Doing so can provide the following results:

- Reduces the area for the design because less voter logic is inserted.
- Improves timing or quality of results (QoR) for the design. The logic in the critical path is reduced through the synchronous voters.
- Does not change the interface for the TMR module.

This attribute can only be applied on a view. Do not use this attribute on any of it sub-hierarchies or an instance. If this attribute is set to false, the message "Found sequential loop" will not appear in the log file reducing its size and runtime because messages are not included and is less than when voter logic is inserted for sequential feedback paths. You must use this attribute with the syn_radhardlevel attribute. For more information, see *syn_radhardlevel*, on page 1462.

Constraint File Syntax and Example

The constraint file syntax for the attribute is

define_attribute {*object*} syn_vote_loops {true|false}

where *object* is a view. For example:

```
define_attribute {v:work.M1} syn_vote_loops {false}
```

The default is true.

Verilog Syntax and Example

object **/* synthesis syn_vote_loops = "true | false"*/ ;**

where *object* is a view or module. For example:

```
module ram (addra,addrb,clka,din1a,acc_out,rst,we,re)
   /* synthesis syn_vote_loops = "true" */ ;
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute syn_vote_loops:string;
attribute syn_vote_loops of RTL: architecture is "true | false" ;

Apply syn_vote_loops to the architecture. For example:

```
attribute syn_vote_loops : string;
attribute syn_vote_loops of RTL : architecture is "false";
```

translate_off/translate_on

Directive. Allows you to synthesize designs originally written for use with other synthesis tools without needing to modify source code. All source code that is between these two directives is ignored during synthesis.

Another use of these directives is to prevent the synthesis of stimulus source code that only has meaning for logic simulation. You can use translate_off/translate_on to skip over simulation-specific lines of code that are not synthesizable.

When you use translate_off in a module, synthesis of all source code that follows is halted until translate_on is encountered. Every translate_off must have a corresponding translate_on. These directives cannot be nested, therefore, the translate_off directive can only be followed by a translate_on directive.

Note: See also, *pragma translate_off/pragma translate_on*, on page 1214. These directives are implemented the same in the source code.

Verilog Syntax and Example

The Verilog syntax for these directives is as follows:

/* synthesis translate_off */

/* synthesis translate_on */

For example:

```
module test(input a, b, output c);

//synthesis translate_off
assign c=a&b

//synthesis translate_on
assign c=a|b;
endmodule
```

For SystemVerilog designs, you can alternatively use the synthesis_off/synthesis_on directives. The directives function the same as the translate_off/translate_on directives to ignore all source code contained between the two directives during synthesis.

Copyright © 2013 Synopsys, Inc.

For Verilog designs, you can use the synthesis macro with the Verilog ‘ifdef directive instead of the translate on/off directives. See *synthesis Macro*, on page 753 for information.

VHDL Syntax and Example

For VHDL designs, you can alternatively use the synthesis_off/synthesis_on directives. Select Project->Implementation Options->VHDL and enable the Synthesis On/Off Implemented as Translate On/Off option. This directs the compiler to treat the synthesis_off/on directives like translate_off/on and ignore any code between these directives.

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

The following is the VHDL syntax for translate-off/translate_on:

synthesis translate_off

synthesis translate_on

For example:

```
architecture behave of ram4 is
begin

-- synthesis translate_off
stimulus: process (clk, a, b)

-- Source code you DO NOT want synthesized

end process;
-- synthesis translate_on

-- Other source code you WANT synthesized
```

xc_area_group

Attribute; Xilinx (Virtex and Spartan families). Available in the Synplify Pro product only. The Synplify Premier tool *does not use* this attribute; it is used by other Synopsys FPGA products to assign a compile point to a Xilinx area group, specifying its location and area. Physical regions you define with the Design Planner option in the Synplify Premier tool are automatically assigned to corresponding Xilinx area groups. During fitting, the Xilinx place-and-route tool places all of the compile-point logic within the assigned area group.

You apply this attribute to a compile point (see *Compile Point Basics*, on page 1088). The attribute assigns the compile point to a Xilinx area group, specifying its location and area. During fitting, the Xilinx place-and-route tool places all of the compile point logic within the assigned area group. You must not use this attribute on a module that has been instantiated more than once.

Note: Do *not* use this attribute with the Synplify Premier tool. Physical assignments can only be made in the Design Planner view; not by assigning attributes. The Synplify Premier tool *ignores* applications of this attribute in constraint files: it does not forward-annotate them to the Xilinx place-and-route tool.

The attribute value specifies the location and area of the area group, by locating its top left and bottom right corners.

For information on compile points and the compile-point synthesis flow, see *Synthesizing Compile Points*, on page 1109.

Constraint File Syntax and Examples

define_attribute {v:*module*|*architectureName***}**
xc_area_group {*topleft***:***bottomright***}**

where *module* or *architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a *compile point*, and *topleft* and *bottomright* are the coordinates of the top left and bottom right corners of the area group, respectively.

Copyright © 2013 Synopsys, Inc.

Each of *topleft* and *bottomright* has the following syntax, where *N1* and *N2* are integers representing the X and Y coordinates of the corner, respectively:

- For Spartan-II, Spartan-IIE, Spartan-3/3E, Spartan-6: **CLB_R** *N1* **C** *N2*
- For Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, Virtex-6: **SLICE_X** *N1* **Y** *N2*

The following example assigns the compile point uc_alu.U1 to the area group whose top-left and bottom-right CLBs have coordinates (10, 29) and (16, 35), respectively.

```
define_attribute {uc_alu.U1} xc_area_group {CLB_R10C29:CLB_R16C35}
```

Note: You apply xc_area_group only in a constraint file; you cannot use this attribute in source code.

xc_clockbuftype

Attribute; Xilinx (Virtex families). Uses the Clock Delay Locked Loop primitive (CLKDLL) for a clock port. This is inferred as a buffer called BUFGDLL, which includes the CLKDLL primitive.

Constraint File Syntax and Example

define_attribute {*port*} xc_clockbuftype {BUFGDLL}

For example:

```
define_attribute {clk} xc_clockbuftype {BUFGDLL}
```

Verilog Syntax and Example

object **/* synthesis xc_clockbuftype = "BUFGDLL" */ ;**

For example:

```
module test_clkbuftype(d, clk, rst, q);
input [3:0] d;
input clk /* synthesis xc_clockbuftype = "BUFGDLL" */;
input rst;
output [3:0] q;

// Other code
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute xc_clockbuftype of *object* **:** *objectType* **is "BUFGDLL" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

For example:

```
library ieee, synplify;
use ieee.std_logic_1164.all;

entity test_clkbuftype is
   port (d : in std_logic_vector(3 downto 0);
         clk, rst : in std_logic;
         q : out std_logic_vector(3 downto 0) );
attribute xc_clockbuftype : string;
attribute xc_clockbuftype of clk : signal is "BUFGDLL";
end test_clkbuftype;
```

EDIF Output Example

Applying this attribute causes the resultant EDIF output file to appear similar to the following:

```
(instance clk_ibuf (viewRef PRIM (cellRef BUFGDLL
   (libraryRef VIRTEX)))
```

xc_fast

Attribute.
Synplify Pro and Synplify Premier.

Speeds up transition time of the output driver.

Vendor	Technology
Xilinx	All

xc_fast Values

Value	Description
0 | false (Default)	Keeps the transition time slow.
1 | true	Speeds up the transition time.

Description

Use this attribute to make the transition time of the output driver fast. The default transition time is slow (see *xc_slow*, on page 1624).

The synthesis tool provides attributes that are passed into the XNF or EDIF netlist for Xilinx placement and routing. These attributes affect the input setup times and output transition times for your I/Os. The transition time of the output driver can be programmed as fast or slow for Xilinx devices. Use the xc_fast attribute to decrease the transition time for the output driver.The xc_fast attribute gets passed to the Xilinx I/O Block Parameter as FAST in the XNF or EDIF netlist.

xc_fast Syntax

Global Support	Object
No	Output port

Copyright © 2013 Synopsys, Inc.

This is how you specify the attribute in different files:

FDC	define_attribute {*object*} xc_fast {*value*}	Constraint Examples
Verilog	*Object* /* synthesis xc_fast = "*value*" */;	Verilog Example
VHDL	attribute xc_fast of *object*: objectType is "*value*";	VHDL Example

Constraint Examples

define_attribute {DATA0[5]} xc_fast {1}

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	✔		p:DATA0[7:0]	xc_fast	1		

Verilog Example

```
module counter (CLK, RST, DATA0);
input CLK, RST;
output [7:0] DATA0 /* synthesis xc_fast = 1 */;
reg [7:0] DATA0 ;

always @(posedge CLK or posedge RST)
begin
   if (RST)
      DATA0 = 0;
   else
      DATA0 = DATA0 + 1;
end
endmodule
```

VHDL Example

```
entity count is
port(clk : in bit;
      rst : in bit;
   data0 : out std_logic_vector(7 downto 0)
      );
attribute xc_fast : Boolean;
attribute xc_fast of data0 : signal is true;
end count;

architecture behave of count is
   signal data0_i: std_logic_vector(7 downto 0);
begin

   counter: process (rst,clk)
   begin
      if  rst = '1' then
         data0_i <= "00000000";
      elsif clk = '1' and clk'event then
            data0_i  <= data0_i + "00000001";
            end if;
      end process counter;

    data0 <= data0_i;
end behave;
```

Effect of Using xc_fast

The following table shows a design without the xc_fast attribute.

Verilog	`output [7:0] DATA0 /* synthesis xc_fast = 0 */;`
VHDL	`attribute xc_fast of data0 : signal is false;`

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell counter (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT))
           (port CLK (direction INPUT)
 )
           (port RST (direction INPUT))
         )
```


Copyright © 2013 Synopsys, Inc.

The following table shows a design with the xc_fast attribute.

Verilog	output [7:0] DATA0 `/* synthesis xc_fast = 0 */;`
VHDL	`attribute xc_fast of data0 : signal is false;`

```
(library work
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell counter (cellType GENERIC)
       (view verilog (viewType NETLIST)
         (interface
           (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT)
           (property xc_fast (integer 1))
 )
```

When place and route is run, a line is added for each output port bit with an xc_fast attribute in the following files:

```
E:\srm\xc_fast\rev_1\pr_1\counter.pad(257):P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMOS25*|2|12
|FAST||||UNLOCATED|NO|NONE|
E:\srm\xc_fast\rev_1\pr_1\counter_map.mrp(172): | DATA0[0] | IOB| OUTPUT | LVCMOS25|| 12| FAST ||
E:\srm\xc_fast\rev_1\pr_1\counter_pad.txt(258):|P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMOS25*|2|12|
FAST||||UNLOCATED |NO|NONE |
E:\srm\xc_fast\rev_1\pr_1\proj_3.edf(193):     (property FAST (string ""))
E:\srm\xc_fast\rev_1\pr_1\counter_map.xrpt(239):  <item  label="Slew&#xA;Rate"  stringID="SLEW_RATE"
value="FAST"/>
E:\srm\xc_fast\rev_1\pr_1\counter_par.xrpt(1450):  <item  label="Slew&#xA;Rate"  stringID="Slew_Rate"
value="FAST"/>
E:\srm\xc_fast\rev_1\pr_1\proj_3.xise(223): <property xil_pn:name="Output Slew Rate" xil_pn:value=
"Fast"  il_pn:valueState="default"/>
```

xc_fast_auto

Attribute; Xilinx (Virtex families). By default (xc_fast_auto value of 1), the synthesis tool infers the fast output buffers OBUF_F_24, OBUFT_F_24, and IOBUF_F_24. Use this global attribute to force inference of slow buffers. In HDL source code you specify xc_fast_auto for the top-level module or architecture. You can override this attribute on an individual basis by using the xc_padtype attribute.

The xc_fast_auto attribute has no affect on output ports specified with the xc_padtype attribute.

Constraint File Syntax and Example

define_global_attribute xc_fast_auto {0|1}

For example:

```
define_global_attribute xc_fast_auto {0};
```

Verilog Syntax and Example

object **/* synthesis xc_fast_auto = 0 | 1 */ ;**

For example:

```
module top (q, d, addr, we, clk) /*synthesis xc_fast_auto=0 */;

// Other code
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute xc_fast_auto of *object* **:** *objectType* **is false | true ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
   port (q : out std_logic_vector(3 downto 0);
         d : in std_logic_vector(3 downto 0);
         addr : in std_logic_vector(2 downto 0);
         we : in std_logic;
         clk : in std_logic );
end ramtest;

architecture beh of top is
attribute xc_fast_auto : boolean;
attribute xc_fast_auto of beh : architecture is false;

-- Other code
```

xc_global_buffers

Attribute; Xilinx. Controls the number of global buffers used in a device. The synthesis tool automatically adds global buffers for clock nets with high fanout. Use this attribute to specify a maximum number of buffers and restrict the amount of global buffer resources added. Also, if there is a black box in the design that has global buffers, you can use xc_global_buffers to prevent the synthesis tool from inferring clock buffers or exceeding the number of global resources. You can only specify this attribute through a constraint file (cannot be specified in HDL source code).

The xc_global_buffers attribute has the same effect as the syn_global_buffers attribute. If both attributes are specified in the same Xilinx design, syn_global_buffers overwrites xc_global_buffers. See *syn_global_buffers*, on page 1339 for details about this attribute.

It is recommended that you use the Xilinx global buffers attribute syn_global_buffers instead; see *syn_global_buffers*, on page 1339.

Constraint File Syntax and Example

define_global_attribute xc_global_buffers {*maximum***}**

For example:

```
define_global_attribute xc_global_buffers {3}
```


Copyright © 2013 Synopsys, Inc.

xc_isgsr

Directive; Xilinx (XC4000 families). Specifies that a port on a black box is connected to an internal STARTUP block. Use this directive with designs targeting XC4000 families where the synthesis tool infers a STARTUP block.

Constraint File Syntax and Example

define_attribute {*instance.resetPort***} xc_isgsr {1}**

For example:

```
define_attribute {bbgsr.gsrin} xc_isgsr {1}
```

Verilog Syntax and Example

object **/* synthesis xc_isgsr = 1 */ ;**

For example:

```
module bbgsr(gsrin, a, b); /* synthesis syn_black_box */
input gsrin /* synthesis xc_isgsr = 1 */;

// Other code
```

VHDL Syntax and Example

attribute xc_isgsr of *object* **:** *objectType* **is true ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library IEEE;
use IEEE.std_logic_1164.all;
library synplify;

entity bbgsr is
   port (gsrin : in std_logic;
         b: in std_logic;
         a : out std_logic );
attribute xc_isgsr : boolean;
attribute xc_isgsr of gsrin : signal is true;
end bbgsr;

architecture bb of bbgsr is
attribute syn_black_box : boolean;
attribute syn_black_box of bb : architecture is true;

-- Other code
```


Copyright © 2013 Synopsys, Inc.

xc_loc

Attribute; Xilinx. Specifies the location (placement) of ports. Refer to the Xilinx databook for valid placement locations. This attribute can be specified in a top-level source file or the constraint file.

It is recommended that you use the Xilinx location attribute syn_loc instead; see *syn_loc*, on page 1388.

Constraint File Syntax and Example

define_attribute {*portDesignName***} xc_loc {***placements***}**

The following example is a constraint file assignment of a pad location to all bits of an 5-bit bus.

```
define_attribute {DATA0[4:0]} xc_loc {P14,P12,P11,P5,P21}
```

Verilog Syntax and Example

object **/* synthesis xc_loc = "***placements***" */ ;**

Here is a Verilog example:

```
module test(a ,b, clk, out1);
   input clk;
   input [2:0]a;
   input [2:0]b;
   output [2:0] out1;
   reg [2:0] out1/* synthesis xc_loc = "P14,P12,P11" */;

always@(posedge clk)
begin
   out1 <= a + b;
end
endmodule
```

VHDL Syntax and Example

attribute xc_loc of *object* **:** *objectType* **is "***placements***" ;**

Here is a VHDL example:

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
   generic (s : integer := 4);

   port (
      clk: in std_logic;
      in1: in std_logic_vector(s downto 0);
      in2: in std_logic_vector(s downto 0);
      d_out: out std_logic_vector(9 downto 0) );
      attribute xc_loc : string;
      attribute xc_loc of d_out : signal is "P14,P12,P11,P5,P21";
end test;

architecture beh of test is
begin
   process (clk)
   begin
      if rising_edge(clk) then
         d_out <= in1 & in2;
      end if;
   end process;
end beh;
```

Examples: xc_loc on Individual Bus Bits

You can use xc_loc to specify locations of individual bits of the bus in one of two ways. This example specifies the bit:

```
define_attribute {valid[0]} xc_loc {R6}
define_attribute {valid[1]} xc_loc {T2}
define_attribute {valid[2]} xc_loc {R5}
define_attribute {valid[3]} xc_loc {R1}
```

This example uses the b: prefix as well as the bit slice:

```
define_attribute {b:valid[0]} xc_loc {R6}
define_attribute {b:valid[1]} xc_loc {T2}
define_attribute {b:valid[2]} xc_loc {R5}
define_attribute {b:valid[3]} xc_loc {R1}
```


Copyright © 2013 Synopsys, Inc.

xc_map

Attribute; Xilinx (Virtex families). Specifies that a design unit is an fmap, hmap, or lut primitive. This attribute is used in conjunction with the xc_rloc and xc_uset. See *Xilinx I/O Support*, on page 2096 for further information.

Constraint File Syntax and Example

define_attribute {v:*primitiveName***} xc_map {fmap|hmap|lut}**

For example:

```
define_attribute {v:hmap_xor4} xc_map {fmap}
```

Verilog Syntax and Example

object **/* synthesis xc_map = "fmap | hmap | lut" */ ;**

For example:

```
module fmap_xor4(z, a, b, c, d) /* synthesis xc_map="fmap" */;
```

VHDL Syntax and Example

attribute xc_map of *object* **:** *objectType* **is "fmap | hmap | lut" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
architecture rtl of fmap_xor4 is
attribute xc_map : string;
attribute xc_map of rtl : architecture is "fmap";
```

xc_padtype

Attribute; Xilinx (Virtex families). Specifies an I/O buffer standard. For example, applying AGP to an IBUF results in the use of an IBUF_AGP instead of an IBUF primitive. Refer to *I/O Standards*, on page 489 or to the Xilinx documentation for a list of I/O buffer standards.

It is recommended that you use the Xilinx I/O buffer standard attribute syn_pad_type instead; see *syn_pad_type*, on page 1440.

Constraint File Syntax and Examples

define_attribute {*port*} xc_padtype {*buffer_standard*}

An underscore (_) is required between the *buffer* and the *standard*.

Examples:

```
define_attribute {a[3:0]} xc_padtype {IBUF_GTLP}
define_attribute {bidir[3:0]} xc_padtype {IOBUF_AGP}
define_attribute {q[3:0]} xc_padtype {OBUF_CTT}
```

Verilog Syntax and Example

object /* **synthesis xc_padtype = "***buffer_standard***"** */ **;**

For example:

```
module test_padtype(a, b, clk, rst, en, bidir, q);
   input [3:0] a /* synthesis xc_padtype = "IBUF_AGP" */, b;
   input clk, rst, en;
   inout [3:0] bidir /* synthesis xc_padtype = "IOBUF_CTT" */;
   output [3:0] q /* synthesis xc_padtype = "OBUF_F_12" */;

// Other code
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax and Example

attribute xc_padtype of *object* **:** *objectType* **is "***buffer_standard***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee, synplify;
use ieee.std_logic_1164.all;

entity test_padtype is
   port (a : in std_logic_vector(3 downto 0);
         b : in std_logic_vector(3 downto 0);
         clk, rst, en : in std_logic;
         bidir : inout std_logic_vector(3 downto 0);
         q : out std_logic_vector(3 downto 0) );
attribute xc_padtype : string;
attribute xc_padtype of a : signal is "IBUF_SSTL3_I";
attribute xc_padtype of bidir : signal is "IOBUF_HSTL_III";
attribute xc_padtype of q : signal is "OBUF_S_8";
```

Copyright © 2013 Synopsys, Inc.

xc_props

Attribute; Xilinx. This is an obsolete attribute that was used to specify the Xilinx attributes to forward-annotate to the gate-level netlist. These attributes are now forward-annotated directly with synthesis attribute statements as shown below. For step-by-step procedures of different ways to specify INIT values, see *Initializing Xilinx RAM*, on page 398 in the *User Guide*.

Constraint File Syntax

Old xc_props

```
define_attribute {FOO_0} xc_props {"HU_SET=FOOBAR,
  RLOC_ORIGIN=X5Y2, RLOC=X0Y0"}
define_attribute {FOO_1} xc_props {"HU_SET=FOOBAR,
  RLOC=X0Y1"}
```

New synthesis attribute

```
define_attribute    {i:FOO_0} HU_SET {FOOBAR}
define_attribute    {i:FOO_0} RLOC_ORIGIN {X5Y2}
define_attribute    {i:FOO_0} RLOC {X0Y0}
define_attribute    {i:FOO_1} HU_SET {FOOBAR}
define_attribute    {i:FOO_1} RLOC {X0Y1}
define_attribute    {i:FOO_0} syn_useioff {0}
define_attribute    {i:FOO_1} syn_useioff {0}
```

Verilog Syntax

Old xc_props

```
FD FOO_0 (.D(D0),.Q(Q0),.C(CLK))
  /* synthesis xc_props="HU_SET=FOOBAR, RLOC_ORIGIN=X5Y2,
RLOC=X0Y0" */;
  FD FOO_1 (.D(D1),.Q(Q1), .C(CLK))
  /* synthesis xc_props="HU_SET=FOOBAR, RLOC=X0Y1" */;;
```

New synthesis attribute

```
FD FOO_0 (.D(D0),.Q(Q0),.C(CLK))
  /* synthesis HU_SET="FOOBAR" RLOC_ORIGIN="X5Y2"
     RLOC="X0Y0" */;
  FD FOO_1 (.D(D1),.Q(Q1), .C(CLK))
  /* synthesis HU_SET="FOOBAR" RLOC="X0Y1" */;
```


Copyright © 2013 Synopsys, Inc.

VHDL Syntax

Old xc_props

```
architecture struct of attr is
  attribute xc_props : string;
  attribute xc_props of FOO_0:  label is "HU_SET=FOOBAR,
    RLOC_ORIGIN=X5Y2, RLOC=X0Y0";
  attribute xc_props of FOO_1:  label is "HU_SET=FOOBAR,
    RLOC=X0Y1";
begin
FOO_0 : FD port map(D => D0, Q => Q0, C => CLK);
FOO_1 : FD port map(D => D1, Q => Q1, C => CLK)
```

New synthesis attribute

```
architecture struct of attr is
  attribute HU_SET : string;
  attribute RLOC_ORIGIN : string;
  attribute RLOC : string;
  attribute HU_SET of FOO_0:  label is "FOOBAR";
  attribute RLOC_ORIGIN of FOO_0:  label is "X5Y2";
  attribute RLOC of FOO_0:  label is "X0Y0";
  attribute HU_SET of FOO_1:  label is "FOOBAR";
  attribute RLOC of FOO_1:  label is "X0Y1";
begin
FOO_0 : FD port map(D => D0, Q => Q0, C => CLK);
FOO_1 : FD port map(D => D1, Q => Q1, C => CLK);
```

If you want to specify different synthesis attributes values for each register bit on a bus, currently you must specify the values separately in the constraint file; you cannot do it in the source code. See *INIT Values*, on page 1617 and *RLOC Constraints*, on page 1618 for details.

Register Preservation

The xc_props attribute also worked as a syn_preserve directive, and ensured that the registers on which it was set were not optimized away:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
//Original xc_props attribute
reg [3:0] rst_cntr /* synthesis xc_props="INIT=1" */;
initial rst_cntr = 4'hf;

always @(posedge clk)
   if (|rst_cntr)
      rst_cntr <= rst_cntr - 1'b1;
      assign rst_cntr_out = rst_cntr;
endmodule
```

With the new synthesis syntax, registers can be optimized. This is because the synthesis tool does not use the INIT="1" attribute, but merely forwards the attribute in the EDIF netlist. In the following example, the register rst_cntr[3:0] is optimized away by the synthesis tool because the tool assumes that rst_cntr[3:0] initializes to all zeroes, so the expression conditional-if (|rst_cntr) is never executed:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
//New synthesis attribute
reg [3:0] rst_cntr /* synthesis INIT="1" */;
   always @(posedge clk)|
      if (|rst_cntr)
         rst_cntr <= rst_cntr - 1'b1;
         assign rst_cntr_out = rst_cntr;
endmodule
```

Verilog INIT Example with syn_preserve

To ensure that you keep the register, you must explicitly set the syn_preserve directive, as shown in this Verilog example:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
//New synthesis attribute
reg [3:0] rst_cntr /* synthesis INIT="1" syn_preserve=1 */;
   always @(posedge clk)
      if (|rst_cntr)
         rst_cntr <= rst_cntr - 1'b1;
         assign rst_cntr_out = rst_cntr;
endmodule
```


Copyright © 2013 Synopsys, Inc.

VHDL INIT Example with syn_preserve

This is a similar VHDL example with an explicit syn_preserve directive:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all ;

entity init_attr is
   port (clk:in std_logic;
         rst_cntr_out: out std_logic_vector(3 downto 0) );
end init_attr;

architecture behave of init_attr is
signal rst_cntr: std_logic_vector(3 downto 0);
attribute INIT: string;
attribute INIT of rst_cntr : signal is "1";
attribute syn_preserve : boolean;
attribute syn_preserve of rst_cntr : signal is true;

begin
   process (clk)
   begin
      if rising_edge(clk) then
         if (rst_cntr /= 0) then
            rst_cntr <= rst_cntr-1;
         end if;
      end if;
   end process;
rst_cntr_out <= rst_cntr;
end behave;
```

Constraint File and syn_preserve

Even if you specify the register INIT values in the constraint file as shown below (INIT="1"), you must still include the syn_preserve directive in the source code to ensure that the register is retained during optimization.

```
define_attribute {rst_cntr[3:0]} INIT {"1"}
```

This value is then embedded in the EDIF netlist:

```
(instance (rename rst_cntr_0 "rst_cntr[0]") (viewRef PRIM
   (cellRef FD (libraryRef UNILIB)))
   (property INIT (string "1"))
)
```

INIT Values

You can use the synthesis syntax to pass INIT values. However, Xilinx ISE 8.2sp3 and later versions require that the INIT value be a string rather than an integer. This means that you must specify the INIT value in quotes:

Constraint	`define_attribute {i:rst_cntr} INIT {"1"}`
Verilog	`reg [3:0] rst_cntr /* synthesis INIT="1" */;`
VHDL	`attribute INIT: string;` `attribute INIT of rst_cntr : signal is "1";`

Currently, if you want to specify different synthesis attributes (such as INIT values) for each register bit on a bus, you must use the constraint file. You cannot do it in the source code. Note that for register INIT, if you want to retain the register during optimization, you must still specify syn_preserve in the source code, because syn_preserve is a directive. So, you set syn_preserve only in the source code and specify different INIT values for each register bit on a bus in the constraint file. The following Verilog example shows syn_preserve set in the source code:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
reg [3:0] rst_cntr /* synthesis syn_preserve=1 */;
always @(posedge clk)
   if (|rst_cntr)
      rst_cntr <= rst_cntr - 1'b1;
assign rst_cntr_out = rst_cntr;
endmodule

module top(input clk, output [3:0] rst_cntr_outver, output [3:0]
   rst_cntr_outvhdl );
init_attr init_attrver (rst_cntr_outver, clk);
endmodule
```

To specify different bit values for this register, you must include constraints in the constraint file, as shown below. The i: prefix in the example indicates that it is an instance, and the period is the hierarchical separator. The module init_attrver is instantiated within the top-level module that contains the rst_cntr[3:0] register. You need not name the top-level module in the hierarchical path. If the instance is in the top-level module, you do not have to specify a hierarchical path. The example specifies different bit values for the bus. Each value is enclosed in quotes so that it is passed to the Xilinx ISE tool as a string value, as required by that tool.

Copyright © 2013 Synopsys, Inc.

```
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"}
```

The tool forward-annotates INIT values as is, so it is up to you to ensure that the values are in the right binary format for place-and-route.

RLOC Constraints

This section describes how to pass RLOC and RLOC_ORIGIN attributes to the Xilinx P&R tool.

VHDL RLOC Example

```
library ieee;
use ieee.std_logic_1164.all;

entity attr_vhdl is
   port (clk, d: in std_logic;
         q_out: out std_logic );
end attr_vhdl;

architecture behave of attr_vhdl is
signal q: std_logic;
attribute RLOC_ORIGIN : string;
attribute RLOC_ORIGIN of behave : architecture is "X0Y2";
attribute RLOC : string;
attribute RLOC of q : signal is "X0Y0";
attribute syn_useioff : boolean;
attribute syn_useioff of q : signal is false;

begin
   process (clk)
   begin
      if rising_edge(clk) then
         q <= d;
      end if;
   end process;

q_out <= q;
end behave;
```

Verilog RLOC Example

```
module attr (output q,input clk, input d);
reg FOO_0 /* synthesis RLOC_ORIGIN="X0Y2" RLOC="X0Y0"
   syn_useioff = 0 */;

always @(posedge clk)
FOO_0 <= d;
assign q = FOO_0;
endmodule
```

As with the INIT values described in *INIT Values*, on page 1617, you cannot apply the individual RLOC values to a bus using a single RLOC synthesis attribute in the source code:

```
reg [3:0] tmp /* synthesis RLOC="X3Y0 X2Y0 X1Y0 X0Y0" */;
```

This causes the following error in the ISE tool:

```
ERROR:Map:6 - Bad format for RLOC constraint "X3Y0 X2Y0 X1Y0
   X0Y0" on FD
```

Instead, specify RLOC_ORIGIN for the Verilog module or VHDL architecture, and then specify individual RLOCs for the registers in the constraint file. The following Verilog example and the subsequent constraints illustrate this technique.

```
module attr (clk, di, do) /* synthesis RLOC_ORIGIN="X5Y0" */;
input  clk;
input  [3:0] di;
output [3:0] do;
reg    [3:0] tmp /* synthesis syn_useioff = 0 */;

always @(posedge clk)
   begin
      tmp <= di;
   end
assign do = tmp;
endmodule
```

You then define the RLOCs for the individual registers in the constraint file as follows:

```
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"}
```


Copyright © 2013 Synopsys, Inc.

xc_pullup/xc_pulldown

Attribute; Xilinx (Virtex families). Specifies that a port is either a pullup or pulldown.

Constraint File Syntax and Example

define_attribute {*portName*} xc_pullup {1}
define_attribute {*portName*} xc_pulldown {1}

For example, you can apply the xc_pullup/xc_pulldown attribute as follows:

- On a multi-bit port such as RXD[3:0]

  ```
  define_attribute {RXD[3:0]} xc_pullup {1}
  ```

- On a single-bit port such as RXA0

  ```
  define_attribute {RXA0} xc_pullup {1}
  ```

- If you want to define this attribute on the nth bit of the multi-bit port RXD[3:0], then you must separate out the output port RXD[*n*]. For example, apply the attribute on the single-bit port RXD_*n*.

  ```
  define_attribute {RXD_1} xc_pullup {1}
  ```

  Otherwise, you cannot define this attribute on a single bit of a multi-bit port.

For the Verilog and VHDL code examples that follow, apply the syntax below if you are using a constraint file.

```
define_attribute {qrs[3:0]} xc_pulldown {1}
define_attribute {data[3:0]} xc_pullup {1}
```

Verilog Syntax and Example

object **/* synthesis xc_pulldown = 1 */ ;**

For example:

```
module dff1(clk, reset, set, data, qrs);
input clk, reset, set;
input [3:0] data /* synthesis xc_pullup = 1 */;
output [3:0] qrs /* synthesis xc_pulldown = 1 */;
reg [3:0] qrs;

// Other code
```

VHDL Syntax and Example

attribute xc_pulldown of *object* **:** *objectType* **is true ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
   port (clk, reset, set : in std_logic;
         data : in std_logic_vector(3 downto 0);
         qrs: out std_logic_vector(3 downto 0) );
attribute xc_pullup : boolean;
attribute xc_pullup of data : signal is true;
attribute xc_pulldown of qrs : signal is true;
end dff1;
```


Copyright © 2013 Synopsys, Inc.

xc_rloc

Attribute; Xilinx (Virtex families). Specifies the relative locations for component instances. This attribute is used in conjunction with xc_uset and xc_map. Use this attribute to specify locations for components grouped together using xc_uset. See *Xilinx I/O Support*, on page 2096 for information.

Constraint File Syntax and Example

define_attribute {*designName***} xc_rloc {***instanceName***}**

Examples:

```
define_attribute {x03} xc_uset {SET1}
define_attribute {x03} xc_rloc {R0C0.f}

define_attribute {x47} xc_uset {SET1}
define_attribute {x47} xc_rloc {R0C0.g}

define_attribute {zz} xc_uset {SET1}
define_attribute {zz} xc_rloc {R0C0.h}
```

Verilog Syntax and Example

object **/* synthesis xc_rloc = "***instanceName***" */ ;**

For example:

```
module clb_xor9(z, a);
output z;
input [8:0] a;
wire z03, z47;

fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
   (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
   (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
   (z, z03, z47, a[8]);
endmodule
```

VHDL Syntax and Example

attribute xc_rloc of *object* **:** *objectType* **is "***instanceName***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
   port (a : in std_logic_vector(8 downto 0);
         z : out std_logic );
end clb_xor9;

architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component hmap_xor3
   port (a : in std_logic;
         b : in std_logic;
         c : in std_logic;
         z : out std_logic );
end component;

component fmap_xor4
   port (a : in std_logic;
         b : in std_logic;
         c : in std_logic;
         d : in std_logic;
         z : out std_logic );
end component;

attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";

begin
x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
zz : hmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```


Copyright © 2013 Synopsys, Inc.

xc_slow

Attribute; Xilinx (Coolrunner and Coolrunner-II). Specifies that the transition time of the driver of an output port is slow (the default). You can speed up the transition time with the xc_fast attribute.

The synthesis tool provides attributes, passed into the XNF/EDIF netlist for Xilinx placement and routing, that affect the input setup times and output transition times for your I/Os. The xc_slow timing attribute gets passed to the Xilinx I/O Block parameter as SLOW in the XNF/EDIF netlist.

The transition time of the output driver can be programmed as either fast or slow. The xc_slow attribute increases the transition time which reduces the noise level in the circuit.

Constraint File Syntax and Example

define_attribute {*outputPort*} xc_slow {1}

For example:

```
define_attribute {DATA0[5]} xc_slow {1}
```

Verilog Syntax and Example

object **/* synthesis xc_slow = 1 */ ;**

For example:

```
output [7:0] DATA0 /* synthesis xc_slow = 1 */;
```

VHDL Syntax and Example

attribute xc_slow of *object* **:** *objectType* **is true ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
attribute xc_slow of DATA0: signal is true;
```

xc_use_keep_hierarchy

Attribute; Xilinx. Available only in the Synplify Pro and Synplify Premier tools. Preserves the hierarchy of the marked module for diagnostic purposes and timing simulation. The marked modules are not flattened in the place-and-route tool, and their hierarchy is preserved.

When active, the software adds the Xilinx attribute KEEP_HIERARCHY=TRUE in the EDIF file for all marked compile points and hierarchical blocks.

xc_use_keep_hierarchy Implementation Details

- Make sure that all blocks with this attribute are registered.
- You must use xc_use_keep_hierarchy together with the syn_hier attribute with a value of "hard" (*syn_hier*, on page 1344) to preserve the hierarchy of the module.
- Use this attribute in conjunction with the xc_area_group compile point attribute (*xc_area_group*, on page 1595).
- When this attribute is applied, the synthesis tool does not do boundary optimizations, so use the attribute with caution, because it can compromise design performance.
- When specified as a global attribute, all compile points in the design are marked (KEEP_HIERARCHY=TRUE) as independent hierarchical modules.

Constraint File Syntax and Example

define_attribute {*module*} xc_use_keep_hierarchy {0|1}

define_global_attribute xc_use_keep_hierarchy {0|1}

Examples:

```
define_global_attribute xc_use_keep_hierarchy
define_attribute {test} xc_use_keep_hierarchy
```

Verilog Syntax and Example

object **/* synthesis xc_use_keep_hierarchy = "0 | 1" */ ;**

For example:

Copyright © 2013 Synopsys, Inc.

```
Module test (in1, in2,....out1,out2)
   /* xc_use_keep_hierarchy=1 */;
```

VHDL Syntax and Example

attribute xc_use_keep_hierarchy of *object* **:** *objectType* **is false | true ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
attribute xc_use_xmodule : Boolean;
attribute xc_use_keep_hierarchy of my_arch : architecture is true;
```

xc_use_rpms

Attribute; Xilinx (Virtex-II Pro, Virtex-4, and *Spartan-3).* Specifies how to handle relationally placed macros (RPMs) during global placement and physical synthesis. Xilinx IP cores might contain RPMs that can have illegal placement, which causes the Synplify Premier tool to error out and generate a message on a workaround for the placement error.

You can use the xc_use_rpms attribute to instruct the tool how to handle RPMs for your design. Values are as shown in this table:

0	Removes all RPMs for global placement and physical synthesis. With this value, RPMs can be moved during global placement and physical synthesis. This eliminates the chance of physical synthesis errors related to illegal placement, however, QoR advantages for using RPMs are missed.
1	(Default). Maintains RPMs for global placement and physical synthesis. With this value, the QoR advantages of using RPMs are maintained, however, if any of the RPMs are illegally placed, physical synthesis will error out.
2	Honors RPMs during global placement and removes them for physical synthesis. With this value, some QoR advantages can be maintained for global placement, however, the tool has the option to optimize placement during physical synthesis.

Constraint File Syntax and Example

define_global_attribute xc_use_rpms {*value*}

where *value* is a string value of 0, 1, or 2.

Copyright © 2013 Synopsys, Inc.

xc_use_timespec_for_io

Attribute; Xilinx (Virtex and Spartan families). Uses the TIMESPEC FROM ... TO command for writing flop-to-out and in-to-flop I/O constraints to the Xilinx constraint (`ncf`) file.

By default, the Xilinx OFFSET keyword is used for writing flop-to-out and in-to-flop I/O constraints to the Xilinx constraint (`ncf`) file for forward annotation to placement and routing. To use the TIMESPEC FROM ... TO command instead, you globally assign the xc_use_timespec_for_io attribute a value of 1. The default value of the attribute is 0 (use OFFSET).

When TIMESPEC FROM ... TO is used, only the data delay is taken into account by the Xilinx place-and-route tool; when OFFSET is used, the clock arrival time and clock delay are also taken into account. For more information on the behavior of these keywords, refer to the appropriate Xilinx documentation.

Note: When the xc_use_timespec_for_io attribute is enabled (1), there is *no* relaxation of constraints that are forward-annotated to the `ncf` file.

Constraint File Syntax and Example

define_global_attribute xc_use_timespec_for_io {0|1}

For example:

```
define_global_attribute xc_use_timespec_for_io {1}
```

Verilog Syntax and Example

object **/* synthesis xc_use_timespec_for_io = 0 | 1 */ ;**

For example:

```
module test (clk,rst,a,b,out1)
   /* synthesis xc_use_timespec_for_io = 0 */;
// Other code
endmodule
```

VHDL Syntax and Example

attribute xc_use_timespec_for_io of *object* **:** *objectType* **is false | true;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
   port (clk, rst, a, b : in std_logic;
         out1 : out std_logic );
end test;

architecture behave of test is
attribute xc_use_timespec_for_io : boolean;
attribute xc_use_timespec_for_io of behave : architecture is
false;

-- Other code
end behave;
```


Copyright © 2013 Synopsys, Inc.

xc_uset

Attribute; Xilinx (Virtex families). Assigns a group name to component instances. This attribute is used with the xc_rloc and xc_map attributes to specify a relative location to a group of components instances. See *Xilinx I/O Support*, on page 2096, for more information.

Constraint File Syntax and Example

define_attribute {*instanceName***} xc_uset {***groupName***}**

Examples:

```
define_attribute {x03} xc_uset {SET1};
define_attribute {x03} xc_rloc {ROCO.f};

define_attribute {x44} xc_uset {SET1};
define_attribute {x44} xc_rloc {ROCO.g};

define_attribute {zz} xc_uset {SET1};
define_attribute {zz} xc_rloc {ROCO.h};
```

Verilog Syntax and Example

object **/* synthesis xc_uset = "***groupName***" */ ;**

For example:

```
module clb_xor9(z, a);
output z;
input [8:0] a;
wire z03, z47;
fmap_xor4 x03 /* synthesis xc_uset = "SET1" xc_rloc = "R0C0.f" */
   (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /* synthesis xc_uset = "SET1" xc_rloc = "R0C0.g" */
   (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /* synthesis xc_uset = "SET1" xc_rloc = "R0C0.h" */
   (z, z03, z47, a[8]);
endmodule
```

VHDL Syntax and Example

attribute xc_uset of *object* **:** *objectType* **is "***groupName***" ;**

See *VHDL Attribute and Directive Syntax*, on page 963 for different ways to specify VHDL attributes and directives.

```
library synplify;
architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component hmap_xor3
   port (a, b, c : in std_logic;
         z : out std_logic );
end component;

component fmap_xor4
   port (a, b, c : in std_logic;
         z : out std_logic );
end component;

attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";

begin
x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
zz : hmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```


Copyright © 2013 Synopsys, Inc.

Summary of Global Attributes

Design attributes in the synthesis environment can be defined either globally, (values are applied to all objects of the specified type in the design), or locally, values are applied only to the specified design object (module, view, port, instance, clock, and so on). When an attribute is set both globally and locally on a design object, the local specification overrides the global specification for the object.

In general, the syntax for specifying a global attribute in a constraint file is:

define_global_attribute *attribute_name* **{***value***}**

The table below contains a list of attributes that can be specified globally in the synthesis environment.

For complete descriptions of any of the attributes listed below, see *Summary of Attributes and Directives*, on page 1169.

Global Attribute	Can Also Be Set On Design Objects
syn_allow_retiming	x
syn_allowed_resources	x
syn_auto_insert_bufg	
syn_auto_partition Attribute	
syn_clean_reset	
syn_disable_purifyclock	x
syn_dspstyle	
syn_edif_bit_format	
syn_edif_name_length	
syn_edif_scalar_format	
syn_force_pads	x
syn_forward_io_constraints	
syn_global_buffers	x

Copyright © 2013 Synopsys, Inc.

Global Attribute	Can Also Be Set On Design Objects
syn_map_dffrs	
syn_max_memsize_reg	
syn_multstyle	x
syn_netlist_hierarchy	
syn_noarrayports	
syn_noclockbuf	x
syn_packer_effort_level	
syn_preserve_sr_priority	
syn_ramstyle	x
syn_reduce_controlset_size	
syn_reduce_controlset_size (Xilinx)	x
syn_replicate	x
syn_romstyle syn_romstyle (Altera) syn_romstyle (Lattice) syn_romstyle Attribute (Lattice iCE65/iCE40) syn_romstyle Attribute (Xilinx)	x
syn_rw_conflict_logic	x
syn_srlstyle	x
syn_useioff syn_useioff (Altera) syn_useioff (Lattice) syn_useioff (Lattice iCE65/iCE40) syn_useioff (Xilinx)	x
syn_use_xilinx_placement	
syn_user_instance	x

Copyright © 2013 Synopsys, Inc.

Global Attribute	Can Also Be Set On Design Objects
xc_fast_auto	
xc_global_buffers	
xc_use_keep_hierarchy	x
xc_use_rpms	
xc_use_timespec_for_io	

Copyright © 2013 Synopsys, Inc.

CHAPTER 17

Batch Commands and Scripts

This chapter describes Tcl commands and scripts.

- Introduction to Tcl, on page 1636
- Tcl Commands for Synthesis, on page 1637
- synhooks File Syntax, on page 1719
- Log File Commands, on page 1721
- High-Reliability Tcl Commands, on page 1723
- VIF Commands, on page 1726
- Netlist Editing Commands, on page 1733
- Tcl Script Examples, on page 1766

Copyright © 2013 Synopsys, Inc.

Introduction to Tcl

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

Tcl scripts are text files that have a `tcl` file extension and contain a set of Tcl commands designed to complete a task or set of tasks. In the Synplify Pro and Synplify Premier tools, you can also run Tcl scripts through the Tcl window (see *Tcl Script Window*, on page 85).

The Synopsys FPGA Tcl commands are described here. For information on the standard Tcl commands, syntax, language, and conventions, refer to the Tcl online help (Help->TCL Help).

Tcl Conventions

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

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

```
designs/big_design/test.v
```

Tcl Scripts and Batch Mode

For procedures for creating Tcl scripts and using batch mode, see *Working with Tcl Scripts and Commands*, on page 1139 in the *User Guide*:

- Running Batch Mode on a Project File, on page 1132

- Running Batch Mode with a Tcl Script, on page 1133
- Generating a Job Script, on page 1140
- Creating a Tcl Synthesis Script, on page 1141
- Using Tcl Variables to Try Different Clock Frequencies, on page 1143
- Running Bottom-up Synthesis with a Script, on page 1145

Tcl Commands for Synthesis

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

- add_file, on page 1639
- add_folder, on page 1645
- assign_to_region, on page 1645
- command_history, on page 1646
- constraint_file, on page 1648
- create_region, on page 1649
- define_link, on page 1650
- dh_module_sources, on page 1651
- generate_instance_constraints, on page 1652
- get_env, on page 1653
- get_option, on page 1653
- hdl_define, on page 1654
- hdl_param, on page 1654
- impl, on page 1656
- job, on page 1657
- launch_system_designer, on page 1658
- open_design, on page 1659

Copyright © 2013 Synopsys, Inc.

- open_file, on page 1660
- partdata, on page 1660
- project, on page 1662
- project_data, on page 1669
- project_file, on page 1670
- project_folder, on page 1671
- recording, on page 1673
- run_config, on page 1674
- run_tcl, on page 1676
- sch_sim_disp, on page 1676
- sch_sim_load, on page 1677
- sch_sim_watch, on page 1679
- set_option, on page 1679
- slice_primitive, on page 1714
- sxml2pxml Tcl Command, on page 1716
- ucftosdc Tcl Command, on page 1004
- Technology-Specific Tcl Commands, on page 1717

Copyright © 2013 Synopsys, Inc.

add_file

The add_file command adds one or more files to a project.

Syntax

add_file [**-***filetype*] *fileName* [*fileName* [...]]

add_file -verilog *fileName* [*fileName* [...]] [**-folder** *folderName*]

add_file -vhdl [**-lib** *libName*[*libName*]] *fileName* [*fileName* [...]] [**-folder** *folderName*]

add_file -include *fileName* [*fileName* [...]]

add_file [**-***filetype*] **-job_owner par** | **simulation** [*fileName* [...]]

add_file -clearbox_verilog *fileName* [*fileName* [...]]

add_file -clearbox_vhdl *fileName* [*fileName* [...]]

add_file -placement_constraint *fileName*

add_file -tooltag *tooltagName* **-toolargs** [*toolArguments*] *fileName*

add_file -vlog_std *standard fileName* [*fileName* [...]]

add_file -ucf *fileName* [*fileName* [...]]

Copyright © 2013 Synopsys, Inc.

-filetype	Specifies the type of file being added to the project (files are placed in folders according to their file types; including this argument overrides automatic filename-extension placement). See *Filename Extensions*, on page 1642 for a list of the recognized file types.
fileName	Specifies the name of the file being added to the project. Files are added to the individual project folders according to their filename extensions (View Project Files in Folders must be set in the Project View Options dialog box). You can add multiple files by separating individual filenames with a space, and you can specify different file types (extensions) within the same command.
-verilog or -vhdl	Adds HDL files with non-standard extensions to the Verilog or VHDL directory, so that they can be compiled with the project. For example, the following command adds the file alu.v.new to the project's verilog directory: `% add_file -verilog /designs/megachip/alu.v.new` If you do not specify -verilog, the file is added to the Other directory (new is not a recognized Verilog extension), and the file would not be compiled with the files in the Verilog directory.
[-lib *libName*]	Specifies the library associated with VHDL files. The default library is work. The -lib option sets the VHDL library to *libName*. Note: You can also specify multiple libraries for VHDL files. For example: `add_file -vhdl -lib {mylib,work} "ff.vhd"` Both the logical and physical libraries must be specified in the Project file (if you only specify the logical library associated with the VHDL files, the compiler treats the module as a black box).

[-folder *folderName*]	Creates logical folders with custom files in various hierarchy groupings within your Project view. For example:
	`add_file -verilog -folder memory "ram_1.v"`
	`add_file -verilog -folder memory "C:/examples/verilog/common_rtl/memory/ram_1.v"`

-clearbox_verilog -clearbox_vhdl	Designates the specified file as an Altera Clearbox model file and copies it to the *implementationName*/par_1 directory that is created after synthesis. When you run Quartus from the synthesis interface with the Run P&R option, it requires that the Clearbox files be in this directory.

Copyright © 2013 Synopsys, Inc.

-include	Indicates that the specified file is to be added to the project as an include file (include files are added to the Include directory regardless of their extension). Include files are not passed to the compiler, but are assumed to be referenced from within the HDL source code. Adding an include file to a project, although not required, allows it to be accessed in the user interface where it can be viewed, edited, or cross-probed.
-job_owner	Allows you to determine how files are used; you can specify these options from the File Options dialog box. For example, you can automatically decide to pass files to the backend place-and-route tool (Use for Place and Route) or use them for test benches containing HDL constructs for simulation (Use for Simulation only).

-placement_constraint	Adds the timing or placement constraint file (pdc) generated by Import Xilinx Constraints to the project file. The pdc and timing constraint files use the same format; the extension was renamed to denote that this file be used in the Physical Plus flow. You can also add the timing constraint file to the Physical Plus flow. For example:
	`add_file -placement_constraint` *fileName*`_ucf.pdc`
	`add_file -placement_constraint` *fileName*`_ncd.pdc`

Copyright © 2013 Synopsys, Inc.

-tooltag *tooltagName*	Creates a tool tag name for the application tool you want to invoke from within the Synopsys FPGA synthesis tools. For example: `add_file -tooltag {Xilinx Coregen} "ram.v"`
-toolargs tool	Specifies any argument options to use with the application tool you want to invoke from within the Synopsys FPGA synthesis tools. For example: `add_file -tooltag {System Designer} -toolargs {$SynCode} "ram.v"`
-vlog_std *standard*	Overrides the global Verilog standard for an individual file. The accepted values for *standard* are v95 (Verilog 95), v2001 (Verilog 2001), and sysv (SystemVerilog). The file (*fileName*) is added to the Verilog folder in the project; the specified standard is listed after the filename in the project view and is enclosed in angle brackets (for example, commchip.v <sysv>). Note that when you add a SystemVerilog file (a file with an sv extension) to a project, the add_file entry in the project file includes the -vlog_std *standard* string. The default standard for new projects is SystemVerilog. For Verilog 2005 extensions, use sysv (SystemVerilog).

Filename Extensions

Files with the following extensions are automatically added to their corresponding project directories; files with any other extension are added to the Other directory. The *-filetype* argument overrides automatic filename extension placement.

Extension	-Filetype	Project Folder
.adc	-analysis_constraint	Analysis Design Constraint
.cdc[1]	-compiler_constraint	Compiler Directive
.edf, .edn	-edif	EDIF
.fdc	-fpga_constraint/-constraint	Logic Constraints (FDC)
.ngc, .ngo	-xilinx	Xilinx File
.nrf[1]	-nrf	Netlist Restructure
.opt	-xflow	P&R Options

Copyright © 2013 Synopsys, Inc.

Extension	-Filetype	Project Folder
.pdc[1]	-physical_constraint	Physical Constraints
.prt	-partition	Partition Scripts
.ptf	-ptf	Prototyping Constraint
.sdc	-constraint	Logic Constraints (SDC)
.sfp[1]	-floorplan	Design Plan
.sv[2]	-verilog	Verilog
.tcl	-tcl	Tcl Script
.ucf	-xilinx	Xilinx Constraint File
.v	-verilog	Verilog
.vhd, .vhdl	-vhdl	VHDL
any	-include	Include
-	-clearbox_verilog	Clearbox Verilog
-	-clearbox_vhdl	Clearbox VHDL

1. Available in Synplify Premier tools only.
2. Use the sv format for SystemVerilog keyword support. Both Verilog and SystemVerilog formats are added to the Verilog folder.

Example: Add Files

Add a series of VHDL files to the VHDL directory and add an include file to the project:

```
% add_file /designs/sequencer/top.vhd
% add_file /designs/sequencer/alu.vhdl
% add_file -vhdl /designs/sequencer/reg.vhd.fast
% add_file -include /designs/std/decode.vhd
```

The corresponding directory structure in the Project view is shown in the following figure:

Copyright © 2013 Synopsys, Inc.

Example: Designate Clearbox Files

Designate some files as Clearbox model files for Quartus:

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

Example: File Options Designation

Designate some IP core wrappers as well as their associated instantiated component files that must be passed on to the place-and-route tool, since they are not written to the final netlist:

```
add_file -verilog -job_owner par "my_ip_core.v"
add_file -verilog -job_owner par "my_ip_core_enc.v"
```

Example: Add Xilinx UCF Files

Add Xilinx user constraint files (`ucf`) to the project:

```
add_file -ucf "./test.ucf"
```

You can add multiple `ucf` files to the project. However, the synthesis tool does not honor these `ucf` files. They are copied to the `synplicity.ucf` file in the order that they were added to the project and forward annotated. If the `ucf` file contains timing constraints only (for example, clock constraints or path constraints), it is recommended that you run ucf2sdc to convert the `ucf` file to `sdc` constraints before running synthesis.

Copyright © 2013 Synopsys, Inc.

add_folder

The add_folder command adds a custom folder to a project.

Syntax

add_folder *folderName*

Creates logical folders with files in various custom hierarchy groupings within your Project view. These custom folders can be specified with any name or hierarchy level.

```
add_folder verilog
add_folder verilog/common_rtl
add_folder verilog/common_rtl/prep
```

For more information about custom folders, see *Managing Project File Hierarchy*, on page 220 in the *User Guide*.

assign_to_region

The assign_to_region command assigns start and end points of the critical paths to a region created in the Design Planner. You need to determine which critical paths are collected for $*sel* constraints that are assigned to the region.

The assign_to_region command is supported in the Synplify Premier tool.

Syntax

assign_to_region *regionName* **$***sel*

The following table describes the command arguments.

Option	Description
regionName	Name of the region created for the floorplan, such as rgn1.
$*sel*	Determine which critical path start and end points can be collected and are placed in $*sel* and which are then assigned to the floorplan region you created.

Copyright © 2013 Synopsys, Inc.

Example

The following example contains the Tcl expand commands to find the RTL instances for the start and end points of the critical path and create a region for these assignments.

```
TCL EXPAND COMMANDS FOR RTL INSTANCE FIND AND REGION ASSIGNMENT:

set start [define_coll  [find -inst {mem_add[9:0]} -hier]
[find -inst {mau_yu[1:0]} -hier] ]
set end [find -inst {yu_add[9:0]} -hier]
set sel [expand -from $start -to $end -hier]
select $assignments1
create_region region1 21 36 32 31
assign_to_region region1 $assignments1
```

If the collection of start and end points is placed in assignments and the region name is rgn1, then use the following command to assign them to the region:

```
assign_to_region rgn1 $assignments
```

command_history

Displays a list of the Tcl commands executed during the current session.

Syntax

command_history [**-save** *filename*]

Arguments and Options

-save

Writes the list of Tcl commands to the specified *filename*.

Description

The command_history command displays a list of the Tcl commands executed during the current session. Including the -save option, saves the commands to the specified file to create Tcl scripts.

Copyright © 2013 Synopsys, Inc.

Examples

```
command_history -save C:/DesignsII/tut/proto/myTclScript.tcl
```

See Also

- *recording*, on page 1673

Copyright © 2013 Synopsys, Inc.

constraint_file

The constraint_file command manipulates the constraint files used by the active implementation.

Syntax

constraint_file
-enable *constraintFileName*
-disable *constraintFileName*
-list
-all
-clear

The following table describes the command arguments.

Option	Description
-enable	Selects the specified constraint file to use for the active implementation.
-disable	Excludes the specified constraint file from being used for the active implementation
-list	Lists the constraint files used by the active implementation
-all	Selects (includes) all the project constraint files for the active implementation.
-clear	Clears (excludes) all the constraint files for the active implementation

Examples

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

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

% constraint_file -disable eight_bit_uc.fdc
```

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

```
% constraint_file -clear
```

Copyright © 2013 Synopsys, Inc.

```
% constraint_file -enable clocks2.fdc
```

create_region

The create_region command is used to create a floorplan region. The create_region command is only supported in the Synplify Premier tool.

Syntax

create_region *regionName coordinates*

The following table describes the command arguments.

Option	Description
regionName	Name of the region you create on the floorplan, such as rgn1.
coordinates	Location of the floorplan region you create. Specify the lower-left and upper-right CLB corners of the region.

Example

The following example creates a region called rgn1 and specifies its CLB locations on the floorplan.

```
create_region rgn1 21 36 32 31
```


Copyright © 2013 Synopsys, Inc.

define_link

Use the define_link command to add or remove an instance or design block to/from the subproject mapping.

Syntax

define_link [{**i** |**v**}: {*instanceName* | *designBlockName*}]
[**-name** *linkSubProjectName*] [**-run_type** *value*] [**-remove**] [**-list**] [**-parameter**]

The following table describes the define_link command options.

Option	Description
i: *instanceName* or v: *designBlockNeme*	Specifies the name of the instance or design block to be linked for the subproject.
-name	Specifies the name of the subproject implementation that contains the design block.
-run_type *value*	Defines how the subproject is run. You can specify: • top-down – Runs the top-down flow from RTL or SRS. • bottom-up – Runs the bottom-up flow from EDIF.
-remove	Removes the link to an instance or design block from the subproject database. When the -remove option is not used, the instance or design block is removed from the top-level project and added as a subproject for that project.
-list	Lists all existing links for the project.
-parameter	Maps the subproject base on the parameterized design block.

Examples

Link the instance or design block to the subproject implementation that contains the block definition. For example:

```
"define_link "i:inst1" -name proj1_1|rev_1"
```

or

```
"define_link "v:designblockA" -name proj2|rev_3"
```

Copyright © 2013 Synopsys, Inc.

dh_module_sources

Use the dh_module_sources command to return a list of the source files defining a module hierarchy. The dh_module_sources command is only supported in the Synplify Premier tool.

Syntax

dh_module_sources [**-of**] [**-noself**] *moduleName*

The following table describes the dh_module_sources command options.

Option	Description
moduleName	Specifies the module (view) name of the module hierarchy.
-of	Includes list of sources referencing the module.
-noself	Excludes sources in which the module definition occurs.

Examples

```
dh_module_sources work.sub2.verilog
```

```
dh_module_sources -of -noself work.sub2.vhdl
```


Copyright © 2013 Synopsys, Inc.

generate_instance_constraints

The generate_instance_constraints command generates the timing and resource constraints for the instance-based subproject(s) of a hierarchical design.

Syntax

generate_instance_constraints [**-time_budget** [*instanceName*]] [**-resources** [*instanceName*]] [**-all** [*instanceList*]]

The following table describes the generate_instance_constraints command options.

Option	Description
instanceName	Specifies the instance name of the subproject.
instanceList	Specifies the names for selected instances of the subprojects in the design hierarchy.
-time_budget	Specifies the timing constraints (for example, the clock definitions, I/O delays, and timing exceptions) for the instance-based subproject.
-resources	Specifies the resource constraints (for example, compile points and I/O standards) for the instance-based subproject.
-all	Specifies both the timing and resource constraints for the selected instances of subprojects in the design hierarchy.

Examples

This example generates timing and resource constraints for all instance-based subprojects.

```
generate_instance_constraints -all
```

This example generates timing constraints for all instance-based subprojects.

```
generate_instance_constraints -time_budget
```

This example generates timing and resource constraints for instances I1, I2, and I3 of the subprojects.

```
generate_instance_constraints -all I1 I2 I3
```

Copyright © 2013 Synopsys, Inc.

get_env

The get_env command reports the value of a predefined system variable.

Syntax

get_env *systemVariable*

Use this command to view system variable values. The following example shows you how to use the get_env command to see the value of the previously created MY_PROJECT environment variable. The MY_PROJECT variable contains the path to an HDL file directory, so get_env reports this path.

```
get_env MY_PROJECT
```

```
d:\project\hdl_files
```

In the project file or a Tcl script, you can define a Tcl variable that contains the environment variable. In this example, my_project_dir contains the MY_PROJECT variable, which points to an HDL file directory.

```
set my_project_dir [get_env MY_PROJECT]
```

Then, use the $*systemVariable* syntax to access the variable value. This is useful for specifying paths in your scripts, as in the following example which adds the file myfile1.v to the project.

```
add_file $my_project_dir/myfile1.v
```

get_option

The get_option command reports the settings of predefined project and device options. The options are the same as those for set_option. See *set_option*, on page 1679 for details.

Syntax

get_option -*optionName*

Copyright © 2013 Synopsys, Inc.

hdl_define

For Verilog designs, this command specifies values for Verilog text macros. You can specify text macro values that you would normally enter using the Verilog `define statement in a Verilog file included at the top of the synthesis project. The parameter value is valid for the current implementation only.

This command is equivalent to the set_option -hdl_define command.

Syntax

hdl_define
-set "*directive***=***value***"**
-list

Example:

```
hdl_define -set "SIZE=32"
```

This statement specifies the value 32 for the SIZE directive; the following statement is written to the project file:

```
set_option -hdl_define -set "SIZE=32"
```

To define multiple directive values using hdl_define, enclose the directives in quotes and use a space delimiter. For example:

```
hdl_define -set "SIZE=32 WIDTH=8"
```

The software writes the following statement to the `prj` file:

```
set_option -hdl_define -set "size=32 width=8"
```

See Also

Compiler Directives and Design Parameters, on page 215 for information on specifying compiler directives in the GUI.

hdl_param

The hdl_param command shows or sets HDL parameter overrides. For the GUI equivalent of this command, select Project->Implementation Options->Verilog/VHDL.

Copyright © 2013 Synopsys, Inc.

Syntax

hdl_param
-add {*paramName***}**
-list | -set *paramName* **{***paramValue***}**
-clear
-overrides

The following table describes the command arguments.

Option	Description
-add	Adds a parameter override to the project.
-list	Shows parameters for the top-level module only and lists values for parameters if there is a parameter override.
-set	Sets a parameter override and its value for the active implementation. Only the parameter value is enclosed within curly braces.
-clear	Clears all parameter overrides of the active implementation.
-overrides	Lists all the parameter override values used in this project.

Examples

In batch mode, to set generic values using the set_option command in a project file, specify the hdl_param generic with quotes and enclose it within {}. For example:

```
set_option -hdl_param -set ram_file {"init.mem"}

set_option -hdl_param -set simulation {"false"}
```

Suppose the following parameter is set for the top-level module.

```
set_option -hdl_param -set {"width=8"}
```

Add a parameter override and its value, then list the parameter override.

```
hdl_param -add {"size=32"}
hdl_param -list "size=32"
```


Copyright © 2013 Synopsys, Inc.

impl

The impl command adds, removes, or modifies an implementation. This command is available only in the Synplify Pro and Synplify Premier tools.

Syntax

impl
-add [*implName*] [*model*]
-name *implName*
-remove *implName*
-active [*implName*]
-list
-type *implType*
-result_file
-dir

The following table describes the command arguments.

Option	Description
-add	Adds a new device implementation. If: • *implName* is not specified, creates a unique implementation name by incrementing the name of the active implementation. • you want to add a new implementation copied from implementation *model*.
-name	Changes the name of the active implementation.
-remove	Removes the specified implementation.
-active	Reports the active implementation. If you specify an implementation name, changes the specified name to the active implementation.
-list	Lists all the implementations used in this project.
-type	Specifies the type of implementation to add. For example, the: • -type fpga option creates an FPGA implementation. • -type identify option creates an Identify implementation.
-result_file	Displays the implementation results file.
-dir	Displays the implementation directory.

Copyright © 2013 Synopsys, Inc.

Examples

List all implementations, report the active implementation, then make a different implementation active.

```
% impl -list
design_worst design_typical design_best

% impl -active
design_best

%impl -active design_typical

% impl -active
design_typical

% impl -add rev_1_identify mixed -type identify
```

job

The job command, for place and route job support, creates, removes, identifies, runs, cancels, and sets/gets options for named P&R jobs.

Syntax

job *jobName* [**-add** *jobType* |**-remove** |**-type** |**-run** [*mode*] |**-cancel** |
-option *optionName* [*optionValue*]]

job -list

The following table describes the command options.

Option	Description
-run	Runs the P&R job, according to the specified options:
-add *jobType*	Creates a new P&R job for the active implementation.
-cancel	Cancels a P&R job in progress.
-remove	Removes a P&R job from an active implementation
-list	Returns a list of the P&R jobs in the active implementation.
-remove	Removes a P&R job from the active implementation

Copyright © 2013 Synopsys, Inc.

Option	Description
-option *optionName* [*optionValue*]	Get/set options for *jobName*.
-type	Returns the P&R job type.

Examples

```
% job pr_2 -add par
% job pr_2 -option enable_run 1
% job pr_2 -option run_backannotation 1
% job pr_2 -run
```

launch_system_designer

The launch_system_designer command starts the System Designer tool, which lets you select, configure, and assemble internal and third-party IP in the IP-XACT format, integrate that IP, and then implement it in an FPGA vendor device.

Syntax

launch_system_designer [**-board_file** *boardName*] [**-batch**]

The following table describes the command options.

Option	Description
-board_file *boardName*	Specifies an optional HAPS single-FPGA board file (vb) to be used in the System Designer design. You can only specify files for the following supported boards: haps-31, haps-51, or haps-51t. You cannot specify a multi-FPGA board file. See the *System Designer User Guide* for information on using the board file.
-batch	Runs System Designer in batch mode.

open_design

The open_design command specifies a netlist file (srs, srp, or srm) that can be used to search the database with the Tcl find command in batch mode. With open_design, you can use find without having to open an RTL, partitioned RTL, or Technology view. Use open_design to read in the srs, srp, or srm file before issuing the find command. See the example below.

Syntax

open_design [**-shared 0**|**1**] *filename*

Where:

- -shared — indicates if the database is to be shared (read-only). Setting this switch to 0 allows the database being searched to be updated which requires additional memory resources.
- *filename* is the RTL (srs) , partitioned RTL (srp), or Technology (srm) file that can be used to search the database.

Example

```
project -load ../examples/vhdl/prep2_2.prj
open_design prep2_2.srs
set a [find -inst *]
c_print $a -file a.txt
open_design prep2_2.srm
set b [find -net *]
c_print $b -file b.txt
```

In the example above, prep2_2 is loaded and the information from the RTL view file is read in. Then, the find command searches for all instances in the design and prints them to file a. Next, the technology view file is read in, then find searches for all nets in the design and prints them to file b.

See Also

- *Tcl find Command*, on page 1774.

Copyright © 2013 Synopsys, Inc.

open_file

The open_file command opens views within the tool. The command accepts two arguments: -rtl_view and -technology_view.

Syntax

open_file -rtl_view |-technology_view

The -rtl_view option displays the RTL view for the current implementation, and the -technology_view option displays the technology view for the current implementation. Views remain displayed until overwritten and multiple views can be displayed.

partdata

The partdata command loads part files and returns information regarding a part such as available families, family parts, vendors, attributes, grades, packages.

Syntax

partdata
-load *filename*
-family
-part *family*
-vendor *family*
-attribute *attribute family*
-grade [*family*:]*part*
-package [*family*:]*part*
-oem [*family*:]*part*

Option	Description
-load *filename*	Loads part file.
-family	Lists available technology families.
-part *family*	Lists all parts in specified family.
-vendor *family*	Returns vendor name for the specified family.

Copyright © 2013 Synopsys, Inc.

Option	Description
-attribute *attribute family*	Returns the value of the job attribute for the specified family.
-grade [*family*:]*part*	Lists the speed grades available for the specified part.
-package [*family*:]*part*	Lists the packages available for the specified part.
-oem [*family*:]*part*	Returns true if the part entered is an OEM part.

Example

The following example prints out the available vendors, their supported families, and the parts for each family.

```
% foreach vendor [partdata -vendorlist]
% puts VENDOR:$vendor;
% foreach family [partdata -family $vendor]
% puts \tFAMILY:$family;
% puts \t\tPARTS:;
% foreach part [partdata -part $family]
% puts \t\t$part;
```

program_terminate

Immediately terminates the tool session without saving any data.

Syntax

program_terminate

Arguments and Options

None

Description

The program_terminate command terminates a tool session without prompting or saving data.

Copyright © 2013 Synopsys, Inc.

Examples

```
program_terminate
```

program_version

Returns software product version.

Syntax

program_version

Arguments and Options

None

Description

The program_version command returns the software product version number.

Examples

```
% program_version
Synplify Pro G-2012.09
```

project

The project command runs job flows to create, load, save, and close projects, to change and examine project status, and to archive projects.

Syntax

project -**run** [**-fg**] [**-all**] [*mode*]

project {**-new** [*projectPath*] |**-load** *projectPath* | **-close** [*projectPath*]
|**-save** [*projectPath*] |**-insert** *projectPath*} |

project {**-active** [*projectName*] |**-dir** |**-file** |**-name** |**-list** |**-filelist** |
-fileorder *filepath1 filepath2* [... *filepathN*] |**-addfile** *filepath* |
-movefile *filepath1* [*filepath2*] |**-removefile** *filepath*}

project {**-result_file** *resultFilePath* |**-log_file** [*logfileName*] }**project -copy** [**-project** *filename*] [**-implement** *implementationName*]
[**-dest_dir** *pathname*] [**-copy_type** {**full** | **local** | **customize**}]
[**-add_srs** [*fileList*] **-no_input**]

project -unarchive [**-archive_file** *pathname/filename*] [**-dest_dir** *pathname*]

The following table describes the command options.

Copyright © 2013 Synopsys, Inc.

Option	Description
-run [**-fg**] [**-all**] [*mode*]	Synthesizes the project, according to the specified options: • **-fg** – Synthesizes in the foreground. • **-all** – Synthesizes all implementations (Synplify Pro and Synplify Premier tools only). The *mode* can be one of the following keywords: • **backannotation** – Runs backannotation only without impacting the synthesis or P&R results. To run backannotation only in batch mode, see *Example: Running Backannotation in Batch Mode*, on page 1668. • **compile** – Compiles the active project, but does not map it. • **constraint_check** - Validates the syntax and applicability of constraints defined in one or more constraint files. • **estimator** – Runs area estimation on the design (Synplify Premier product only). • **fsm_explorer** – Selects optimum FSM-encoding style for finite-state machines. • **syntax_check** – Verifies that the HDL is syntactically correct; errors are reported in the log file. • **synthesis** – Default mode if no mode is specified. Compiles (if necessary) and synthesizes the currently active project. If followed by the -clean option (project -run synthesis -clean), resynthesizes the entire project, including the top level and *all compile points*, whether or not their constraints, implementation options or source code changed since the last synthesis. If not followed by -clean, only compile points that have been modified are resynthesized. • **synthesis_check** – Verifies that the design is functionally correct; errors are reported in the log file. • **timing** – Runs the Timing Analyst. This is equivalent to clicking the Generate Timing button in the Timing Report Generation dialog box with user-specified values.

Copyright © 2013 Synopsys, Inc.

Option	Description
	• **write_netlist** – Writes the mapped output netlist to structural Verilog (vm) or VHDL (vhm) format. You can also use this command in an incremental timing analysis flow. For details, see *Run Menu*, on page 237 and *Generating Custom Timing Reports with STA*, on page 729.
-new [*projectPath*]	Creates a new project in the current working directory. If *projectPath* is specified, creates the project in the specified directory. For the Hierarchical Project Management flow, you can create a new sub-project for the top-level project.
-load *projectPath*	Opens and loads the project file specified by *projectPath*.
-close [*projectPath*]	Closes the currently active project. If *projectPath* is specified, closes the specified project.
-save [*projectPath*]	Saves the currently active project. If *projectPath* is specified, saves the specified project.
-insert *projectPath*	Adds the specified project to the workspace project.
-insert *projectFile*	For the Hierarchical Project Management flow, adds the specified subproject for the top-level project. See *Example: Inserting Subproject for a Top-level Project*, on page 1668.
-active [*projectName*]	Shows the active project. If *projectName* is specified, makes the specified project the active project.
-dir	Shows the project directory for the active project.
-file	Returns the path to the active project.
-name	Returns the filename (prj) of the active project.
-list	Returns a list of the loaded projects.
-filelist	Returns the pathnames of the files in the active project.
-fileorder *filepath1 filepath2* [... *filepathN*]	Reorders files by adding the specified files to the end of the project file list.
-addfile *filepath*	Adds the specified file to the project.

Copyright © 2013 Synopsys, Inc.

Option	Description
-movefile *filepath1* [*filepath2*]	Moves *filepath1* to follow *filepath2* in project file list. If *filepath2* is not specified, moves *filepath1* to top of list.
-removefile *filepath*	Removes the specified file from the project.
-result_file *resultFilePath*	Changes the name of the synthesis result file to the path specified.
-log_file [*logfileName*]	Reports the name of the project log file. If *logfileName* is specified, changes the base name of the log file.

-archive
-project *filename*
[**-root_dir** *pathname*]
-archive_file *filename*.**sar**
-archive_type {**full** | **local** | **customize**}
-add_srs [*fileList*] **-no_input**
	• **project** *filename* – copies a project other than the active project. If you do not use this option, by default the active project is copied.
	• **root_dir** *pathname* – specifies the top-level directory containing the project files.
	• **archive_file** *filename* – is the name of the archived project file.
	• **archive_type** - specifies the type of archive:
	  • full – performs a complete archive; all input and result files are contained in the archive file.
	  • customize – performs a partial archive; only the project files that you select are included in the archive.
	  • local – includes only project input files in the archive; does not include result files.
	• **add_srs** – adds the listed `srs` files to the archived project. Use the -no_input option with this command. If *fileList* is omitted, adds all `srs` files for the project/implementations. The `srs` files are the RTL schematic views that are output when the design is compiled (Run->Compile Only).
	For examples using the project -archive command, see *Archive Examples*, on page 1668.

Option	Description

-copy
-project *filename*
-implement *implementationName*
-dest_dir *pathname*
-copy_type {**full** | **local** | **customize**}
-add_srs [*fileList*] **-no_input**
	• **project** *filename* – copies a project other than the active project. If you do not use this option, by default the active project is copied.
	• **implement** *implementation_name* – archives all files in the specified implementation.
	• **dest_dir** *directory_pathname* – specifies the directory in which to copy the project files.
	• **copy_type** – specifies the type of file/project copy:
	  • full – performs a complete copy; all input and result files are contained in the archive file.
	  • customize – performs a partial copy; only the project files that you select are included in the archive.
	  • local – includes only project input files in the copy; does not include result files.
	• **add_srs** – adds the listed `srs` files to the archived project. Use the -no_input option with this command. If *fileList* is omitted, adds all `srs` files for the project/implementations. The `srs` files are the RTL schematic views that are output when the design is compiled (Run->Compile Only).

-unarchive **-archive_file** *pathname/filename* **-dest_dir** *pathname*	• **archive_file** *pathname/filename* – is the name of the archived project file. • **dest_dir** *pathname* – specifies the directory in which to write the project files. For examples using the project -unarchive command, see *Un-Archive Example*, on page 1669.

Examples

Load the project top.prj and compile the design without mapping it. Compiling makes it possible to create a constraint file with the SCOPE spreadsheet and display an RTL schematic representation of the design.

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

Load a project and synthesize the design.

Copyright © 2013 Synopsys, Inc.

```
% project -load top.prj
% project -run synthesis
```

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

Example: Inserting Subproject for a Top-level Project

You can add a project within a project using the following syntax:

```
project -insert [projectFile]
```

This command inserts a subproject within the top-level design, which becomes the active project.

```
% project -insert "./block2/block2.prj"
% project -insert "./block3/block3.prj"
% project -insert "./block1/block1.prj"
% project -insert "./control/control.prj"
```

Example: Running Backannotation in Batch Mode

To run backannotation only on the placed and routed data to get accurate timing and placement information back to the Synplify Premier tool, enter the following commands in a Tcl file:

```
project -load projectName.prj
project -run backannotation
project -close projectName.prj
```

Then source this Tcl file in batch mode. For example:

```
synplify_premier_dp -batch -tcl ba.tcl
```

Archive Examples

The following example archives all files in the project and stores the files in the specified `sar` file:

```
project -archive -project c:/proj1.prj
      -archive_file c:/archive/proj1.sar
```

The next example archives the project file (`prj`) and all local input files into the specified `sar` file.

```
project -archive -project c:/proj1.prj -archive_type local
      -archive_file c:/archive/proj1.sar
```

The following example archives the project file (prj) only for selected srs files into the specified sar file. Any input source files that are in the project are not included.

```
project -archive -project c:/proj1.prj -archive_type customize
      -add_srs -no_input -archive_file c:/archive/proj1.sar
```

Un-Archive Example

The following example extracts the project files from c:/archive/proj1.sar to directory c:/proj1. All directories and sub-directories are created if they do not already exist.

```
project -unarchive -archive_file c:/archive/proj1.sar
         -dest_dir c:/proj1
```

Copy Examples

The following example copies only selected srs files for the project to the destination project file directory.

```
project -copy -project d:/test/proj_2.prj -copy_type customize
         –add_srs –no_input -dest_dir d:/test_1
```

The next example copies all input source files and srs files selected for the project to the destination project file directory.

```
project -copy -project d:/test/proj_2.prj -copy_type customize
         -dest_dir d:/test_1
```

project_data

The project_data command shows or sets properties of a project.

Syntax

project_data {**-active** [*projectName*] | **-dir** | **-file** }

The following table describes the command options.

Copyright © 2013 Synopsys, Inc.

Option	Description
-active	Set/show active project. With no argument, shows the active project. If *projectName* is specified, changes the active project to *projectName*.
-dir	Show directory of active project.
-file	Show the project file for the active project. The full path is included with the file name.

project_file

The project_file command manipulates and examines project files.

Syntax

project_file {**-lib** *fileName* [*libName*] | **-name** *fileName* [*newPath*] |
-time *fileName* [*format*] | **-date** *fileName* | **-type** *fileName* |
-savetype *fileName* [**relative** | **absolute**] **-move** *fileName1* [*fileName2*] |
-remove *fileName* | **-top** *topModule* |
-tooltag *applicationTagName* | **- toolargs** [*arguments*] *fileName* }

The following table describes the command options.

Option	Description
-lib	Shows the project file library associated with *fileName*. If *libName* is specified, changes the project file library for the specified file to *libName*.
-name	Shows the project file path for the specified file. If *newPath* is specified, changes t1he location of the specified project file to the directory path specified by *newPath*.

Copyright © 2013 Synopsys, Inc.

Option	Description
-time	Shows the file time stamp. If a *format* is specified, changes the composition of the time stamp according to the combination of the following time formatting codes: **%H** (hour 00-23) **%M** (minute 00-59) **%S** (second 00-59) **%d** (day 01-31) **%b** (abbreviated month) **%Y** (year with century)
-date	Shows the file date.
-type	Shows the file type.
-savetype	Sets or shows whether a file is saved relative to the project or its absolute path.
-move	Positions *fileName1* after *fileName2* in HDL file list. If *fileName2* is not specified, moves *fileName1* to the top of the list.
-remove	Removes the specified file from the project file list.
-top	Sets or shows the top-level module of the specified file for the active implementation.
-tooltag	Sets or shows the third-party tool tag for the specified file.
-toolargs	Sets or shows the third-party tool tag arguments for the specified file.

Examples

List the files added to a project. Remove a file.

```
% project -filelist path_name1/cpu.v path_name1/cpu_cntrl.v
   path_name2/cpu_cntrl.vhd

% project_file -remove path_name2/cpu_cntrl.vhd
```

project_folder

The project_folder command manipulates and examines attributes for project folders.

Copyright © 2013 Synopsys, Inc.

Syntax

project_folder [*folderName*] [**-folderlist**] [**-filelist**] [**-printout**] [**-add**] [**-remove**] [**-r**] [**-tooltag**] [**-toolargs**]

The following table describes the command options.

Option	Description
folderName	Specifies the name of the folder for which attributes are examined.
-folderlist	Lists folders contained in the specified project folder.
-filelist	Lists files contained in the specified project folder.
-printout	Prints the specified project folder hierarchy including its files.
-add	Adds a new project folder.
-remove	Removes the specified project folder.
-r	Removes the specified project folder and all its containing sub-folders. Files are removed from the project folder, but are not deleted.
-tooltag	Sets or shows the third-party tool tag name.
-toolargs	Sets or shows the additional arguments for the third-party tool tag.

Examples

Add a folder and list the files added to a project folder.

```
% project_folder -add newfolder
% project_folder -filelist newfolder
```

Copyright © 2013 Synopsys, Inc.

recording

Allows you to record and store the Tcl commands generated when you work on your projects in the GUI. You can use this command for creating job scripts. The complete syntax for the recording command is:

recording {**-on**|**-off** **-file** [*historyLogFile*] | **-save** [*historyLogFile*] } **-state**

In the command line:

- **on|off**– turns Tcl command recording on (1) or off (0). Recording mode is off by default.
- **file** – if you specify a history log file name, this option uses the specified file in which to store the recorded Tcl commands for the current session. If you do not specify a history log name, reports the name of the current history log file.
- **save** – if you do not specify a file name, updates the current history log. If you specify a history log file name, saves Tcl command history to the specified file.
- **state** – returns the Boolean value of recording mode.

Examples

Turn on recording mode and save the Tcl commands in the cpu_tcl_log file created.

```
% recording -on
% recording -file cpu_tcl_log
```

report_clocks

Reports the clocks in the design database.

Syntax

report_clocks -netlist [*srsNetlistFile*] [**-csv_format**] [**-out** *fileName*]

Copyright © 2013 Synopsys, Inc.

Arguments and Options

srsNetlistFile
: The name of the srs netlist file. If this optional argument is not specified, the netlist file is taken from the active project implementation.

-csv_format
: Displays the report in spread-sheet format.

-out
: Specifies the name of the output report file (default name is *designName*_clk.rpt).

Description

The report_clocks command generates a report of the clocks found in the design database. The report includes a listing of the clock domain, parent clock, and clock type for each clock. If the -csv_format option is included, the report is output in spread-sheet format.

Examples

```
report_clocks c:/designs/mem_ctrl/mem_ctrl.srs -csv_format
```

run_config

The run_config command lets you set up the subproject implementations and configure how to run these implementations with the top-level project. You can specify:

- Which implementations to run for each subproject.
- Whether to run the subproject top-down or bottom-up.
- Which options to synchronize for all the subprojects.

Copyright © 2013 Synopsys, Inc.

Syntax

run_config
-add *implementationList* [**-impl** *implementationName*]
-set *implementationList* [**-impl** *implementationName*]
-run_type *runType* [**-subproject** *projectName*] [**-impl** *implementationName*]
-list
-clear
-reset_default

The following table describes the run_config command options.

Option	Description
-add	Lets you add subproject implementations to run with the top-level project for this configuration. Specify *implementationName* as { *projectName*| *implName* }.
-set	Lets you select the subproject implementations to run with the top-level project for this configuration. Specify *implementationName* as { *projectName*| *implName* }.
-impl	Specifies the parent implementation to apply for this configuration run. The default is the active implementation.
-subproject	Specifies the path for the subproject.
-run_type	Specifies how to run the subprojects. You can choose: • top_down • bottom_up
-list	Lists the current subproject implementations to run with the top-level project for this configuration.
-clear	For this configuration, removes all the subprojects for the top-level project.
-reset_default	Resets the subprojects back to its original settings to run with the top-level project for this configuration.

Examples

Set the subproject implementations for the top-level project to run with the current configuration.

Copyright © 2013 Synopsys, Inc.

```
run_config -set -impl rev_1
{block2/block2.prj|rev_1 block3/block3.prj|rev_1
block1/block1.prj|rev_1 control/control.prj|rev_1}
```

Specify whether to run the top_down of bottom_up flow for the specified subproject.

```
run_config -run_type top_down -impl rev_1 -subproject
control/control.prj
```

run_tcl

The run_tcl command lets you synthesize your project using a Tcl script file from the Tcl Script window of the synthesis tool.

Syntax

run_tcl [**-fg**] *tclFile*

You can also use the following command:

source *tclFile*

These commands are equivalent.

The following table describes the run_tcl command options.

Option	Description
-fg	Synthesizes the project in foreground mode.
TclFile	Specifies the name of the Tcl file used to synthesize the project. To create a Tcl Script file, see *Creating a Tcl Synthesis Script*, on page 1141.

sch_sim_disp

The sch_sim_disp command displays simulation values for nets in the Control Panel signal watch list.

Syntax

sch_sim_disp
-out [*filename*]
-append *filename*
-at [*value*] [*netNames*]
-to *value* [*netNames*]
-bracket
-csv
-table

The following table describes the sch_sim_disp command options.

Option	Description
-out	Writes display output to a file. The default is to write to the Tcl window.
-append	Appends data to the end of the specified output file.
-at *value*	Displays value at a specified time. The default is the current time.
-to *value*	Displays range of values from -at to -end.
-bracket	Displays values for times before and after the time range.
-csv	Specifies the CSV format for importing simulation values to a spreadsheet. CSV is a comma separated value text file.
-table	Displays a table format for the simulation values.
netNames	List of net names for which simulation values are displayed. The default is to display all watched nets.

sch_sim_load

The sch_sim_load command loads a VCD simulation file for use in the HDL Analyst view.

Syntax

sch_sim_load [**-top_scope** *value*] [**-validate**] [**-log** *logFile*] [**-start** *value*] [**-end** *value*] [**-timeunits** *value*] [**-scopes** *value*] [**-top_module** *value*] [**-identify**] [**-simulator**] *modulePath* | *VCDPath*

Copyright © 2013 Synopsys, Inc.

The following table describes the sch_sim_load command options.

Option	Description
-top_scope *value*	Scope path through the test bench to the top-level module. This option is only used in simulator mode.
-validate	Validates the VCD file with the root netlist after being loaded.
-log *value*	Allows you to specify a log file to record diagnostic messages.
-start *value*	Specifies the start time for loading the VCD file.
-end *value*	Specifies the end time for loading the VCD file.
-timeunits *units*	Specifies the start/end time units, such as ps; for example: 100ps. The default time unit is pico seconds (ps).
-scopes *value*	Specifies a list of delimited VCD scope paths to load. The default is to load all scope paths.
-top_module *value*	Specifies the name of the top-level module in the VCD file. The default is the netlist root.
-identify	Loads the VCD file in Identify mode.
-simulator	Loads the VCD file in simulator mode.
modulePath	Specifies the module path through the test bench to the root netlist.
VCDPath	Specifies the path to the VCD file.

Example 1: Opens a VCD file generated by VCS in the SRM View. The test bench path to the top-level module is test.x:

```
sch_sim_load -top_scope test.x single1.vcd
```

Example 2: Opens a VCD file generated by Identify in the SRS View:

```
sch_sim_load iice_1.vcd
```

Copyright © 2013 Synopsys, Inc.

sch_sim_watch

The sch_sim_watch command allows you to manage which nets to watch for simulation values.

Syntax

sch_sim_watch
-add [**-coll**] [**-select**] [*names*]
-rem [**-coll**] [*names*]
-list
-clear

The following table describes the sch_sim_watch command options.

Option	Description
-add	You can use the find -net command to select nets and then specify the -add option to watch this collection of nets. Otherwise, identify specific nets to watch.
-rem	Specifies a collection of nets or else specific nets to remove from the VCD Control Panel watch list.
-coll	Specifies the collection *names*.
-list	Specifies the list of nets that are being watched for simulation values.
-clear	Removes all the nets from the VCD Control Panel watch list.

set_option

The set_option command sets options for the technology (device) as well as for the design project.

Syntax

set_option *-optionName optionValue*

For syntax and descriptions of the options and related values, see one of the following tables:

- Device Options for set_option/get_option

Copyright © 2013 Synopsys, Inc.

- Project Options for set_option/get_option

Device Options for set_option/get_option

The following table lists *generic* device arguments for the technology, part, and speed grade. These are the options on the Implementation Options-> Device tab.

Information on all other Implementation Options tabs are listed in the next section, *Project Options for set_option/get_option*, on page 1680.

Option Name	Description
-technology *parameter*	Sets the target technology for the implementation. *parameter* is the string for the vendor architecture. Check the Device panel in the GUI or see *Device Panel*, on page 195, for a list of supported families.
-part *part_name*	Specifies a part for the implementation. Check the Device panel of the Implementation Options dialog box (see *Device Panel*, on page 195) for available choices.
-speed_grade -*value*	Sets the speed grade for the implementation. Check the Device panel of the Implementation Options dialog box (see *Device Panel*, on page 195) for available choices.
-package *value*	Sets the package for the implementation. This option is not available for certain vendor families, because it is set in the place-and-route software. Check the Device panel of the Implementation Options dialog box (see *Device Panel*, on page 195) for available choices.
-grade -*value*	Same as -speed_grade. Included for backwards compatibility.

In general, device options are technology-specific, or have technology-specific defaults or limitations. For vendor-specific details, see *Technology-Specific Tcl Commands*, on page 1717.

Project Options for set_option/get_option

Below is a list of options for the set_option and get_option commands. Click on the option below for the corresponding description and GUI equivalents. Options set through the Device tab are listed in *Device Options for set_option/get_option*, on page 1680.

analysis_constraint
areadelay
area_delay_percent
auto_constrain_io
auto_infer_blackbox
automatic_compile_point
autosm
beta_vfeatures
block
clique/cliquing
compiler_compatible
compiler_constraint
constant_prop
constprop
constraint
continue_on_error
conv_mux_xor_gated_clocks
createhierarchy
dc_root
default_enum_encoding
disable_io_insertion
do_synplify_place
dup
dw_foundation
dw_library
dw_minpower
dw_stop_on_nolic
enable64bit
enable_nfilter
enable_prepacking
enhanced_optimization
fanin_limit
fanout_limit
fast_synthesis
feedthrough
fix_gated_and_generated_clocks
floorplan/sfp
force_async_genclk_conv
force_gsr
frequency
frequency auto
globalthreshold
hamming3
hdl_define
hdl_param
hdl_qload
help
identify_debug_mode
ignore_undefined_libs
include_path
job (PR)
job (BA)
job (PC)
job (SP)
job (xtclsh)
level_hierarchy
library_path
log_file
map_logic
maxfan
maxfan_hard
maxfanin
maxterms
max_parallel_jobs
max_terms_per_macrocel
multi_file_compilation_unit
no_sequential_opt
num_critical_paths
num_startend_points
opcond
par_use_xflow
par_use_xilinx_partition
physical_synthesis
pipe
popfeed
power
preserve_registers
project_relative_includes
report_path
reporting_*type*
resolve_multiple_driver
resource_sharing
result_file
retiming
run_physical_plus
run_prop_extract
run_xilinx_io_clock_place
RWCheckOnRam
safe_case
soft/soft_buffers
supporttypedflt
symbolic_fsm_compiler
syn_altera_model
synthesis_onoff_pragma
timequest
top_module
update_models_cp
use_fsm_explorer
use_vivado
verification_mode
vlog_std
voltage
write_apr_constraint
write_verilog
write_vhdl

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-analysis_constraint *path/filename*.adc	Specifies the analysis design constraint file (adc) you can use to modify constraints for the stand-alone Timing Analyst only.	Constraint File section on the Timing Report Generation Parameters dialog box
-areadelay *percentValue* **-area_delay_percent** *percentValue*	Sets the percentage of paths you want optimized. This option is available only in certain device technologies.	Percent of design to optimize for timing, Device Panel
-auto_constrain_io 1|0	Determines whether default constraints are used for I/O ports that do not have user-defined constraints. When disabled, only define_input_delay or define_output_delay constraints are considered during synthesis or forward-annotated after synthesis. When enabled, the software considers any explicit define_input_delay or define_output_delay constraints, as before.	Use clock period for unconstrained IO check box, Constraints Panel
-auto_infer_blackbox 1|0	*Available only in Synplify Premier* Determines action taken when an undefined Verilog module is encountered. When disabled (0, the default), an undefined module causes the compiler to error out; when enabled, a black box is created for the undefined module and a warning is issued.	Auto Infer Blackbox field, Verilog Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-automatic_compile_point 1|0	Enables/disables the automatic compile point flow, which can analyze a design and identify modules that can automatically be defined as compile points and mapped in parallel using Multiprocessing.	Automatic compile point check box, Options Panel
-autosm 1|0 **-symbolic_fsm_compiler 1|0**	Enables/disables the FSM compiler.	FSM Compiler check box, Options Panel
-beta_vfeatures 1 | 0	Enables/disables the use of Verilog compiler beta features.	Beta Features for Verilog, Verilog Panel
-block 1|0 **-disable_io_insertion 1|0**	Enables/disables I/O insertion in some technologies.	Disable I/O Insertion check box, Device Panel
-clique 1|0 **-cliquing 1|0**	Enables/disables the grouping of logic functions in Altera technologies.	Perform Cliquing check box, Device Panel
-compiler_compatible 1|0	Disables pushing of tristates across process/block boundaries.	*Complement* of the Push Tristates Across Process/ Block Boundaries check box, VHDL Panel and Verilog Panel
-compiler_constraint *constraintFile*	When multiple constraint files are defined, specify which constraint files are to be used from the Constraints tab of the Implementation Options panel.	Constraints Files, Constraints Panel
-constant_prop 1|0 **-constprop 1|0**	*Available only in Synplify Premier* Enables/disables constant propagation for Synplify Premier netlist restructuring when the Enable prototyping tools option is enabled.	Constant Propagation check box, GCC & Prototyping Tools Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-constraint *-option*	Manipulates constraint files in the project: -enable/disable *filename* – adds or removes constraint file from active implementation -list – lists all enabled constraint files in active implementation -all – enables all constraint files in active implementation -clear – disables all constraint files in active implementation	Constraint Files, Constraints Panel
-continue_on_error 1|0	The continue_on_error option serves two related, but separate functions. **Compiler** (Synplify Premier only) – when enabled during compilation, allows the compiler to continue, if possible, after encountering a non-syntax error within a design unit and to resume compilation with the next design unit without stopping. Continuing on a compilation error is only available with the Synplify Premier products. **Mapper** – when enabled during compile-point synthesis, allows the mapping operation to continue on error and synthesize the remaining compile points. Continuing on a post-compilation synthesis error is available with both the Synplify Pro and Synplify Premier products. The default for this option (0) is to stop on any compilation or synthesis error.	Continue on Error, Project View checkbox or Options Panel, or Configure Compile Point Process Command

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-conv_mux_xor_gated_clocks 1|0	Performs gated-clock conversion on muxes or OR gates in the clock tree when enabled (Synplify Premier only). This option is automatically enabled when a HAPS technology is selected.	Convert MUX/XOR Gates Clock Tree, GCC & Prototyping Tools Panel
-createhierarchy 1|0 **-level_hierarchy 1|0**	*Available only in Synplify Premier DP* Creates an additional level of hierarchy for always or process blocks in a design plan region.	Create Always/Process Level Hierarchy check box, GCC & Prototyping Tools Panel
-dc_root *installPath*	*Available only in Synplify Premier* Specifies path to Design Compiler installation for access to Synopsys foundation and minPower DesignWare libraries. The default is $SYNOPSYS.	Design Compiler Installation Location field, VHDL Panel and Verilog Panel
-default_enum_encoding default | onehot | gray | sequential	(VHDL only) Sets the default for enumerated types.	Default Enum Encoding, VHDL panel (see VHDL Panel and Verilog Panel)
-disable_io_insertion 1|0 **-block 1|0**	Enables/disables I/O insertion in some technologies. For more information about the impact of using this command, see *syn_insert_pad*, on page 1376.	Disable I/O Insertion, Device Panel
-do_synplify_place 1|0	*Available only in Synplify Premier* Since the Synplify global placer is run by default with the Physical Synthesis and Physical Plus flows, this option is no longer needed. To run the Xilinx global placer, use the syn_use_xilinx_placement attribute.	Obsolete

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-dup	For Verilog designs, allows the use of duplicate module names. When true, the last definition of the module is used by the software and any previous definitions are ignored. You should not use duplicate module names in your Verilog design, therefore, this option is disabled by default. However, if you need to, you can allow for duplicate modules by setting this option to 1.	Allow Duplicate Modules, Verilog Panel
-dw_foundation 1|0	*Available only in Synplify Premier* Specifies the source of the DesignWare models. When set to 1, uses standard DesignWare building blocks from the Synopsys foundation library. If dw_minpower is also 1, minimum power DesignWare building blocks from the Synopsys minPower library overlay the corresponding models from the DesignWare foundation library.	Use DesignWare Foundation Library checkbox, VHDL Panel and Verilog Panel
-dw_library {dw_foundation [dw_minpower]}	*Available only in Synplify Premier* Specifies the Synopsys foundation library as the source of the DesignWare building blocks (same as -dw_foundation 1). Including the dw_minpower argument overlays the Synopsys DesignWare minPower library over the DesignWare foundation library.	Use DesignWare Foundation Library/ Use DesignWare MinPower Library checkbox, VHDL Panel and Verilog Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-dw_minpower 1|0	*Available only in Synplify Premier* Specifies the source of the DesignWare building blocks. When set to 1, overlays minimum power building blocks from the Synopsys minPower library over the corresponding DesignWare foundation library building blocks.	Use DesignWare MinPower Library checkbox, VHDL Panel and Verilog Panel
-dw_stop_on_nolic 1|0	*Available only in Synplify Premier* Specifies action taken when a Synopsys DesignWare building block is encountered and no license is found; 0 (default) black boxes DesignWare building block and continues, 1 stops with error message.	Stop synthesis if no DesignWare license found checkbox, VHDL Panel and Verilog Panel
-enable64bit 1|0	Enables/disables the 64-bit mapping switch. When enabled, this switch allows you to run client programs in 64-bit mode, if available on your system.	Enable 64-bit Synthesis, Options Panel
-enable_nfilter 1|0	*Available only in tynplify Premier* Enables the following netlist filtering options: -constprop (or -constant_prop), -createhierarchy (or -level_hierarchy), -create_mac_hier, -feedthrough (or -popfeed)	Enable Netlist Optimization, GCC & Prototyping Tools Panel
-enable_prepacking 1|0	*Xilinx Virtex-5 and Spartan-6 or newer devices* Enable this option to prepare the synthesis netlist for advanced LUT combining, so that Xilinx ISE P&R can then pack into the LUT6_2 depending on available resources. Use in ISE version 10.1 service pack 1 or later. The default is on.	Enable Advanced LUT Combining (use only with Xilinx version 10.1 sp1 or later), see Virtex and Spartan-II and Later set_option Tcl Command, on page 2124.

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-enhanced_optimization 1|0	*Available only in Synplify Premier; for certain Altera and Xilinx devices* When enabled, this mode uses additional optimizations during logic synthesis to get better QoR. This is the default. When disabled, the tool runs standard logic synthesis without the additional optimizations. See *Logic Synthesis with Enhanced Optimization*, on page 40 of the *User Guide* for details.	Enhanced Optimization, Device Panel
-fanin_limit *value* **-maxfanin 1|0**	*Available only in Synplify and Synplify Pro; for Lattice ispMach and Altera Max devices* Specifies the maximum fan-in.	Maximum Cell Fanin, Device Panel
-fanout_limit *value* **-maxfan** *value*	*Available in Synplify and Synplify Pro; for Microsemi, Lattice ORCA, and Xilinx technologies* Sets the fanout limit guideline for the current project.	Fanout Guide, Device Panel
-fast_synthesis 1|0	*Available in the Synplify Premier; for supported Altera and Xilinx devices* Reduces mapper optimization to provide faster logic synthesis runtimes at a potential cost of reduced QoR. See *Fast Synthesis*, on page 1061 of the *User Guide* for details.	Fast Synthesis, Options Panel
-feedthrough 1|0 **-popfeed 1|0**	*Available only in the Synplify Premier tools.* Enables/disables feedthrough optimization.	Feedthrough Optimization, GCC & Prototyping Tools Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-fix_gated_and_generated_clocks 1|0	*Available only in Synplify Pro and Synplify Premier* Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 of the *User Guide* for details.	Gated Clocks, GCC & Prototyping Tools Panel
-floorplan *filename*	*Available only in Synplify Premier Design Planner* Specifies the Design Plan file (`sfp`) for the implementation.	Select `sfp` file, Design Planning Panel
-force_async_genclk_conv [**1|0**]	When enabled (1), gated-clock conversion occurs regardless of the presence of asynchronous set/reset signals in generated-clock logic or datapath latches. When disabled (the default), conversion is inhibited when asynchronous set/reset signals on generated-clock logic have signals different from the asynchronous signals on the driven datapath latches.	Force Generated Clock Conversion with Asynchronous Signals, GCC & Prototyping Tools Panel
-force_gsr yes | no | auto	Available in the Synplify Pro and Synplify tools for Lattice FPGA technologies; the default value is no. Enables (yes) or disables forced use of the global set/reset routing resources. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is auto.	Force GSR Usage, Device Panel
-frequency *value*	Sets the global frequency.	Frequency, Constraints Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-frequency auto	*Available only in Synplify Pro* Enables/disables auto constraints.	Auto Constrain, Constraints Panel
-globalthreshold *value*	This option applies only to the following Microsemi technologies: • FUSION • IGLOO/IGLOOE/IGLOO+ • ProASIC3/3E/3L Sets the minimum number of fanout loads. Signals that exceed the load value are promoted to global signals. Global buffers are assigned by the synthesis tool to drive the global signals.	Device Panel
-hamming3 1|0	*Available only in Synplify Premier* When enabled, a Hamming 3 encoding is used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers.	High Reliability Panel
-hdl_define	For Verilog designs; used for extracting design parameters and entering compiler directives.	Compiler Directives and Design Parameters, Verilog Panel
-**hdl_param**	Shows or sets HDL parameter overrides. See *hdl_param*, on page 1654 for command syntax.	Use this command in the Tcl window of the UI.
-hdl_qload 1|0	*Synplify Premier* When you enable the HDL Analyst Quick Load switch, the compiler pre-partitions the design into multiple modules that are written to separate netlist files (`srs`). This feature improves memory usage in the HDL Analyst for large designs.	Options Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-help	This option is useful for getting syntax help on the various implementation options used for compiling and mapping a design. For examples, see *help for set_option*, on page 1706.	Use this command in the Tcl window of the UI.
-identify_debug_mode 1|0	When set option to 1, creates an Identify implementation in the Project view. Then, you can launch the Identify Instrumentor or Debugger from within the FPGA synthesis tools.	Select the Identify implementation, then launch: • Launch Identify Instrumentor or • Launch Identify Debugger
-ignore_undefined_libs 1|0	(VHDL only) When enabled (default), the compiler will ignore any declared library files not included with the source file. In previous releases, the missing library file would cause the synthesis tool to error out. To set this option to error out when a library file is missing (as in previous releases), use 0 for the command value.	Not available in the UI

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-include_path *path*	(Verilog only) Defines the search path used by the ‘include commands in Verilog design files. Argument *path* is a string that is a semicolon-delimited list of directories where the included design files can be found. The software searches for include files in the following order: • First, the source file directory. • Then, looks in the included path directory order and stops at the first occurrence of the included file it finds. • Finally, the project directory. The include paths are relative. Use the project_relative_includes option to update older project files.	Include Path Order, Verilog panel (see Verilog Panel, on page 211)
-job *PR_job_name* **-option enable_run 1|0**	If enabled, runs the specified place-and-route job with the appropriate vendor-specific place-and-route tool after synthesis.	Specify the place-and-route job you want to run for the specified implementation. See Place and Route Panel.
-job *PR_job_name* **-option run_backannotation 1|0**	*Available only in Synplify Premier* If enabled, synthesis backannotates placement and timing data after the appropriate vendor-specific place-and-route job is run.	Backannotate placement and timing data following Place & Route option; see *Running P&R Automatically after Synthesis*, on page 842.

Option	Description	GUI Equivalent
-job *PR_job_name* **-option use_placement_constraints 1|0**	*Available in Synplify Premier; for Altera Stratix III, Stratix II GX, and Stratix II devices* • By default, the Quartus II place-and-route tool uses the placement constraints forward annotated from the physical synthesis tool. • When this option is disabled (not checked), the Quartus II place-and-route tool determines how it will handle physical constraints.	Use placement constraints from physical synthesis option; see *Running P&R Automatically after Synthesis*, on page 842.
-job *PR_job_name* **-add par_place** *PR_job_name*	*Available only in Synplify Premier* Runs the Physical Plus flow place-and-route job with the appropriate vendor-specific tool.	Specify the Physical Plus place-and-route job to run for the selected implementation.
-job *PR_job_name* **-option xtclsh_option_file** *options_filename*.**tcl**	This option allows you to specify the place-and-route options file to use when running the PAR job. You can use this command in the various Xilinx synthesis flows, including the incremental flows.	Add Place & Route Options File; see Add P&R Implementation Popup Menu Command. You can choose an Existing Options File or Create New Options File.
-level_hierarchy 1|0 **-createhierarchy 1|0**	*Available only in Synplify Premier* Creates an additional level of hierarchy for always or process blocks for Synplify Premier netlist restructuring when the Enable prototyping tools option is enabled.	Create Always/Process Level Hierarchy check box, GCC & Prototyping Tools Panel.

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-library_path *directory_pathname*	For Verilog designs, specifies the paths to the directories which contain the library files to be included in your design for the project. Defines the search path used by the tool to include all the Verilog design files for your project. The argument *directory_pathname* is a string that specifies the directories where these included library files can be found. The software searches for all included Verilog files and the tool determines the top-level module.	Library Directories on Verilog Panel.
-log_file *logFileName*	Allows you to change the name for a default log file (both the `srr` and `htm` files). For example: `set_option -log_file test` generates the the following files in the Implementation Directory after synthesis is run: • `test.htm` • `synlog\test_premap.srr` • `synlog\test_fpga_mapper.srr` • `synlog\test_fpga_mapper.srr_Min`	Enter command from the Tcl window
-map_logic 1|0	*Available in Synplify Pro and Synplify; for Altera and Lattice technologies* Turns on direct mapping to technology-specific devices during synthesis.	Map Logic to ATOMs | LCells | Macrocells, Device Panel. Macrocells applies to Lattice; ATOMs and LCells apply to Altera.

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-maxfan *value* **-fanout_guide** *value*	*For Synplify Pro and Synplify tools support Microsemi, Lattice ORCA, and Xilinx technologies. For the Synplify Premier tool supports Xilinx technologies.* Sets the fanout limit for the current project. The limit is value is guideline for the tool rather than a hard limit.	Fanout Guide, Device Panel
-maxfan_hard 1	*Available in Synplify Pro and Synplify; for Microsemi designs* This option specifies that the -maxfan value is a hard fanout limit that the synthesis tool must not exceed it.	Hard Limit to Fanout, Device Panel
-maxfanin 1|0 **-fanin_limit** *value*	*Available only in Synplify Pro and Synplify; for Altera Max and Lattice ispMach designs* Specifies the maximum fan-in.	Maximum Cell Fanin, Device Panel
max_parallel_jobs *n*	Lets you run multiprocessing with compile points. This allows the synthesis software to run multiple, independent compile point jobs simultaneously, providing additional runtime improvements for the compile point synthesis flow. For information on setting the maximum number of parallel synthesis jobs, see *Setting Number of Parallel Jobs*, on page 1140 in the *User Guide*.	Maximum number of parallel mapper jobs, on the Configure Compile Point Process dialog box.
-maxterms *value* **-max_terms_per_ macrocel** *value*	*Available in Synplify Pro and Synplify Premier; for Lattice devices* Sets the maximum number of terminals per macrocell. This option is available only if -map_logic is set to true.	Maximum Terms/Macrocell, Device Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-multi_file_compilation_unit 1|0	When you enable the Multiple File Compilation Unit switch, the Verilog compiler uses the compilation unit for modules defined in multiple files.	Verilog Panel
-no_sequential_opt 1|0	Enables or disables the sequential optimizations for the design. (Note that unused registers will still be removed from the design.) The default value is true (sequential optimizations not performed). When true, delay and area size might increase. Value can be 1 or true, 0 or false. With this option enabled, the FSM Compiler is effectively disabled.	Device Panel
-num_critical_paths *value*	Specifies the number of critical paths to report in the timing report.	Number of Critical Paths, Timing Report Panel
-num_startend_points *value*	Specifies the number of start and end points to include when reporting paths with the worst slack in the timing report.	Number of Start/End Points, Timing Report Panel. Number of Start/End Points, Timing Report Generation dialog box.
-opcond *value*	This option applies only to the Microsemi Axcelerator, Fusion, IGLOO, and ProASIC families of technologies. Sets the operating condition for device performance in the areas of optimization, timing analysis, and timing reports. Values are Default, MIL-WC, IND-WC, COM-WC, and Automotive-WC. See *Operating Condition Device Option*, on page 2049 for more information.	Device Panel

Option	Description	GUI Equivalent
-par_use_xflow 1|0	When enabled, you can use the previous xflow function to run the Xilinx place-and-route tool. This option is turned off by default.	Use Xilinx xflow, Device Panel
-par_use_xilinx_partition 1|0	When enabled, you can use compile points with the Xilinx Partition flow to compare partitions or modules in the previous and current implementations. Partitions that have not changed are preserved from the previous implementation. This increases overall design stability by freezing the implementation for blocks that have met timing. This feature also saves significant time in that the implementation for the entire design does not need to be rerun. This option is turned off by default. You must also enable the Use Xilinx Xflow option (-par_use_xflow 1) as well.	Use Xilinx Partition Flow, Device Panel
-physical_synthesis 1|0	*Available only in Synplify Premier* Enables/disables physical synthesis which includes global placement, placement-aware optimizations, and graph-based placement.	Physical Synthesis, Options Panel
-pipe 1|0	*Available only in Synplify Pro and Synplify Premier* Runs designs at a faster frequency by moving registers into the multiplier, creating pipeline stages.	Pipelining, Device Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-popfeed 1|0 **-feedthrough 1|0**	*Available only in Synplify Premier* When enabled, the software eliminates unused routes through a module after it has been compiled. These optimizations are available when you floorplan the design.	Feedthrough Optimization, GCC & Prototyping Tools Panel
-power 1|0	*Available only in Synplify Premier* When enabled, used to reduce power for designs that contain Xilinx block RAM and DSP48 components.	Power Optimization, Options Panel
-preserve_registers 1|0	*Available only for Microsemi technologies* When enabled, the software uses less restrictive register optimizations during synthesis if area is not as great a concern for your device. The default for this option is disabled (0).	Conservative Register Optimization switch on the Device Panel
-project_relative_includes 1|0	Enables/disables the Verilog include statement to be relative to the project, rather than a verilog file. For projects built with software after 8.0, the include statement is no longer relative to the files but is relative to the project: project_relative (1). See *Updating Verilog Include Paths in Older Project Files*, on page 219 in the *User Guide* for information about updating older project files.	Include Path Order, Verilog Panel
-report_path *integer*	Available for Microsemi 500K, Fusion, IGLOO, and ProASIC family of technologies. Sets the maximum number of critical paths in a forward-annotated SDF constraint file	Max Number of Critical Paths in SDF, Device Panel

Option	Description	GUI Equivalent
-**reporting_***type*	Sets parameters for the stand-alone Timing Analyst report. See *Timing Report Parameters for set_option*, on page 1704 for details.	Analysis->Timing Analyst command: Timing Report Generation Parameters
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver. The default for this option is disabled (0). See *Resolve Mixed Drivers Option*, on page 1707 for details.	Resolve Multiple Drivers, Device Panel
-resource_sharing 1|0	Enables or disables resource sharing. This is a compiler-specific optimization, and does not affect resource sharing in the mapper.	Resource Sharing, Device Panel
-result_file *filename*	Specifies the name of the results file.	Result File Name and Result Format, Implementation Results Panel
-retiming 1|0	*Available only in Synplify Pro and Synplify Premier* When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled).	Retiming, Device Panel
-run_physical_plus 1|0	*Available only in Synplify Premier tools; Xilinx only* Enables the Physical Plus modular flow. After logic synthesis with Enhanced Optimization has been run, you can incrementally run physical synthesis to implement netlist and timing changes using the Process view GUI.	Project View Option or Physical Plus, Options Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-run_prop_extract 1|0	Enables/disables the annotation of certain generated properties relating to clocks and expansion onto the RTL view. This enables the Tcl expand and find commands to work correctly with clock properties.	Options Panel
-run_xilinx_io_clock_place 1|0	When enabled, lets the synthesis software automatically generate the I/O and clock component placement locations required for Physical Plus.	Generate IO & Clock Generating Component Placement for Physical Plus, Constraints Panel
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.	Read Write Check on RAM, Device Panel
-safe_case 1|0	When enabled, the high reliability safe case option turns off sequential optimizations for counters, FSM, and sequential logic to increase the circuit's reliability.	Preserve and Decode Unreachable States (FSM, Counters, Sequential Logic), High Reliability Panel
-soft 1|0 **-soft_buffers 1|0**	*Available only in Synplify Pro and Synplify* Makes non-critical Lcells soft in Altera MAX designs.	Soft LCELLs, Device Panel
-supporttypedflt 1|0	When enabled (1), the compiler passes init values through a syn_init property to the mapper. For more information, see *VHDL Implicit Data-type Defaults*, on page 874.	Implicit Initial Value Support, VHDL Panel

Option	Description	GUI Equivalent
-symbolic_fsm_compiler 1|0 **-autosm 1|0**	Enables/disables the FSM compiler. Controls the use of FSM synthesis for state machines. The default is false (FSM Compiler disabled). Value can be 1 or true, 0 or false. When this option is true, the FSM Compiler automatically recognizes and optimizes state machines in the design. 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. However, if you turn off sequential optimizations for the design, FSM Compiler and/or the syn_state_machine directive and syn_encoding attribute are effectively disabled. See -no_sequential_opt 1|0 for more information on turning off sequential optimizations.	FSM Compiler check box, Device Panel
-syn_altera_model on | off | clearbox_only	Sets Clearbox options, which determine how instantiated and inferred Altera megafunctions are implemented for synthesis. See *Altera Models Parameters for set_option*, on page 1706 for details.	Altera Models option, Device Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-synthesis_onoff_pragma 1|0	Determines whether code between synthesis on/off directives is ignored. When enabled, the software ignores any VHDL code between synthesis_on and synthesis_off directives. It treats these third-party directives like translate_on/ off directives (see translate_off/translate_on, on page 1593 for details).	Synthesis on/off Implemented as Translate on/Off, VHDL Panel
-timequest 1|0	Determines whether to use the Altera Quartus II TimeQuest Timing Analyzer (1) or the Classic Timing Analyzer (0). This option is available only for Altera Stratix II and Stratix II GX devices. **Note:** The Quartus II TimeQuest Timing Analyzer is the default for Altera Stratix III, Stratix IV, Stratix IV GT, Arria GX, and Arria II GX devices.	Use TimeQuest Timing Analyzer, Device Panel
-top_module *name*	Specifies the top-level module. If the top-level entity does not use the default work library to compile the VHDL files, you must specify the library file where the top-level entity can be found. To do this, the top-level entity name must be preceded by the VHDL library followed by the dot (.).	Top-level Entity/Module, VHDL Panel or Verilog Panel
-update_models_cp 1|0	*Available only in Synplify Pro and Synplify Premier* Determines whether (1) or not (0) changes inside a compile point can cause the compile point (or top-level) containing it to change accordingly.	Update Compile Point Timing Data, Device Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-use_fsm_explorer 1|0	*Available only in Synplify Pro and Synplify Premier* Enables/disables the FSM Explorer.	FSM Explorer, Device Panel
-use_vivado 1 | 0	*Xilinx 7 Series (Virtex-7, Kintex-7, and Artix-7) Devices* It is recommended that you use the Vivado Integrated Design environment (IDE) tool to run place and route for these devices. You must enable this option. Disable this option, to run Xilinx ISE place and route.	Use Vivado for Place and Route Implementation Results Panel
-verification_mode 1|0	Enables/disables formal verification to be run. The VIF file is generated during synthesis for Altera designs and the Formality SVF file is generated for Xilinx designs so that equivalence checking can be performed. For more information, see *Overview of the VIF Flow*, on page 893 or *Overview of the Formality Application*, on page 1160.	Verification Mode, Device Panel
-vlog_std v2001 | v95 | sysv	The default Verilog standard for new projects is SystemVerilog. Turning off both options in the Verilog panel defaults to v95.	Verilog 2001, SystemVerilog, Verilog Panel
-voltage none | 0.9V | 1.1V	For Stratix III devices that support speed grade 4, you can specify a core voltage value.	Core Voltage, Device Panel

Copyright © 2013 Synopsys, Inc.

Option	Description	GUI Equivalent
-write_apr_constraint 1|0	Writes vendor-specific constraint files.	Write Vendor Constraint File, Implementation Results Panel
-write_verilog 1|0 **-write_vhdl 1|0**	Writes Verilog or VHDL mapped netlists.	Write Mapped Verilog/VHDL Netlist, Implementation Results Panel

Timing Report Parameters for set_option

The following lists the parameters for the stand-alone timing report (`ta` file).

async_clock	margin
filename	netlist
filter	output_srm
gen_output_srm	

Reporting Option	Description
-reporting_async_clock	Generates a report for paths that cross between clock groups using the stand-alone Timing Analyst.
-reporting_filename *filename*.**ta**	Specifies the standard timing report file (`ta`) generated from the stand-alone Timing Analyst.
-reporting_filter *filter options*	Generates the standard timing report based on the filter options you specify for paths, such as: • From points • Through points • To points For more information, see: • *Timing Report Generation Parameters*, on page 291. • *Combining Path Filters for the Timing Analyzer*, on page 295 • *Timing Analyzer Through Points*, on page 293. • *Specifying From, To, and Through Points*, on page 506.
-**reporting_gen_output_srm 1|0**	Specifies the new name of the output SRM File when you change the default name. If this option is set to 1, this new name is used for the output `srm` file after you run the stand-alone Timing Analyst.
-reporting_margin *value*	You can specify a slack margin to obtain a range of paths within the worst slack time for the design after you run the stand-alone Timing Analyst.
-reporting_netlist *filename*.**srm**	Specifies the associated gate-level netlist file (`srm`) generated from the stand-alone Timing Analyst.
-reporting_output_srm 1|0	Allows you to change the name of the output `srm` file. If you enable the output SRM File option, you can change this default name.

Copyright © 2013 Synopsys, Inc.

For GUI equivalent switches for these parameters, see *Timing Report Generation Parameters*, on page 291.

Altera Models Parameters for set_option

Sets Clearbox options, which determines how you want instantiated and inferred Altera megafunctions implemented for synthesis.

Syntax

set_option -syn_altera_model [**-on**] [**-off**] [**-clearbox_only**]

-on	Specifies that the synthesis software calls Clearbox functions for instantiated and inferred Altera Megafunctions. For example, the altsyncram megafunction is replaced with ram_block primitives.
-off	Specifies that the synthesis software does not call Clearbox functions for instantiated and inferred Altera Megafunctions. Therefore, the altsyncram megafunction is *not* replaced with ram_block primitives.
-clearbox_only	Specifies that the synthesis software only calls the Clearbox function for inferred Altera Megafunctions.

help for set_option

This option is useful for getting syntax help on the various implementation options used for compiling and mapping a design, especially since this list of options keeps growing.

Syntax

```
% set_option -help
```

Usage:

set_option *optionName optionValue* [**-help** [*value*]]

Where:

- *optionName*—specifies the option name.
- *optionValue*—specifies the option value.

- -help [*value*]—to get help on options. Use:
  - -help * for the list of options
  - -help *optionName* for a description of the option

Examples

To list all option commands in the Tcl window:

```
set_option -help *
```

To list all option commands beginning with the letters fi in the Tcl window:

```
% set_option -help fi*

fixgatedclocks
fixgeneratedclocks
fixsmult
```

To get help on a specific option in the Tcl window:

```
% set_option -help fixgatedclocks

0: Don't fix; 1: fix, no report; 2: fix, report exception
registers; 3: fix, report all registers
```

Use the following Tcl commands to print a description of the options:

```
% set_option -help c*
% set hl [set_option -help c*]
% puts $hl
% foreach option $hl { puts "$option:\t [set_option -help
$option]"; }
```

This example will print a list of set_option options that begin with the letter c.

Resolve Mixed Drivers Option

Use the Resolve Mixed Drivers option when mapping errors are generated for input nets with mixed drivers. You might encounter the following messages in the log file:

```
@A:BN313 | Found mixed driver on pin pin:data_out inst:dpram_lut3
of work.dpram(verilog), use option "Resolve Mixed Drivers" in
"Device" tab of "Implementation Options" to automatically resolve
this
@E:BN314 | Net "GND" in work.test(verilog) has mixed drivers
```


Copyright © 2013 Synopsys, Inc.

```
@A:BN313 | Found mixed driver on pin pin:Q[0] inst:dff1.q of
PrimLib.sdffr(prim), use option "Resolve Mixed Drivers" in
"Device" tab of "Implementation Options" to automatically resolve
this
@E:BN314) | Net "VCC" in work.test(rtl) has mixed drivers
```

Whenever a constant net (GND or VCC) and an active net are driving the same output net, enable the Resolve Mixed Drivers option so that synthesis can proceed. To set this switch:

- Check Resolve Mixed Drivers on the Device tab of the Implementation Options panel.

- Use the Tcl command, `set_option -resolve_multiple_driver 1`.

  By default this option is disabled and set to:

  `set_option -resolve_multiple_driver 0`.

When you rerun synthesis, you should now see messages like the following in the log file:

```
@W:BN312 | Resolving mixed driver on net GND, connecting output
pin:data_out inst:dpram_lut3 of work.dpram(verilog) to GND
@N:BN116) | Removing sequential instance dpram_lut3.dout of
view:PrimLib.dffe(prim) because there are no references to its
outputs
@N:BN116 | Removing sequential instance dpram_lut3.mem of
view:PrimLib.ram1(prim) because there are no references to its
outputs
```

Copyright © 2013 Synopsys, Inc.

```
@W:BN312 | Resolving mixed driver on net VCC, connecting output
pin:Q[0] inst:dff1.q of PrimLib.sdffr(prim) to VCC
@N:BN116 | Removing sequential instance dff1.q of
view:PrimLib.sdffr(prim) because there are no references to its
outputs
```

Example – Active Net and Constant GND Driving Output Net (Verilog)

```
module test(clk,data_in,data_out,radd,wradd,wr,rd);
input clk,wr,rd;
input data_in;
input [5:0]radd,wradd;
output data_out;
// component instantiation for shift register module
shrl srl_lut0 (
   .clk(clk),
   .sren(wr),
   .srin(data_in),
   .srout(data_out)
   );
// Instantiation for ram
dpram dpram_lut3 (
   .clk(clk),
   .data_in(data_in),
   .data_out(data_out),
   .radd(radd),
   .wradd(wradd),
   .wr(wr),
   .rd(rd)
   );

endmodule

module shrl (clk,sren,srin,srout);
input clk;
input sren;
input srin;
output srout;

parameter width = 32;
reg [width-1:0] sr;

always@(posedge clk)
begin
   if (sren == 1)
   begin
```


Copyright © 2013 Synopsys, Inc.

```
      sr <= {sr[width-2:0], srin};
   end
end
// Constant net driving

// the output net
assign srout = 1'b0;
endmodule

module dpram(clk,data_in,data_out,radd,wradd,wr,rd);
input clk,wr,rd;
input data_in;
input [5:0]radd,wradd;
output data_out;

reg dout;
reg [0:0]mem[63 :0];

always @ (posedge clk)
begin
   if(wr)
      mem[wradd] <= data_in;
end

always @ (posedge clk)
begin
   if(rd)
      dout <= mem[radd];
   end

assign  data_out = dout;

endmodule
```

See the following RTL and Technology views; the Technology view shows the constant net tied to the output.

Example – Active Net and Constant VCC Driving Output Net (VHDL)

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (clk,rst : in std_logic;
      sr_en : in std_logic;
      data : in std_logic;
      data_op : out std_logic );
end entity test;

architecture rtl of test is
component shrl
generic (sr_length : natural);
port (clk : in std_logic;
      sr_en : in std_logic;
      sr_ip : in std_logic;
      sr_op : out std_logic );
end component shrl;

component d_ff
port (data, clk, rst : in std_logic;
      q : out std_logic );
```


Copyright © 2013 Synopsys, Inc.

```
end component d_ff;

begin
-- instantiation of shift register
shift_register : shrl
generic map (sr_length => 64)
port map (clk => clk,
          sr_en => sr_en,
          sr_ip => data,
          sr_op => data_op );
-- instantiation of flipflop
dff1 : d_ff
port map (data => data,
          clk => clk,
          rst => rst,
          q => data_op );
end rtl;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity shrl is
generic (sr_length : natural);
port (clk : in std_logic;
      sr_en : in std_logic;
      sr_ip : in std_logic;
      sr_op : out std_logic );
end entity shrl;

architecture rtl of shrl is
signal sr_reg : std_logic_vector(sr_length-1 downto 0);
begin
   shreg_lut: process (clk)
   begin
      if rising_edge(clk) then
         if sr_en = '1' then
            sr_reg <= sr_reg(sr_length-2 downto 0) & sr_ip;
         end if;
      end if;
   end process shreg_lut;
-- Constant net driving output net
sr_op <= '1';
end architecture rtl;

library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity d_ff is
port (data, clk, rst : in std_logic;
      q : out std_logic );
end d_ff;

architecture behav of d_ff is
begin
   FF1:process (clk) begin
      if (clk'event and clk = '1') then
         if (rst = '1') then
            q <= '0';
         else q <= data;
         end if;
      end if;
   end process FF1;
end behav;
```

See the following RTL and Technology views; the Technology view shows the constant net tied to the output.

Copyright © 2013 Synopsys, Inc.

slice_primitive

The slice_primitive command defines where a primitive is logically divided by identifying groups of output signals by number, explicitly by name, or by the number of instances. You can place this command in an `.nrf` or a `.tcl` file. The slice_primitive command is available only for the Synplify Premier tool.

Syntax

slice_primitive [**-push**] [**-nl**] [**-numslices**] **{** *primitiveInstanceName* **}**
{ **{** *number* **}** | **{** *bitGroup1* **}** **{** *bitGroup2* **}** ... **{** *bitGroupN* **}** }

The following table describes the command options.

Option	Description
-push	(Optional) Creates a new level of hierarchy (primitive division is shown at the next level down in the hierarchy). This argument is equivalent to enabling Preserve the Hierarchical View on the Bit Slices panel.
-nl	(Optional) Allows multiple instances of the sliced element to be used in certain specific cases. This argument is equivalent to enabling Slice all instances of this type on the Bit Slices panel.
-numslices	(Optional) Divides the primitive into the specified number of instances.
primitiveInstanceName	Name of the output signal to slice into subelements. The RTL bus port name syntax is used for bus signals (for example, big_bus[63:0]). The output signal name is enclosed in curly braces (**{ }**).
number	The number of bits to assign to each primitive or, when the -numslices option is included, the number of sliced instances. Bits are assigned sequentially beginning with bit 0; any remaining bits are assigned to the next primitive. The *number* entry is enclosed in curly braces (**{ }**).
bitGroup1 *bitGroup2* ... *bitGroupN*	Explicitly specifies which bits are assigned to individual primitives. This format requires that you specify every bit. Each bit group entry is enclosed in curly braces (**{ }**) and is separated from the next group by a space.

Copyright © 2013 Synopsys, Inc.

When manually creating entries in an `nrf` or a `tcl` file, if the length of the slice_primitive command exceeds one line, the command can be continued on subsequent lines by including a line continuation character (\) as the last character of each line to be continued.

Copyright © 2013 Synopsys, Inc.

sxml2pxml Tcl Command

If you choose to run the Xilinx P&R tool from outside the synthesis tool, you must first run the sxml2pxml Tcl command for the standalone XML converter. This Tcl command is used in the compile point synthesis and Xilinx Partition flow to generate a Xilinx xpartition.pxml file with compile point information from the xpartition.sxml file for the specified implementation directory and design.

The syntax for the Tcl command is as follows:

sxml2pxml -design_name *moduleName* [**-idir** *directoryName*] [**-odir** *directoryName*]

-design_name *moduleName*	The name of the top-level module in the design.
-idir *directoryName*	The synthesis implementation directory containing the xpartition.sxml file. This input is optional; if missing, the current synthesis directory is assumed.
-odir *directoryName*	The P&R directory to which the xpartition.pxml file is written. This input is optional; if missing the current P&R directory is assumed.

Running XML Converter from the GUI

You can run this command from the Tcl Script window within the synthesis tool. For example:

- Run with all arguments:

```
sxml2pxml -design_name prep2_1 -idir /project/syn/rev_1
   -odir /project/pnr/par_1
```

- Run from the P&R directory:

```
sxml2pxml -design_name prep2_1 -idir /project/syn/rev_1
```

- Run from the P&R directory with `edif/ucf/sxml` files copied to this directory:

```
sxml2pxml -design_name prep2_1
```

Copyright © 2013 Synopsys, Inc.

Running XML Converter in Batch Mode

You can run this command in batch mode using the following methods:

- Run the command in batch mode using a Tcl script file. The sxml2pxml Tcl command is contained in the script file. For example:

```
synplify_pro -batch script.tcl
```

- Run the command in batch mode with the TCL command using the -tclcmd option. For example:

```
synplify_pro -batch -tclcmd "sxml2pxml -design_name value
   [-idir value -odir value]"
```

Technology-Specific Tcl Commands

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

Vendor/Family	Tcl Commands Described in...
Microsemi (Synplify Pro and Synplify tools only)	*Microsemi Tcl set_option Command Options*, on page 2056
Altera Stratix, Cyclone, HardCopy	*Stratix, Arria, and Cyclone Families*, on page 1927
Altera MAX (Synplify Pro and Synplify tools only)	*MAX Family*, on page 1935

Copyright © 2013 Synopsys, Inc.

Vendor/Family	Tcl Commands Described in...
Lattice GAL/PAL and isp (Synplify Pro and Synplify tools only)	*ispGAL, ispGDX, and ispLSI Devices*, on page 2001
Lattice Mach (Synplify Pro and Synplify tools only)	*ispMACH and MACH Devices*, on page 2011
Lattice ispXPGA devices (Synplify Pro and Synplify tools only)	*ispXPGA Devices*, on page 2004
Lattice ispXPLD5000MX and ispLSI	*ispXPLD5000MX and ispLSI5000VE Devices*, on page 2007
Lattice ORCA (Synplify Pro and Synplify tools only)	*ORCA Devices*, on page 1986
Lattice ECP/EC (Synplify Pro and Synplify tools only)	*LatticeECP/EC and Later Devices*, on page 1989
Lattice SC/SCM and XP/XP2 devices (Synplify Pro and Synplify tools only)	*LatticeSC/SCM and LatticeXP2/XP Devices*, on page 1993
Lattice MachXO and Platform (Synplify Pro and Synplify tools only)	*MachXO and Platform Devices*, on page 1997
Lattice iCE40 and iCE65 devices (Synplify Pro and Synplify tools only)	*Lattice iCE65 and iCE40 Devices*, on page 2014
Xilinx Virtex, Spartan-3	*Virtex and Spartan-II and Later Technologies*, on page 2121
Xilinx XC9500, CoolRunner, CoolRunner-II CPLDs (Synplify Pro and Synplify tools only)	*Xilinx CPLDs*, on page 2131

Copyright © 2013 Synopsys, Inc.

synhooks File Syntax

The Tcl hooks commands provide an advanced user with callbacks to customize a design flow or integrate with other products. To enable these callbacks, set the environment variable SYN_TCL_HOOKS to the location of the Tcl hooks file(synhooks.tcl), then customize this file to get the desired customization behavior. For more information on creating scripts using synhooks.tcl, see *Automating Flows with synhooks.tcl*, on page 1146.

Tcl Callback Syntax	Function
proc syn_on_set_project_template {*projectPath*} {*yourDefaultProjectSettings*}	Called when creating a new project. *projectPath* is the path name to the project being created.
proc syn_on_new_project {*projectPath*} {*yourCode*}	Called when creating a new project. *projectPath* is the path name to the project being created.
proc syn_on_open_project {*projectPath*} {*yourCode*}	Called when opening a project. *projectPath* is the path name to the project being created.
proc syn_on_close_project {*projectPath*} {*yourCode*}	Called after closing a project. *projectPath* is the path name to the project being created.
proc syn_on_start_application {*applicationName version currentDirectory*} {*yourCode*}	Called when starting the application. • *applicationName* is the name of the software. For example synplify_pro. • *version* is the name of the version of the software. For example 8.4 • *currentDirectory* is the name of the software installation directory. For example C:\synplify_pro\bin\synplify_pro.exe.
proc syn_on_exit_application {*applicationName version*} {*yourCode*}	Called when exiting the application. • *applicationName* is the name of the software. For example synplify_pro. • *version* is the name of the version of the software. For example 8.4.

Copyright © 2013 Synopsys, Inc.

Tcl Callback Syntax	Function
proc syn_on_start_run {*runName projectPath implementationName***}** **{***yourCode***}**	Called when starting a run. • *runName* is the name of the run. For example compile or synthesis. • *projectPath* is the location of the project. • *implementationName* is the name of the project implementation. For example, rev_1.
proc syn_on_end_run {*runName projectPath implementationName***}** **{***yourCode***}**	Called at the end of a run. • *runName* is the name of the run. For example, compile or synthesis. • *projectPath* is the location of the project. • *implementationName* is the name of the project implementation. For example, rev_1.
proc syn_on_press_ctrl_F8 {} **{***yourCode***}**	Called when Ctrl-F8 is pressed. See Tcl Hook Command Example next.
proc syn_on_press_ctrl_F9 {} **{***yourCode***}**	Called when Ctrl-F9 is pressed.
proc syn_on_press_ctrl_F8 {} **{***yourCode*	Called when Ctrl-F11 is pressed.

Tcl Hook Command Example

Create a modifier key (ctrl-F8) to get all the selected files from a project browser and project directory.

```
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]
}
```

Log File Commands

The Synplify Premier and Synplify Pro software supports Tcl commands that let you filter messages in the log file. The commands are:

- log_filter Tcl Command, on page 1721
- log_report Tcl Command, on page 1722

log_filter Tcl Command

This command lets you filter errors, notes, and warning messages (Synplify Premier and Synplify Pro). The GUI equivalent of this command is the Warning Filter dialog box, which you access by selecting the Warnings tab in the Tcl window and then clicking Filter. For information about using this command, see *Filtering Messages in the Message Viewer*, on page 523 in the *User Guide*.

Syntax

log_filter -field *fieldName*==*value*
log_filter -show_matches
log_filter -hide_matches
log_filter -enable
log_filter -disable
log_filter -clear

The following table shows valid *fieldName* and *value* values for the -field option:

Fieldname	Value
type	Error | Warning | Note
id	The message ID number. For example, MF138
message	The text of the message. You can use wildcards.
source_loc	The name of the HDL file that generated the message.
log_loc	The corresponding srr file (log).
time	The time the message was generated.
report	The log file section. For example, Compiler or Mapper.

Copyright © 2013 Synopsys, Inc.

Example

```
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"
```

log_report Tcl Command

This command lets you write out the results of the log_filter command to a file. For information about using this command, see *Filtering Messages in the Message Viewer*, on page 523 in the *User Guide*.

Syntax

You specify this command after the log_filter commands.

log_report -print *fileName*

Example

```
log_report -print output.txt
```

High-Reliability Tcl Commands

Altera, Xilinx

This section describes Tcl commands that are used to specify I/O redundancy in high-reliability designs.

- syn_connect, on page 1723
- syn_create_err_net, on page 1724

syn_connect

Altera, Xilinx

Specifies 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. It is used with the syn_create_err_net command and the syn_radhardlevel attribute to specify I/O redundancy in high-reliability designs.

This is the syntax:

syn_connect
-from {n: *netName***}**
-to {n: *existingNetpath* | **t:** *inputPinEMIPpath* | **p:** *topErrorport***}**

-from {i: *netName***}**	Specifies the new net name for the error flag connection used with the syn_create_err_net command. See *syn_create_err_net*, on page 1724.
-to **{n:** *existingNetpath* | **t:** *inputPinEMIPpath* | **p:** *topErrorport***}**	Specifies the path to an existing net, the input pin for the Error Monitoring IP (EMIP), or the top-level error port.

Here is a syntax example:

```
syn_connect {-from {n:dwc_err0_net} -to {n:emp[0]}}
```

Limitation

Currently, when using the -to argument with the t: | p: option, an existing net must be present that feeds into the ports.

Copyright © 2013 Synopsys, Inc.

syn_create_err_net

Altera, Xilinx

Specifies 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. It is used with the syn_connect command and the syn_radhardlevel attribute to specify I/O redundancy in high-reliability designs.

This is the syntax:

sync_create_err_net -name {*netName***}**
-inst {i: *highRelInstance***}**
-err_pipe_num {*numPipelineStages***}**
-err_clk {n: *clkSourceForPipelineStages***}**
-err_reset {n:*resetSourceForPipelineStages***}**
-err_set {n:*setSourceForPipelineStages***}**
-err_enable {n:*enableSourceForPipelineStages***}**
-err_synch {*synchValue***}**

netName	Specifies the new net name for the error flag connection.
-inst *highRelInstantiation*	Specifies the high reliability instance being monitored for the error.
-err_pipe_num {*numPipelineStages***}**	Specifies the number of pipeline stages using the retiming property. The default value is 0. **Note:** If err_pipe_num=N, then the property syn_allow_retiming=0 is set on the Nth stage of the pipeline register and syn_allow_retiming=1 is set on the first (N-1)th stage of the pipeline registers by the synthesis tool. Where N>0 and err_clk is specified.
-err_clk {n: *clkSourceforPipelineStages***}**	Specifies the hierarchical path to the input pin for the clock.
-err_reset {n:*resetSourceforPipelineStages***}**	Specifies the hierarchical path to the input pin for the reset signal.

Copyright © 2013 Synopsys, Inc.

-err_set {n:*setSourceforPipelineStages***}**	Specifies the hierarchical path to the input pin for the set signal.
-err_enable {n:*enableSourceforPipelineStages***}**	Specifies the hierarchical path to the input pin for the enable signal.
-err_synch {*synchValue***}**	Specifies the boolean value for synchronous or asynchronous set/reset. The default is TRUE.

Here is a syntax example:

```
syn_create_err_net {-name {dwc_err0_net} -inst {i:inst1}
   -err_pipe_num {2} -err_clk {n:inst1.clk}
   -err_reset {n:inst1.rst} -err_synch {TRUE}}
```


Copyright © 2013 Synopsys, Inc.

VIF Commands

The Tcl commands in the VIF file define and map the HDL and netlist files, the top module, registers, FSMs, and optimizations. The Synopsys FPGA tools use these commands and their options to generate sufficient information for the equivalence checking tools. These commands are only available with the Synplify Pro tool for certain Altera technologies. For more information, see *Overview of the VIF Flow*, on page 893.

vif_add_file

The vif_add_file command identifies the project files as original: the original HDL files and translated: the files generated after synthesis. For example, `vqm` (Altera).

Syntax

vif_add_file -original|-translated *filename*

Example

```
# RTL and technology files

vif_add_file -original -verilog ./prep2_2.v
vif_set_top_module -original -top prep2_2


vif_add_file -translated -verilog prep2_2.vm
vif_set_top_module -translated -top prep2_2
```

Copyright © 2013 Synopsys, Inc.

vif_add_library

vif_add_library specifies the path where the verification model of FPGA primitives can be found. You must specify the path to the original netlist (-original) as well as the optimized netlist (-translated) libraries.

Syntax

vif_add_library -original | **-translated** *pathName*

Example

```
# RTL and technology files

vif_add_file -original -verilog ./prep2_2.v
vif_set_top_module -original -top prep2_2
```

vif_set_constant

The vif_set_constant optimization prunes duplicate flip-flops and latches from the original source.

Syntax

vif_set_constant -original 0|**1**|**-1** *flipflopName*

-1 means the flip-flop is unused or simplified to gates.

Copyright © 2013 Synopsys, Inc.

vif_set_equiv

Register replication occurs when duplicate registers are added to the translated design to increase circuit speed.

Syntax

vif_set_equiv [**-inverted**] **-translated** *sourceReg duplicatedReg*

If -inverted is present, it means the *duplicatedReg* is a logic inversion of *sourceReg*.

Example

```
vif_set_equiv -translated
control_synchronization/fifo_flush_sync_Z
control_synchronization/fifo_flush_sync_fast_Z
```

vif_set_fixed_gated_clock

The vif_set_fixed_gated_clock command adds annotations to the VIF file that contains information about any gated clock that the synthesis tool has fixed during Synthesis. These annotations are useful in verifying designs (with gated clocks) using formal verification tool and the VIF flow.

Syntax

For internal clocks

vif_set_fixed_gated_clock -net *netName*

For input port clocks

vif_set_fixed_gated_clock -clock *clockPort*

Copyright © 2013 Synopsys, Inc.

vif_set_fsm

The vif_set_fsm command is used to define an FSM object.

Syntax

vif_set_fsm -fsm *fsmID*

vif_set_fsmreg

The vif_set_fsmreg command defines FSM register mapping.

Syntax

vif_set_fsmreg -original|-translated -fsm *fsmID fsmReg*

vif_set_map_point

Undefined modules in the HDL code are black boxed and not renamed in the translated netlist. Thus the mapping names for these black boxes in both -original and -translated in the `.vif` file are the same.

Syntax

vif_set_map_point -map register | **latch** | **blackbox** [**-inverted**] **-original** *reg1* **-translated** *reg2*

reg1 in HDL design corresponds to *reg2* in the netlist, if -inverted is present, the logic of *reg2* is inverted from *reg1*.

Example

```
# Map points
vif_set_map_point -register -original fifo/fifo_control/waddr[7] -
translated fifo/fifo_control/waddr_out_Z[7]
vif_set_map_point -register -original fifo/fifo_control/waddr[6] -
translated fifo/fifo_control/waddr_out_Z[6]
```


Copyright © 2013 Synopsys, Inc.

vif_set_merge

Register merging occurs when a redundant register is removed in the original design.

Syntax

vif_set_merge -original [**-fsmopt**] [**-inverted**] *survived deleted*

The -fsmopt option specifies that one or more of the specified registers may not exist in the original design, but may have been introduced after FSM optimizations.

The -inverted option specifies that the deleted register can be derived from survived register by inverting it.

Example

```
vif_set_merge -original camera_wishbone_if/wbm_stb_o
camera_wishbone_if/wbm_cyc_o
```

```
vif_set_merge -original -fsmopt camera_wishbone_if/cur_state[1]
camera_wishbone_if/wbm_cti_o[2]
```

vif_set_port_dir

The Synplify Pro tool may change port directions during synthesis, for example from INOUT to OUT. These port direction changes are specified in the .vif file.

vif_set_port_dir

vif_set_port_dir -original OUT|IN|INOUT *name*

Example

```
# Port directions
vif_set_port_dir –original OUT cpu_rst_l
```

vif_set_result_file

vif_set_result_file sets the log file name.

Syntax

vif_set_result_file *projectName*.**vlf**

Example

```
# Set logfile options
vif_set_result_file prep2_2.vlf
```

vif_set_state_map

vif_set_state_map defines the state encoding map.

Syntax

vif_set_state_map -fsm *fsmID* **-original "***original_code_as_binary_string***"**
-translated "*new_code_as_binary_string***"**

Example

```
# Read FSM encoding
vif_set_fsm -fsm fsm_0
vif_set_fsmreg -original -fsm fsm_0
camera_wishbone_if/cur_state[3:0]
vif_set_fsmreg -translated -fsm fsm_0
camera_wishbone_if/cur_state_Z[1:0]
vif_set_state_map -fsm fsm_0 -original "0001" -translated "00"
vif_set_state_map -fsm fsm_0 -original "0010" -translated "01"
vif_set_state_map -fsm fsm_0 -original "0100" -translated "10"
vif_set_state_map -fsm fsm_0 -original "1000" -translated "11"
```

The above format specifies the original and new state registers. It also specifies the mapping between states. Each FSM is specified using a similar set of commands as shown above.

Copyright © 2013 Synopsys, Inc.

vif_set_top_module

vif_set_top_module defines the top module. You must define the top module for both the original netlist (-original) and the post-synthesis netlist (-translated) file.

Syntax

vif_set_top_module -original|-translated *name*

Example

```
# RTL and technology files

vif_add_file -original -verilog ./prep2_2.v
vif_set_top_module -original -top prep2_2


vif_add_file -translated -verilog prep2_2.vm
vif_set_top_module -translated -top prep2_2
```

vif_set_transparent

Syntax

vif_set_transparent -original 0|1|-1 *latchName*

-1 means the latch is unused or simplified to gates.

Netlist Editing Commands

Netlist editing commands are only recognized when included in a Tcl file applied through the Nfilter prototyping options.

connect_net	Connects a net to one or more pins or ports.
copy_view	Duplicates a view of the current view.
create_cell	Creates an empty view.
create_cgm	Inserts an instance of a clock generator module.
create_instance	Creates one or more instances in the current view.
create_net	Creates new nets in the current view.
create_port	Creates ports of the specified direction in the current view.
create_process_hierarchy	Creates an additional level of hierarchy based on blocks in the source RTL design file
define_current_view	Sets the current view.
disconnect_connector	Disconnects pins and ports from their connected nets.
disconnect_net	Disconnects pins and ports from the specified net.
get_net	Returns the net name of a connector.
insert_buffer	Inserts buffers on specified ports and pins.
load_database	Reads in an IP block.
optimize_netlist	Removes redundant logic from the netlist.
pop_feedthroughs	Eliminates I/O pins associated with signals that pass through a design block.
propagate_constants	Eliminates the I/O pin count associated with tying a logic input signal high or low.
remove_buffer	Deletes the buffers specified.
remove_instance	Deletes the specified instances from the current view.
remove_net	Deletes the specified nets from the current view.

Copyright © 2013 Synopsys, Inc.

remove_port	Deletes the specified ports from the current view.
reset_generate	Generates the clock synchronization control signal and inserts it into the reset chain.
set_property	Sets properties on objects.
swap_instance	Replaces the view of instances.
tie_net	Ties the nets from a list of nets to a constant value.
tie_pin	Ties connectors to a constant value.

Command Listing

The supported Tcl commands are listed alphabetically in the following subsections.

connect_net

Netlist editing command that connects a net to one or more pins or ports.

Syntax

connect_net *netName* [**-hier**] *objectList*

Arguments and Options

netName
The name of the net to be connected.

objectList
The list of pins and/or ports to be connected to the specified net.

-hier
Enables objects to be connected across hierarchies.

Description

The connect_net command connects pins and ports from *objectList* to the net specified in *netName*. The command can be used to connect objects across any hierarchy with the -hier option. Any additional ports for sub-hierarchies can be created as needed.

Objects can be single bit or multiple bits, but they must all be of the same size. Multiple-bit objects must be enclosed in curly brackets. With multi-bits objects, the connection is bit-to-bit, following the declarative order. For example, connect_net foo[1:0] mem1.addr[6:7] connects pin mem1.addr[6] to net foo[1] and connects pin mem1.addr[7] to net foo[0].

An error message is generated for any of the following conditions:

- Net does not exist
- Port or pin does not exist
- Port or pin already connected to net
- Ports and pins not in same level of hierarchy
- Net, ports, and pins not the same size

Examples

```
connect_net fast_cpm_clock clk_in u2.clk_in
connect_net {busA[15:0]} {dataA[15:0]}
```


Copyright © 2013 Synopsys, Inc.

copy_view

Netlist editing command that duplicates a view of the current view.

Syntax

copy_view *currentView newView* [**-bbox**]

Arguments and Options

currentView
Name of the view to be duplicated.

newView
Name of new duplicated view.

-bbox
Treats new duplicated view as a black box.

Description

The copy_view command makes a duplicate view of the current view with a name of *newView*. When the optional -bbox argument is included, *newView* is copied as a black box with only the interconnect in place and without any internal logic. The copy_view command is commonly used to "uniquify" shared views.

Examples

When devices U1 and U2 use the same view, the following sequence can be used to create a unique view for U2:

```
copy_view v:origview v:origview1
swap_instance U2 v:origview1
```

See Also

- *define_current_view*, on page 1743

create_cell

Netlist editing command that creates an empty view.

Syntax

create_cell [*cellName*]

Arguments and Options

cellName
The name of the view you want to create.

Description

The create_cell command creates an empty view. Add content to the view using the create_port, create_instance, and create_net commands. An optional *cellName* argument can be specified to overwrite the view name contained in the .srs file to resolve conflicts with view names.

An error message is generated when the name of the view (from *cellName*) conflicts with the name of an existing view.

Example

```
create_cell SKRAM
```

See Also

- *create_instance*, on page 1739
- *create_net*, on page 1740
- *create_port*, on page 1741

Copyright © 2013 Synopsys, Inc.

create_cgm

Netlist editing command that inserts an instance of a clock generator module (CGM).

Syntax

create_cgm -inst *cgmInstanceName* **-view** *cgmViewName* [**-warnonly**]

Arguments and Options

-inst
Specifies the instance name of the clock-generator module to be inserted.

-view
Specifies the view on the top-level netlist where the clock generator instance is to be inserted.

-warnonly
An optional argument that, when present, causes a warning to be generated when the ports on the module do not match the corresponding nets (a port-net mismatch normally results in an error).

Description

The create_cgm command inserts an instance of a clock generator module in the specified top-level netlist view and connects the module ports to the corresponding nets; an existing, driven clock net is disconnected.

Examples

```
create_cgm -inst u_cgm1 -view clkgen -warnonly
```

Copyright © 2013 Synopsys, Inc.

create_instance

Netlist editing command that creates one or more instances in the current view.

Syntax

create_instance *instanceList referenceName*

Arguments and Options

instanceList
A list of instance names separated by spaces.

referenceName
The type of view

Description

The create_instance command creates one or more instances in the current view. Hierarchical names allow instance creation below the current level. In the command line, *instanceList* specifies a list of instance names separated by spaces, and *referenceName* specifies the type of view. The view type is either library.cell for a library primitive, or simply view for a netlist created from an .srs file with the create_cell command.

Note that an error message is generated when an instance name (from *instanceList*) conflicts with the name of an existing instance or when *referenceName* does not exist.

Example

```
create_instance
```


Copyright © 2013 Synopsys, Inc.

create_net

Netlist editing command that creates new nets in the current view.

Syntax

create_net *netList*

Arguments and Options

netList
A space-separated list of nets to be added to the netlist.

Description

The create_net command creates new nets in the current view. Multiple net entries in the *netList* are separated by spaces, and both single- and multiple-bit nets can be specified (multiple-bit nets must be enclosed in curly brackets). Hierarchical names allow the creation of nets below the current hierarchy level.

Note that an error message is generated when a net name (from *netList*) conflicts with the name of an existing net.

Examples

```
create_net fast_cpm_clock
```

```
create_net {mem1.addr[7:0]}
```

See Also

- *create_port*, on page 1741

Copyright © 2013 Synopsys, Inc.

create_port

Netlist editing command that creates ports of the specified direction in the current view.

Syntax

create_port *portList* [**-direction** {**in**|**out**|**inout**}]

Arguments and Options

portList
A space-separated list of ports to be added to the netlist.

-direction {**in**|**out**|**inout**}
The direction of the port to be created.

Description

The create_port command creates ports of the specified direction in the current view. Multiple port entries in the *portList* are separated by spaces, and both single- and multiple-bit ports can be specified (multiple-bit ports must be enclosed in curly brackets). When no direction is specified, input ports are created. Note that an error message is generated when a port name (from *portList*) conflicts with the name of an existing port.

Examples

```
create_port fast_cpm_clock -direction out
```

```
create_port {dataA[7:0]} {dataB[3:0]} -direction inout
```

See Also

- *create_net*, on page 1740
- *create_instance*, on page 1739

Copyright © 2013 Synopsys, Inc.

create_process_hierarchy

Netlist editing command that creates an additional level of hierarchy based on blocks in the source RTL design file.

Syntax

create_process_hierarchy

Arguments and Options

None

Description

The create_process_hierarchy command creates an additional level of hierarchy based on the always (Verilog) or process (VHDL) blocks in the source RTL design file. This feature allows large modules or entities that cannot be partitioned into a single FPGA to be broken up into a set of logical submodules that can be more easily partitioned among the available FPGAs.

When using the create_process_hierarchy command, be sure to uncheck the Enable prototyping tools check box on the GCC & Prototyping Tools tab of the Implementation Options dialog box to allow the command to be effective during netlist editing.

Examples

```
create_process_hierarchy
```

See Also

- *optimize_netlist*, on page 1752
- *pop_feedthroughs*, on page 1753
- *propagate_constants*, on page 1754

Copyright © 2013 Synopsys, Inc.

define_current_view

Netlist editing command that sets the current view.

Syntax

define_current_view *viewName*

define_current_view [get_attribute "view" {i:*u1.u2***}]**

Arguments and Options

viewName

Description

The define_current_view command sets the current view to *viewName*. Because a design can be incrementally uniquified during editing operations, the actual machine-generated view name for a multiply-instantiated module may not be known – use the get_attribute command argument to retrieve the view name from the instance name.

Copyright © 2013 Synopsys, Inc.

disconnect_connector

Netlist editing command that disconnects pins and ports from their connected nets.

Syntax

disconnect_connector *objectList*

Arguments and Options

objectList
A list of the ports or pins to be disconnected.

Description

The disconnect_connector command disconnects pins and ports in *objectList* from the nets where they are connected. Objects can be single bit or multiple bits (multiple-bit nets must be enclosed in curly brackets). The command can be used to disconnect objects that are in different hierarchies.

Note that an error message is generated for any of the following conditions:

- Net does not exist
- Port or pin does not exist
- Port or pin already disconnected from net
- Ports and pins not in same level of hierarchy
- Net, ports, and pins not the same size

Examples

The following example disconnects pins mem2.addr[1:0] and mem1.addr[7:0] from their connected nets.

```
disconnect_connector {mem2.addr[1:0]} {mem1.addr[7:0]}
```

See Also

- *disconnect_net*, on page 1745

disconnect_net

Netlist editing command that disconnects pins and ports from the specified net.

Syntax

disconnect_net *netName objectList* **| -all**

Arguments and Options

netName
The name of the net.

objectList
The list of individual pins and ports to be disconnected from *netName*.

-all
Disconnects all pins and ports from *netName*.

Description

The disconnet_net command disconnects pins and ports in *objectList* from the net specified by *netName*. The command can be used to disconnect objects that are in different hierarchies. Objects can be single bit or multiple bits, but they must all be of the same size. With multi-bits objects, the disconnection is bit-to-bit, following the declarative order. For example, disconnect_net foo[1:0] mem1.addr[6:7] disconnects pin mem1.addr[6] from net foo[1] and disconnects pin mem1.addr[7] from net foo[0].

Note that an error message is generated for any of the following conditions:

- Net does not exist
- Port or pin does not exist
- Port or pin already disconnected from net
- Ports and pins not in same level of hierarchy
- Net, ports, and pins not the same size

Copyright © 2013 Synopsys, Inc.

Examples

```
disconnect_net fast_cpm_clock clk_in u2.clk_in
disconnect_net {busA[15:0]} {dataA[15:0]}
```

See Also

- *disconnect_connector*, on page 1744

get_net

Returns the net name of a connector during netlist editing.

Syntax

get_net *connectorName*

Arguments and Options

connectorName
The name of the connector.

Description

The get_net command returns the net name of a connector. The net must be a single-bit net, and the connector must be an instance pin or a netlist bit port. An error message is generated if the connector is not connected to any net.

Example

The following command assigns the net name of pin I1 of instance inst1 to the variable net.

```
set net [get_net t:inst1.I1]
```


Copyright © 2013 Synopsys, Inc.

insert_buffer

Netlist editing command that inserts buffers on specified ports and pins.

Syntax

insert_buffer [-inverter_pair] *objectList bufferType*

Arguments and Options

objectList
List of the ports and pins to buffer.

bufferType
The type of buffer to insert. Common buffer types include:

– IBUF

– OBUF

– BUFG

– BUF

-inverter_pair
Specifies two inverters to buffer the ports or pins.

Description

Inserts specified buffer types on the ports and pins in *objectList*. A buffer is defined as a view with either a single input or one or more outputs that function as either input or !input. Use the -inverter_pair option to insert two inverters. A target technology library must be loaded before specifying any insert_buffer commands.

An error message is generated for any of the following conditions:

- Each pin or port is not connected to a net
- Bidirectional pin/port *bufferType* does not exist
- *bufferType* is not a buffer, as specified above
- Without -inverter_pair, *bufferType* does not have non-inverting outputs
- With -inverter_pair, *bufferType* does not have inverting outputs

Copyright © 2013 Synopsys, Inc.

Example

```
load_library $LIB/xilinx/xilinx.syn #loads target technology
library
insert_buffer {ua.q[7]} BUF #inserts a buffer
insert_buffer -inverter_pair {ua.q[3]} INV #inserts an inverter
pair
```


Copyright © 2013 Synopsys, Inc.

load_database

During netlist editing, reads in an IP block.

Syntax

load_database {*netlist*.**srl** |*netlist*.**srs**} [**-base_name** *string* | **-auto_rename**]

Arguments and Options

netlist.**srl** |*netlist*.**srs**
: The name of the `srl` or `srs` netlist file containing the IP block.

-base_name
: An optional argument that prepends the specified base-name string and an underscore to each instance in the IP block.

-auto_rename
: An optional argument that automatically renames modules in an IP block to resolve name conflicts with the remainder of the design.

Description

Reads an IP block. Instance names can conflict with instance names in the core design. For example, an instance named state_1 can appear independently in both an IP block and in the core design. Use the -basename or -auto_rename argument to resolve this type of naming conflict.

- **-base_name** – prepends the specified base-name string and an underscore to each instance in the IP block. For example, if the base name is bist and the IP block contains two modules, one module named aaa and the other named bbb, the modules are renamed bist_aaa and bist_bbb.
- **-auto_rename** – automatically renames modules in an IP block to resolve name conflicts with the remainder of the design. Instance names are not changed, and module names are retrieved by a get_prop command using the following syntax:

  get_prop -prop *view instanceName*

Examples

```
load_database ./tmp/add_select.srl -base_name nle
```

Copyright © 2013 Synopsys, Inc.

See Also

- *open_design*, on page 1659

Copyright © 2013 Synopsys, Inc.

optimize_netlist

Netlist editing command that removes redundant logic from the netlist.

Syntax

optimize_netlist

Arguments and Options

None

Description

The optimize_netlist command removes redundant logic from the netlist.

When using the optimize_netlist command, be sure to uncheck the Enable prototyping tools check box on the GCC & Prototyping Tools tab of the Implementation Options dialog box to allow the command to be effective during netlist editing.

Examples

```
optimize_netlist
```

See Also

- *define_collection*, on page 1801
- *pop_feedthroughs*, on page 1753
- *propagate_constants*, on page 1754

pop_feedthroughs

Netlist editing command that eliminates I/O pins associated with signals that pass through a design block.

Syntax

pop_feedthroughs

Arguments and Options

None

Description

The pop_feedthroughs command eliminates I/O pins associated with signals that pass through a design block without driving any logic or that are unnecessarily routed through a block

When using the pop_feedthroughs command, be sure to uncheck the Enable prototyping tools check box on the GCC & Prototyping Tools tab of the Implementation Options dialog box to allow the command to be effective during netlist editing.

Examples

```
pop_feedthroughs
```

See Also

- *define_collection*, on page 1801
- *optimize_netlist*, on page 1752
- *propagate_constants*, on page 1754

Copyright © 2013 Synopsys, Inc.

propagate_constants

Netlist editing command that eliminates the I/O pin count associated with tying a logic input signal high or low.

Syntax

propagate_constants [**-effort high**]

Arguments and Options

-effort high
Sets the optimization level to include flip-flops and muxes.

Description

The propagate_constants command eliminates the I/O pin count associated with tying a logic input signal high or low by making the connection internal to the instance rather than through the instance interface. When the -effort high option is included, DFFRE and DFFSE flip-flops are optimized away when their input pins are tied to constants which would result in a constant output. Similarly, 2-to-1 muxes are optimized away when both inputs are tied to the same constant value.

When using the propagate_constants command, be sure to uncheck the Enable prototyping tools check box on the GCC & Prototyping Tools tab of the Implementation Options dialog box to allow the command to be effective during netlist editing.

Examples

```
propagate_constants -effort high
```

See Also

- *define_collection*, on page 1801
- *optimize_netlist*, on page 1752
- *pop_feedthroughs*, on page 1753

Copyright © 2013 Synopsys, Inc.

remove_buffer

Netlist editing command that deletes the specified buffers.

Syntax

remove_buffer *instanceList*

Arguments and Options

instanceList

Description

The remove_buffer command deletes the buffers specified in *instanceList*. A buffer is defined by the insert_buffer command. To remove an inverter pair, the command must list both instances.

An error message is generated for any of the following conditions:

- Each instance is not a buffer as defined in the insert_buffer command
- The input pin of an instance is not connected to a net
- No outputs pins of an instance are connected to a net
- More than one outputs pin of an instance is connected to a net
- Inverting buffers are not cascaded with single fanout between them

Copyright © 2013 Synopsys, Inc.

remove_instance

Netlist editing command that deletes the specified instances from the current view.

Syntax

remove_instance *instanceList* | **-all**

Arguments and Options

instanceList
List of the instances to be deleted.

-all
Delete all instances from the current view.

Description

The remove_instance command deletes the specified instances from the current view. Hierarchical names allow deletion of instances below the current level. In the command line, *instanceList* specifies a list of instance names, separated by spaces, to be deleted. The -all argument deletes all instances from the current view.

Note that an error message is generated when an instance name (from *instanceList*) does not exist.

Copyright © 2013 Synopsys, Inc.

remove_net

Netlist editing command that deletes the specified nets from the current view.

Syntax

remove_net *netList* **| -all**

Arguments and Options

netList
List of the nets to be deleted.

-all
Delete all nets from the current view.

Description

The remove_net command deletes the specified nets from the current view. Nets can be single bit or multiple bits (multiple-bit nets must be enclosed in curly brackets). Hierarchical names allow deletion below the current level.

An error message is generated when a net name (from *netList*) does not exist.

Examples

```
remove_net fast_cpm_clock
remove_net {busA[15:0]}
```


Copyright © 2013 Synopsys, Inc.

remove_port

Netlist editing command that deletes the specified ports from the current view.

Syntax

remove_port *portList* **| -all**

Arguments and Options

portList
List of the ports to be deleted.

-all
Delete all ports from the current view.

Description

The remove_port command deletes the specified ports from the current view. Multiple port entries in the *portList* are separated by spaces, and both single- and multiple-bit ports can be specified (multiple-bit ports must be enclosed in curly brackets).

Note that an error message is generated when a port name (from *portList*) does not exist.

Examples

```
remove_port fast_cpm_clock {dataA[7:0]}
```

reset_generate

Generates the clock synchronization control signal and inserts it into the reset chain.

Syntax

reset_generate

Arguments and Options

None

Description

The reset_generate command generates the clock synchronization control signal and inserts it into the reset chain. Specifically, the command:

- Assigns the sync_in/sync_out signals for clock synchronization to the *designName*_hstdm.prt file.
- Assigns reset signals for clock synchronization and/or HSTDM that are output to the *designName*_hstdm.prt file.
- Performs clock synchronization and reset consistency and error checking.

The results of reset_generate command execution are reported in the *designName*_haps.srr log file.

Examples

```
reset_generate
```


Copyright © 2013 Synopsys, Inc.

set_property

Sets properties on objects during netlist editing.

Syntax

set_property -type {**integer**| **string**} *object property value*

Arguments and Options

-type {**integer**| **string**}
The property type; the default is string

object

property
The name of the property assigned to the object.

value
The value of the assigned property.

Description

The set_property command sets properties on objects during netlist editing. In the command syntax, the property type is either integer or string (the default is string).

Examples

```
set_property -type string portA direction inout
```

Copyright © 2013 Synopsys, Inc.

swap_instance

Replaces the view of instances during netlist editing.

Syntax

swap_instance *instanceList viewName*

Arguments and Options

instanceList

viewName

Description

The swap_instance command replaces the view of instances specified in *instanceList* with the view specified by *viewName*. The interface (pin count, pin names, pin directions) of the new view must be identical to the original view.

Copyright © 2013 Synopsys, Inc.

tie_net

Ties the nets from the list of nets to a constant value during netlist editing.

Syntax

tie_net *netsList* **"***value***"**

Arguments and Options

netsList

List of nets; the *netList* argument can have one of the forms:

netName or **n:***netName* – denotes a net

instName.*netName* or **n:***instName*.*netName* – denotes a net within the netlist of an instance

"*value***"**

The constant value; the *value* argument can be any of the following:

0 – changes the net driver to GND

1 – changes the net driver to VCC

- (dash) – keeps the net driver unchanged

x or **X** – disconnects the net driver

Note that it is not necessary to quote *value* and that no spaces are allowed within the value argument string.

Description

The tie_pin command ties the nets from the list of nets to a constant value. The arguments can be scalar or vector names, but the widths of all arguments must match. The command assigns the same value to all connectors in the list. If the nets are vectors, the assignment is done in bit-wise manner.

Copyright © 2013 Synopsys, Inc.

Examples

The command

```
tie_net {n:KONNECT6[0]} "x"
```

disconnects the net driver from bit 0 of KONNECT6.

The commands

```
tie_net {n:KONNECT6[0:3]} "x-01"
tie_net {n:KONNECT7[5:0]} "1-x100"
```

are equivalent to the following individual tie_net commands:

```
tie_net {n:KONNECT6[0]} "x"  (disconnect bit 0)
tie_net {n:KONNECT6[1]} "-"  (leave bit 1 unchanged)
tie_net {n:KONNECT6[2]} "0"  (set bit 2 to GND)
tie_net {n:KONNECT6[3]} "1"  (set bit 3 to VCC)

tie_net {n:KONNECT7[5]} "1"  (set bit 5 to VCC)
tie_net {n:KONNECT7[4]} "-"  (leave bit 4 unchanged)
tie_net {n:KONNECT7[3]} "x"  (disconnect bit 3)
tie_net {n:KONNECT7[2]} "1"  (set bit 2 to VCC)
tie_net {n:KONNECT7[1]} "0"  (set bit 1 to GND)
tie_net {n:KONNECT7[0]} "0"  (set bit 0 to GND)
```

See Also

tie_pin, on page 1764

Copyright © 2013 Synopsys, Inc.

tie_pin

Ties connectors to a constant value during netlist editing.

Syntax

tie_pin *connectorList* **"***value***"**

Arguments and Options

connectorList

List of connectors; the *connectorList* argument can have one of the following forms:

instName.*pinName* or **t:***instName*.*pinName* – denotes an instance pin

p:*portName* – denotes a top-level port

p:*instName*.*portName* – denotes a port of the internal netlist of an instance

"*value***"**

The constant value; the *value* argument can be any of the following:

0 – changes the net driver to GND

1 – changes the net driver to VCC

- (dash) – keeps the net driver unchanged

x or **X** – disconnects the net driver

Note that it is not necessary to quote *value* and that no spaces are allowed within the value argument string.

Description

The tie_pin command ties the connectors from the list of connectors to a constant value. The arguments can be scalar or vector names, but the widths of all arguments must match. The command assigns the same value to all connectors in the list. If the connectors are vectors, the assignment is done in bit-wise manner.

Copyright © 2013 Synopsys, Inc.

Examples

The command

```
tie_pin {t:U6.ALARM_HRS[0]} "1"
```

ties pin ALARM_HRS[0] of instance U6 to constant value 1 (VCC).

The commands

```
tie_pin {t:U6.ALARM_HRS[0:3]} "x-01"
tie_pin {t:U6.TIME_MINS[5:0]} "1-x100"
```

are equivalent to the following individual tie_pin commands:

```
tie_pin {t:U6.ALARM_HRS[0]} "x"  (disconnect pin 0)
tie_pin {t:U6.ALARM_HRS[1]} "-"  (keep current connection of pin 1)
tie_pin {t:U6.ALARM_HRS[2]} "0"  (tie pin 2 to constant value 0)
tie_pin {t:U6.ALARM_HRS[3]} "1"  (tie pin 3 to constant value 1)

tie_pin {t:U6.TIME_MINS[5]} "1"  (tie pin 5 to constant value 1)
tie_pin {t:U6.TIME_MINS[4]} "-"  (keep current connection of pin 4)
tie_pin {t:U6.TIME_MINS[3]} "x"  (disconnect pin 3)
tie_pin {t:U6.TIME_MINS[2]} "1"  (tie pin 2 to constant value 1)
tie_pin {t:U6.TIME_MINS[1]} "0"  (tie pin 1 to constant value 0)
tie_pin {t:U6.TIME_MINS[0]} "0"  (tie pin 0 to constant value 0)
```

See Also

tie_net, on page 1762

Copyright © 2013 Synopsys, Inc.

Tcl Script Examples

This section provides the following examples of Tcl scripts:

- Using Target Technologies, on page 1766
- Different Clock Frequency Goals, on page 1767
- Setting Options and Timing Constraints, on page 1768
- Bottom-up Synthesis, on page 1770

Using Target Technologies

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

# Open a new Project.
   project -new

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

# Add a Verilog file to the source file list.
   add_file -verilog "D:/test/simpletest/prep2_2.v"

# Create a new Tcl variable, called $try_these, used to synthesize
# the design using different target technologies.

   set try_these {

         Virtex4 Virtex5 Virtex6 # 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
   }
```

Different Clock Frequency Goals

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

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


   project -load "design.prj"

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

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

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

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

# Run synthesis.
   project -run

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


Copyright © 2013 Synopsys, Inc.

Setting Options and Timing Constraints

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

# Open a new Project
   project -new

# Set the target technology, part number, package, and speed grade options.
   set_option -technology VIRTEX2
   set_option -part XC2V40
   set_option -package CS144
   set_option -speed_grade -6

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

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

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

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

# Set the design frequency.
   set_option -frequency 30.0

# Save the existing Project to a file. The default synthesis Result File
# is named "<project_name>.<ext>". To name the synthesis Result File
# something other than "design.xnf", use project -result_file "<name>.xnf"
   project -save "design.prj"

# Synthesize the existing Project
   project -run

# Open an RTL View
   open_file -rtl_view

# Open a Technology View
   open_file -technology_view
```

```
# ---------------------------------------------------
# This constraint file, "design.fdc," is read by "test3.tcl"
# with the add_file -constraint "design.fdc" command. Constraint files
# are for timing constraints and synthesis attributes.
# ---------------------------------------------------
# Timing Constraints:
# ---------------------------------------------------
# The default design frequency goal is 30.0 MHz for four clocks. Except
# that clk_fast needs to run at 66.0 MHz. Override the 30.0 MHz default
# for clk_fast.
   define_clock {clk_fast} -freq 66.0

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

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

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

# ----------------------------------------------------
# Xilinx-specific Attribute Constraint:
# ----------------------------------------------------
# Assign a location for scalar port "sel". These VHDL object names are
# case sensitive.
   define_attribute {sel} xc_loc "P139"

# Assign a pad location to all bits of a bus.
   define_attribute {b[7:0]} xc_loc "P14, P12, P11, P5, P21,
   P18, P16, P15"

# Assign a fast output type to the pad.
   define_attribute {a[5]} xc_fast 1

# Use a regular buffer, not a clock buffer for clock "clk_slow". Save the
# clock buffers for the fast clocks
   define_attribute {clk_slow} syn_noclockbuf 1

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

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


Copyright © 2013 Synopsys, Inc.

Bottom-up Synthesis

```
# Bottom-up synthesis of a large design.

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

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

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

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

# Set the top level options
   set_option -technology Virtex5
   set_option -part XC5VLX20T
   set_option -speed_grade -1
   set_option -frequency 50.0
   set_option -symbolic_fsm_compiler true

# Set the output file information
   project -result_file top_level.edf
   project -log_file top_level.srr

# Save the Project to file
   project -save top_level.prj

# Run the Project
   project -run

# Open the top level RTL and Technology Views
   open_file -rtl_view
   open_file -technology_view

# ---------------------------------------------------
# This file, "statemach.tcl," is read by "bottom_up.tcl," (the
# bottom up Tcl script) with the command "source statemach.tcl."
# The other .tcl scripts are similar.
# ---------------------------------------------------
```

```
# Open a new Project for "statemach"
   project -new

# Add the VHDL file for this logic block.
   add_file -vhdl statemach.vhd

# Add the constraint file for this logic block.
   add_file -constraint statemach.fdc

# Set the other options for "statemach".
   set_option -technology Virtex5
   set_option -part XC5VLX20T
   set_option -speed_grade -1
   set_option -frequency 50.0
   set_option -symbolic_fsm_compiler true

# Set the Project outputs
   project -result_file statemach.edf
   project -log_file statemach.srr

# Save this Project
   project -save statemach.prj

# Run this Project
   project -run

# ---------------------------------------------
# This file (statemach.fdc) is the constraint file read by
# "statemach.tcl," with the command add_file -constraint statemach.fdc.
# This constraint file is specific to this logic block " statemach "
# ---------------------------------------------
# Timing Constraints:
# ---------------------------------------------
   define_input_delay -default -100
   define_output_delay -default -100
   define_input_delay RESET -10

# ---------------------------------------------
# Altera-Specific Attributes:
# ---------------------------------------------
   define_attribute {inst1.sqrt8} altera_implement_in_eab 1
```


Copyright © 2013 Synopsys, Inc.

CHAPTER 18

Tcl Find, Expand, and Collections

The FPGA synthesis software includes powerful search functionality in the Tcl find and expand commands. Objects located by these commands can be grouped into collections and manipulated. The following sections describe the commands and collections in detail:

- Tcl find Command, on page 1774
- Tcl Find -filter Command, on page 1785
- Tcl expand Command, on page 1791
- Collection Commands, on page 1794
- Object Query Commands, on page 1803
- Synopsys Standard Collection Commands, on page 1815

Copyright © 2013 Synopsys, Inc.

Tcl find Command

The Tcl find command identifies design objects based on specified criteria. Use this command to locate multiple objects with a common characteristic. If you want to locate objects that share connectivity, use the expand command instead of the find command (*Tcl expand Command*, on page 1791).

You can specify the find command from the SCOPE environment or enter it as a Tcl command. This command operates on the RTL database.

You can define objects identified by find as a group or *collection*, and operate on all the objects in the collection at the same time. To do this, you embed the find command as part of a collection creation or manipulation command to do this in a single step. The combination of find and collection commands provides you with very powerful functionality to operate on and manipulate multiple design objects simultaneously.

The table summarizes where to find detailed information:

For...	See...
Command syntax	*Tcl Find Syntax*, on page 1775
Syntax details: object types, expressions, case sensitivity, and special characters	*Tcl Find Command Object Types*, on page 1778 *Regular Expressions, Wildcards, and Special Characters*, on page 1778 *Tcl Find Command Case Sensitivity*, on page 1780
Examples of find syntax	Demos and Examples button, accessible from the tool UI *Tcl Find Syntax Examples*, on page 1781
Filtering find searches by property	*Tcl Find -filter Command*, on page 1785 *Find Filter Properties*, on page 1786 *Refining Tcl Find Results with -filter*, on page 154 in the *User Guide*.
Using find search patterns and using find in collections	*Finding Objects with Tcl find and expand*, on page 152 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Tcl Find Syntax

find [-*objectType*] [*pattern*]
[**-in $***collectionName* | *listName*]
[**-hier**] [**-hsc** *separator*]
[**-regexp**] [**-nocase**] [**-exact**]
[**-print**]
[**-namespace techview** | **netlist**]
[**-flat**]
[**-leaf**]
[**-rtl** | **-tech**
[**-filter** *expression*]

If used, the -filter option must be specified as the last argument in the command. Descriptions of each command option are listed alphabetically in the following table.

Argument	Description
-objectType pattern	Specifies the type of object to be found: view, inst, port, pin, net or seq. The object type must be preceded by the appropriate prefix, as described in *Tcl Find Command Object Types*, on page 1778. *pattern* specifies the search pattern to be matched, and can include the * and ? wildcard characters.
-exact	Disables simple pattern matching. Use it to search for objects that contain the * and ? wildcard characters. You cannot use this argument with -nocase or -regexp. See *Regular Expressions, Wildcards, and Special Characters*, on page 1778 for additional information.
-filter *expression*	Refines the results of find further, by filtering the results by the specified object property. For details about the syntax, refer to *Tcl Find -filter Command*, on page 1785. If you use the -filter option, it must be specified as the last argument to the find command.
-flat	Allows wildcard * to match the hierarchy separator.

Copyright © 2013 Synopsys, Inc.

Argument	Description
-hier	Searches for the pattern from every level of hierarchy, instead of just the top level. By default, the search occurs from the top-level hierarchy for the given pattern; this option allows you to search for the pattern from every level of hierarchy. When -hier is not specified, the search is limited to the current view and wildcards do not match the hierarchy delimiter character. You can still traverse downward through the hierarchy by adding the delimiter in the pattern. Thus an asterisk (*) matches any object at the current level, but *.* matches any object one level below the current view. For more about wildcard characters, see *Regular Expressions, Wildcards, and Special Characters*, on page 1778.
-hsc *separator*	Specifies the hierarchy delimiter character. The default is the dot (.). The dot can be ambiguous if it is used both as a delimiter and as a normal character. For example, block1.u1 could mean the instance u1 in block1, or a record named block1.u1. Use the -hsc *separator* option to specify an unambiguous character as the hierarchy delimiter. For example, find -hsc "@" [block1@u1] finds the hierarchical instance in the block, while find -hsc "@" [block1.u1] finds the record. See *Tcl Syntax Guidelines for Constraint Files*, on page 123 for more information.
-in $*collectionName*|*listName*	Restricts the search to the specified list or collection.
-leaf	Returns only non-hierarchical instances.
-nocase	Ignores case when matching patterns. The default is to take case into account (-case). You cannot use the -nocase argument with -exact or -regexp. See *Tcl Find Command Case Sensitivity*, on page 1780 for more information.
-namespace techview | netlist	Determines the database to search for the find operation. • techview searches the mapped (srm) database. This is the default. • netlist searches the output netlist. This option is not available for an RTL view. To select a database view (`srs`, `srp`, or `srm`) with find in batch mode, use open_design (*open_design*, on page 1659).

Argument	Description

-print	Prints the first 20 results. For a full list of objects found, use c_print or c_list.
	If you specify this command from an HDL Analyst view, the results are printed to the Tcl window; if you specify it in the constraint file, the results are printed to the log file, at the beginning of the Mapper section.
	Reported object names have prefixes that identify the object type, and double quotes around each name to allow for spaces in the names. For example:
	`"i:reg1"`
	`"i:\weird_name[foo$]"`
	`"i:reg2"`
	`<<found 233 objects. Displaying first 20 objects. Use c_print or c_list for all. >>`

-regexp	Treats the pattern as a regular expression instead of a simple wildcard pattern. It also uses the == and != filter operators to compare regular expressions, rather than simple wildcard patterns. You cannot use this argument with -nocase or -exact. If you do not specify -regexp, there are only two special characters that are used as wildcards: * and ?. See *Regular Expressions, Wildcards, and Special Characters,* on page 1778 for details about regular expressions.
-rtl | -tech	Uses the most recently activated RTL or Technology view. If none are available, it opens a new view. The RTL view is the default.

Copyright © 2013 Synopsys, Inc.

Tcl Find Command Object Types

You can specify the following types of objects:

Object	Prefix	Example	Synopsys
view (Design)	v:	v:work.cpu.rtl is the master cell of the cpu entity, rtl architecture, compiled in the VHDL work library.	lib_cell
inst (Instance)	i:	Default object type. i:core.i_cpu.reg1 points to the reg1 instance inside i_cpu.	cell
port	p:	p:data_in[3] points to bit 3 of the primary data_in port. work.cpu.rt1|p:rst is the hierarchical rst port in the cpu view. This eventually points to all instances of cpu.	port
pin	t:	t:core.i_cpu.rst points to the hierarchical rst pin of instance i_cpu.	pin
net	n:	n:core.i_cpu.rst points to the rst net driven in i_cpu.	net
seq (Seqential instance)	i:	i:core.i_cpu.reg[7:0]	cell

Regular Expressions, Wildcards, and Special Characters

The Tcl find command significantly differs from a simple Tcl search. A simple Tcl search does not treat any character, except for the backslash (\), as a special character, so * matches everything in a string. The Tcl find command uses various regular expressions and special characters, as shown in the following table.

Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern, and the backslash to escape a single character.

Syntax	Matches...
Meta Characters: Used to match certain conditions in a string	
^	At the beginning of the string
$	At the end of the string. Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern.

Syntax	Matches...
.	Any character. If you want to use the dot (.) as a hierarchy delimiter, you must escape it with a backslash (\), because it has a special meaning in regular expressions.
k	Interprets and matches the specified non-alphanumeric character as an ordinary, non-reserved character (where *k* is the non-alphanumeric character). For example, \$ matches a dollar symbol, not the character in its reserved sense of matching the end of a string. Similarly \.d in a.b.c\.d.e indicates that c.d must be interpreted as part of the instance name, not as a hierarchy separator.
c	The specified non-alphanumeric character (where *c* is the non-alphanumeric character) when it is used in an escape sequence.
|	Equivalent to an OR.
Character Class: A list of characters to match	
[*list*]	Any single character from the *list*. For example, [abc] matches a lower-case a, b, or c. To specify a range of characters in the list, use a dash. For example, [A-Za-z] matches any alphabetical character. Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern.
[^*list*]	Characters not in the list. You can specify a range, as described above. For example, [^0-9] matches any non-numeric character.
\	Used as a prefix to escape special characters like the following: ^ $ \ . | ( ) [ ] { } ? + *
Escape Sequences: Shortcuts for common character classes	
\d	A digit between 0 and 9
\D	A non-numeric character
\s	A white space character
\S	A non-white space character
\w	A word character; i.e., alphanumeric characters or underscores
\W	A non-word character

Copyright © 2013 Synopsys, Inc.

Syntax	Matches...
Quantifiers: Number of times to match the preceding pattern	
*	A sequence of 0 or more matches If you do not specify -hier, the search is restricted to the current view only. To traverse downward through the hierarchy, either use the -hier argument or specify the hierarchical levels to be searched by adding the hierarchical delimiter to the pattern. For example, *.* matches objects one level below the current view.
+	A sequence of 1 or more matches
?	A sequence of 0 or 1 matches
{*N*}	A sequence of exactly *N* matches Use curly brackets to interpret special characters as ordinary characters within a pattern.
{*N*,}	A sequence of *N* or more matches
{*N*,*P*}	A sequence of *N* through *P* matches (*P* included); *N*<=*P*

Tcl Find Command Case Sensitivity

Case sensitivity depends on the rules of the language used to specify the object. If the object was generated in VHDL, it is case-insensitive; if it was generated in Verilog, it is case-sensitive. In mixed-language designs, the case-sensitivity rules for the parent object prevail, even when another language is used to define the lower-level object.

Tcl Find Syntax Examples

The following are examples of find syntax:

Example	Description
find {a*}	Finds any object in the current view that starts with a
find {a*} -hier -nocase	Finds any object that starts with a or A
find -net {*synp*} -hier	Finds any net the contains synp
find -seq * -filter {@clock==myclk}	Finds any register in the current view that is clocked by myclk
find -flat -seq {U1.*}	Finds all sequential elements at any hierarchical level under U1 (* matches hierarchy separator)
find -hier -flat -inst {i:A.B.C.*} -filter @view==ram*	Finds all RAM instances starting from a submodule and all lower hierarchical levels from A downwards
find -hier-seq {*} -filter @clock_enable==ena	Finds all registers enabled by the ena signal.
find -hier-seq {*} -filter @slack <{-0.0}	Finds all sequential elements with negative slack.
find -hier-seq {*} -filter {@clock ==clk1}	Finds all sequential elements within the clk1 clock domain
find -hier-net {*} -filter {@fanout >20}	Finds high fanout nets that drive more than 20.destinations
find -hier-seq * -in $all_inst_coll	Finds sequential elements inside the all_inst_coll collection
find -net -regexp {[a-b].*}	Finds all nets in hierarchy a and b. This means {n:a.*} and {n:b.*}

Use the {} characters to protect patterns that contain [] from Tcl evaluation. For example, use the following command to find instance reg[4]:

```
find -inst {reg[4]}
```


Copyright © 2013 Synopsys, Inc.

Find Equivalents for UCF Group Commands

The following table shows UCF commands (simple Tcl) and their equivalent find commands:

UCF	Constraint Using find Command
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]

Example: Custom Report Showing Paths with Negative Slack

Use the following commands:

```
open_design implementation_a/top.srm
set find_negslack[find -hier –seq –inst {*} -filter @slack <
   {-0.0}]
c_print -prop slack -prop view $find_negslack -file negslack.txt
```

The result of running these commands is a report called negslack.txt:

```
Object Name                          slack  view
{i:CPU_A_SOC.CPU.DATAPATH.GBR[0]} -3.264  "FDE"
{i:CPU_A_SOC.CPU.DATAPATH.GBR[1]} -3.158  "FDE"
{i:CPU_A_SOC.CPU.DATAPATH.GBR[2]} -3.091  "FDE"
```

Example: Custom Report for Negative Slack FFs in a Clock Domain

The following procedure steps through the commands used to find all negative slack flip-flops with a given clock domain:

1. Create a collection that contains all sequential elements with negative slack:

   ```
   set negFF  [find -tech -hier -seq {*} -filter @slack < {-0.0}]
   ```

2. Create a collection of all sequential elements within the clk clock domain

   ```
   set clk1FF  find -hier -seq * -filter {@clock==clk1}
   ```

3. Isolate the common elements in the two collections:

```
set clk1Slack [c_intersect $negFF $clk1FF]
```

4. Generate a report using the c_print command:

```
c_print [find -hier -net * -filter @fanout>=2]
   {n:ack1_tmp}
   {n:ack2_tmp}
   ...
   {n:blk_xfer_cntrl_inst.lfsr_data[20:14]}
   {n:blk_xfer_cntrl_inst.lfsr_inst.blk_size[6:0]}
   {n:blk_xfer_cntrl_inst.lfsr_inst.clk_c}
   ...
```

Custom Fanout Report Example

The following command generates a fanout report:

```
% c_print -prop fanout [find -hier -net * -filter @fanout>=2]
```

This is an example of the report generated by the command:

```
Object Name                                          fanout
{n:ack1_tmp}                                              3
{n:ack2_tmp}                                              4
...
{n:blk_xfer_cntrl_inst.lfsr_data[14]}                     3
{n:blk_xfer_cntrl_inst.lfsr_data[15]}                     3
{n:blk_xfer_cntrl_inst.lfsr_data[16]}                     2
...
```

You can add additional information to the report, by specifying more properties. For example:

```
% c_print -prop fanout [find -hier -net * -filter @fanout>=2] -prop
pins
```

This command generates a report like the one shown below:

```
Object Name    Fanout      Pins
{n:ack1_tmp}    3          "t:word_xfer_cntrl_inst.ack1_tmp
                           t:word_xfer_inst.ack1_tmp"
{n:ack2_tmp}    4          "t:blk_xfer_cntrl_inst.ack2_tmp
                           t:blk_xfer_inst.ack2_tmp"
{n:adr_o_axb_1} 2          "t:blk_xfer_inst.adr_o_axb_1
                           t:adr_o_cry_1_0.S t:adr_o_s_1.LI"
{n:adr_o_axb_2} 2          "t:blk_xfer_inst.adr_o_axb_2
                           t:adr_o_cry_2_0.S t:adr_o_s_2.LI"
{n:adr_o_axb_3} 2          "t:blk_xfer_inst.adr_o_axb_3
```


Copyright © 2013 Synopsys, Inc.

```
                                t:adr_o_cry_3_0.S t:adr_o_s_3.LI"
{n:adr_o_axb_4} 2               "t:blk_xfer_inst.adr_o_axb_4
                                t:adr_o_cry_4_0.S t:adr_o_s_4.LI"
{n:adr_o_axb_5} 2               "t:blk_xfer_inst.adr_o_axb_5
                                t:adr_o_cry_5_0.S t:adr_o_s_5.LI"
{n:adr_o_axb_6} 2               "t:blk_xfer_inst.adr_o_axb_6
                                t:adr_o_cry_6_0.S t:adr_o_s_6.LI"
...
```

To save the report as a file, use a command like this one:

```
c_print -prop fanout [find -hier -net * -filter @fanout>=2]
   -prop pins –file prop.txt
```

Tcl Find -filter Command

The Tcl find command includes the optional -filter option, which provides a powerful way to further refine the results of the find command and filter objects based on properties. See the following for details about the find -filter command:

- Find -filter Syntax, on page 1785
- Find Filter Properties, on page 1786
- Find Filter Examples, on page 1789

For the Tcl find command syntax, see

Find -filter Syntax

find *pattern other_args* **-filter [!]{@***property_name operator value***}**

!	Optional character to specify the negative. Include the ! character if you are checking for the absence of a property; leave it out if you are checking for the presence of a property.
@property_name	Property name to use for filtering. The name must be prefixed with the @ character. For example, if clock is the property name, specify {@clock==myclk}.
operator	Evaluates and determines the property value used for the filter expression. You can use the following operators: • Relational operators: =, <, >, ==, >=, <= • Logical operators: &&, ||, !
value	Property value for the property in the filter expression, when the property has a value. The value can either be an object name such as myclk in {@clock==myclk}, or a value, such as 60 in {@fanout>=60}.

When specified, the -filter option must be the last option specified for the find command.

Copyright © 2013 Synopsys, Inc.

Find Filter Properties

The object properties are based on the design or constraint, and are used to qualify searches and build collections. To generate these properties, open Project->Implementation Options->Device and enable the Annotated Properties for Analyst check box. The properties display in the Tcl window when the RTL or Technology view is active. Some properties are only available in a certain view. The tool creates `.sap` and `.tap` files (design and timing properties, respectively) in the project folder.

The table below lists the common filter object properties. It does not include some vendor-specific properties. Use the table as a guide to filter the properties you want. Here is how to read the columns:

Property Name	Property Value	HDL View	Comment
Common Properties			
type	view | port | net | instance | pin]	All	
View Properties			
compile_point	locked	Tech	
is_black_box	1	All	
is_verilog	0 | 1	All	
is_vhdl	0 | 1	All	
syn_hier	remove | flatten | soft | firm | hard	Tech	
Port Properties			
direction	input | output | inout	All	
fanout	*value*	All	Total fanout (integer)
side	bottom | top| left | right	Tech	Xilinx only
Instance Properties			
area	*area_value*	Tech	

Copyright © 2013 Synopsys, Inc.

Property Name	Property Value	HDL View	Comment
arrival_time	*value*	Tech	Corresponds to worst slack
async_reset	**n:***netName*	All	
async_set	**n:***netName*	All	
clock	*clockName*	All	Could be a list if there are multiple clocks
clock_edge	rise | fall | high | low	All	Could be a list if there are multiple clocks
clock_enable	**n:***netName*	All	Highest branch name in the hierarchy, and closest to the driver
compile_point	locked	Tech	Automatically inherited from its view
hier_rtl_name	*hierInstanceName*	All	
inout_pin_count	value	All	
input_pin_count	value	All	
inst_of	viewName	All	
is_black_box	1 (Property added)	All	Automatically inherited from its view
is_hierarchical	1 (Property added)	All	
is_sequential	1 (Property added)	All	
is_combinational	1 (Property added)	All	
is_pad	1 (Property added)	All	
is_tristate	1 (Property added)	All	
is_keepbuf	1 (Property added)	All	
is_clock_gating	1 (Property added)	All	
is_vhdl	0 | 1	All	Automatically inherited from its view

Copyright © 2013 Synopsys, Inc.

Property Name	Property Value	HDL View	Comment
is_verilog	0 | 1	All	Automatically inherited from its view
kind	*primitive* For example: inv | and |dff | mux | statemachine | ...)	All	Tech view contains vendor-specific primitives
location	(*x*, *y*)	Tech	Format can differ
name	*instanceName*	All	
orientation	N | S | E | W	Tech	
output_pin_count	*value*	All	
pin_count	*value*	All	
placement_type	unplaced | placed	All	
rtl_name	*nonhierInstanceName*	All	
slack	*value*	Tech	Worst slack of all arcs
slow	1	Tech	
sync_reset	**n:***netName*	All	
sync_set	**n:***netName*	All	
syn_hier	remove | flatten | soft | firm | hard	Tech	Automatically inherited from its view
view	*viewName*	All	
Pin Properties			
arrival_time	*timingValue*	Tech	
clock	*clockName*	All	Could be a list if there are multiple clocks
clock_edge	rise | fall | high | low	All	Could be a list if there are multiple clocks
direction	input | output | inout	All	
fanout	*value*	All	Total fanout (integer)

Copyright © 2013 Synopsys, Inc.

Property Name	Property Value	HDL View	Comment
is_clock	0 | 1	All	
is_gated_clock	0 | 1	All	Set in addition to is_clock
slack	*value*	Tech	
Net Properties			
clock	*clockName*	All	Could be a list if there are multiple clocks
is_clock	0 | 1	All	
is_gated_clock	0 | 1	All	Set in addition to is_clock
fanout	*value*	All	Total fanout (integer)

Find Filter Examples

The following examples show how find -filter is used to check for the presence or absence of a property, with the ! character indicating a negative check:

`c_print [find -hier -view{ *} -filter (@is_black_box)]`	Finds all objects that are black boxes.
`c_print [find -hier -view {*} -filter (!@is_black_box)]`	Finds all objects that are not black boxes

The following are additional positive check examples:

`find * -filter @fanout>8`
(Finds all ports, pins, and nets from the top level with a fanout greater than 8)

`find * -hier -filter @view!=andv || @view!=orv`
(Finds all instances other than andv and orv in the design)

`find -hier -inst * -filter @inst_of==statemachine`
(Finds all instances of statemachine throughout the hierarchy)

`find -hier -inst * -filter @kind==statemachine`
(Finds all instances of statemachine throughout the hierarchy)

Copyright © 2013 Synopsys, Inc.

```
find -hier -inst {*reg*} -filter @clock==CLK
```
(Finds all instances throughout the hierarchy with the name reg and that are clocked by CLK)

```
find -hier -net {*} -filter (@fanout > 4)
```
(Finds all nets throughout the hierarchy that have a fanout greater than 4)

This is another example of a negative check:

```
find -inst *big* -filter (!@is_black_box && @pin_count > 10
```
(Finds all instances from the top level that have the name big, are not black boxes, and have more than 10 pins)

You can also specify Boolean expressions on multiple properties:

```
find * -filter @pin_count>8 && @slack<0
```
(Finds all instances from the top level that have more than 8 pins and with negative slack)

Tcl expand Command

The expand command identifies objects based on their connectivity, by expanding forward from a given starting point. For more information, see *Using the Tcl expand Command to Define Collections*, on page 157 of the *User Guide*.

Tcl expand Syntax

The syntax for the expand command is as follows:

expand [-*objectType*] [**-from** *object*] [**-thru** *object*] [**-to** *object*] [**-level** *integer*] [**-hier**] [**-leaf**] [**-seq**] [**-print**]

Argument	Description
-from *object*	Specifies a list or collection of ports, instances, pins, or nets for expansion forward from all the pins listed. Instances and input pins are automatically expanded to all output pins of the instances. Nets are expanded to all output pins connected to the net. If you do not specify this argument, backward propagation stops at all sequential elements.
-hier	Modifies the range of any expansion to any level below the current view. The default for the current view is the top level and is defined with the define_current_design command as in the compile-point flow.
-leaf	Returns only non-hierarchical instances.
-level *integer*	Limits the expansion to N logic levels of propagation. You cannot specify more than one -from, -thru, or -to point when using this option.
-*objectType*	Optionally specifies the type of object to be returned by the expansion. If you do not specify an *objectType*, all objects are returned. The object type is one of the following: • **-instance** returns all instances between the expansion points. This is the default. • **-pin** returns all instance pins between the expansion points. • **-net** returns all nets between the expansion points. • **-port** returns all top-level ports between the expansion points.

Copyright © 2013 Synopsys, Inc.

Argument	Description

-print	Evaluates the expand function and prints the first 20 results. If you use this command from HDL Analyst, these results are printed to the Tcl window; for constraint file commands, the results are printed to the log file at the start of the Mapper section.
	For a full list of objects found, you must use c_print or c_list. Reported object names have prefixes that identify the object type. There are double quotes around each name to allow for spaces in the names. For example:
	`"i:reg1"`
	`"i:reg2"`
	`"i:\weird_name[foo$]"`
	`"i:reg3"`
	`<<found 233 objects. Displaying first 20 objects. Use c_print or c_list for all. >>`

-seq	Modifies the range of any expansion to include only sequential elements. By default, the expand command returns all object types. If you want just sequential instances, make sure to define the *object_type* with the -inst argument, so that you limit the command to just instances.
-thru *object*	Specifies a list or collection of instances, pins, or nets for expansion forward or backward from all listed output pins and input pins respectively. Instances are automatically expanded to all input/output pins of the instances. Nets are expanded to all input/output pins connected to the net. You can have multiple -thru lists for product of sum (POS) operations.
-to *object*	Specifies a list or collection of ports, instances, pins, or nets for expansion backward from all the pins listed. Instances and output pins are automatically expanded to all input pins of the instances. Nets are expanded to all input pins connected to the net. If you do not specify this argument, forward propagation stops at all sequential elements.

Copyright © 2013 Synopsys, Inc.

Tcl expand Syntax Examples

Example	Description
expand -hier -from {i:reg1} -to {i:reg2}	Expands the cone of logic between two registers. Includes hierarchical instances below the current view.
expand -inst -from {i:reg1}	Expands the cone of logic from one register. Does not include instances below the current view.
expand -inst -hier -to {i:reg1}	Expands the cone of logic to one register. Includes hierarchical instances below the current view.
expand -pin -from {t:i_and2.z} -level 1	Finds all pins driven by the specified pin. Does not include pins below the current view.
expand -hier -to {t:i_and2.a} -level 1	Finds all instances driving an instance. Includes hierarchical instances below the current view.
expand -hier -from {n:cen}	Finds all elements in the transitive fanout of a clock enable net, across hierarchy.
expand -hier -from {n:cen} -level 1	Finds all elements directly connected to a clock enable net, across hierarchy.
expand -hier -thru {n:cen}	Finds all elements in the transitive fanout and transitive fanin of a clock enable net, across hierarchy.

Copyright © 2013 Synopsys, Inc.

Collection Commands

Synplify Pro and Synplify Premier only

A collection is a group of objects. Grouping objects lets you operate on multiple group members at once; for example you can apply the same constraint to all the objects in a collection. You can do this from both the SCOPE editor (see *Collections*, on page 480) or in a Tcl file.

The following table lists the commands for creating, copying, evaluating, traversing, and filtering collections, and subsequent sections describe the collections, except for find and expand, in alphabetical order. For information on using collections, see *Using Collections*, on page 161 in the *User Guide*.

Command	Description
Creation	
define_collection	Creates a collection from a list
set modules	Creates a collection
set modules_copy $*modules*	Copies a collection
Creation from Objects Identified by Embedded Commands	
Tcl find Command	Does a targeted search and finds objects. Embedding the find command in a collection creation command first finds the objects, and then creates a collection out of the identified group of objects.
Tcl expand Command	Identifies related objects by expanding from a selected point. Embedding the expand command in a collection creation command first finds the objects, and then creates a collection out of the identified group of objects.
Operators for Comparison and Analysis	
c_diff	Identifies differences between lists or collections
c_intersect	Identifies objects common to a list and a collection
c_symdiff	Identifies objects that belong exclusively to only one list or collection
c_union	Concatenates a list to a collection

Copyright © 2013 Synopsys, Inc.

Command	Description
Operators for Evaluation and Statistics	
c_info	Prints statistics for a collection
c_list	Converts a collection to a Tcl list for evaluation
c_print	Displays collections or properties for evaluation

Tcl Collection Commands that manipulate two or more collections

c_union

c_diff

c_symdiff

c_intersect

c_diff

Identifies differences by comparing collections, or a list and a collection. For this command to work, the design must be open in the GUI.

Syntax

c_diff $*collection1* **$***collection2* |*list* **-print**

This command also includes a -print option to display the result.

Copyright © 2013 Synopsys, Inc.

Examples

The following examples combines the set with the c_diff command to create a new collection that contains the results of the c_diff command. The first example compares two collections and puts the results in diffCollection:

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

The next example compares collection1 to a Tcl list containing one object, and puts the results in a collection called diffCollection:

```
%set collection1 {i:reg1 i:reg2}
%set diffCollection [c_diff $collection1 {i:reg1}]
%c_print $diffCollection
{i:reg2}
```

c_info

Returns specifics of a collection, including database name, number of objects per type, and total number of objects. You can save the results to a Tcl variable (array) using the -array *name* option.

Syntax

c_info $*mycollection* [**-array** *name*]

c_intersect

Defines common objects that are included in each of the collections or lists being compared.

Syntax

c_intersect $*collection1* **$***collection2* |*list* **-print**

This command also includes a -print option to display the result.

Example

The following example uses the set command to create a new collection that contains the results of the c_intersect command. The example compares a list to a collection (myCollection) and puts the common elements in a new collection called commonCollection:

```
%set mycollection {i:reg1 i:reg2}
%set commonCollection [c_intersect $mycollection {i:reg1 i:reg3}]
%c_print $intercollection
   {i:reg1}
```

c_list

Converts a collection to a Tcl list of objects. You can evaluate any collection with this command. If you assign the collection to a variable, you can then manipulate the list using standard Tcl list commands like lappend and lsort. Optionally, you can specify object properties to add to the resulting list with the -prop option:

```
(object prop_value ... prop_value)...
   (object prop_value ... prop_value)
```

Syntax

c_list **$***collection*|*list* [**-prop** *propertyName*]*

Example

```
$set myModules [find -view *]
%c_list $myModules
{v:top}{v:block_a}{v:block_b}

%c_list $myModules -prop is_vhdl -prop is_verilog
```

Name	is_vhdl	is_verilog
{v:top}	0	1
{v:block_a}	1	0
{v:block_b}	1	0

Copyright © 2013 Synopsys, Inc.

c_print

Displays collections or properties in column format. Object properties are printed using one or more –prop *propertyName* options.

Syntax

c_print $*collection*|*list* [**-prop** *propertyName*]* [**-file** *filename*]

To print to a file, use the -file option. The following command in a constraint file prints the whole collection to a file:

```
c_print -file foo.txt $col
```

Note that the command prints the file to the current working directory. If you have multiple projects loaded, check that the file is written to the correct location. You can use the pwd command in the Tcl window to echo the current directory and then use cd *directoryName* to change the directory as needed.

Example

```
%set modules [find -view *]
%c_print $modules
{v:top}
{v:block_a}
{v:block_b}

%c_print -prop is_vhdl -prop is_verilog $modules
Name is_vhdl is_verilog
{v:top}0 1
{v:block_a}1 0
{v:block_b}1 0
```

c_symdiff

Compares a collection to another collection or Tcl list and finds the objects that are unique, not shared between the collections or Tcl lists being compared. It is the complement of the c_intersect command (*c_intersect*, on page 1796).

Syntax

c_symdiff **$***collection1* **$***collection2* |*list* **-print**

This command also includes a -print option to display the result.

Examples

The following example uses the set command together with the c_symdiff command to compare two collections and create a new collection (symDiffCollection) that contains the results of the c_symdiff command.

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

The next example is more detailed. It compares a list to a collection (collection1) and creates a new collection called symDiffCollection from the objects that are different. In this case, reg1 is excluded from the new collection because it is common to both the list and collection1.

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

You can also use the command to compare two collections:

c_union

Adds collections, or a list to a collection, and removes any redundant instances. For this command to work, the design must be open in the GUI.

Syntax

c_union **$***collection1* **$***collection2* |*list* **-print**

The c_union command automatically removes redundant elements. This command also includes a -print option to display the result.

Copyright © 2013 Synopsys, Inc.

Examples

You can concatenate two collections into a new collection using the c_union and set commands, as shown in the following example where collection1 and collection2 are concatenated into combined_collection:

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

The following example creates a new collection called sumCollection, which is generated by adding a Tcl list with one object (reg3) to collection1, which consists of reg1 and reg2. The new collection created by c_union contains reg 1, reg2, and reg3.

```
%set collection1 [find -instance {reg?} -print]
   {i:reg1}
   {i:reg2}
%set sumcollection [c_union $collection1 {i:reg3}]
%c_list $sumcollection
   {i:reg1} {i:reg2} {i:reg3}
```

If instead you added reg2 and reg3 to collection1 with the c_union command, the command removes redundant instances (reg2), so that the new collection still consists of reg1, reg2, and reg3.

```
%set collection1 {i:reg1 i:reg2}
%set sumcollection [c_union $collection1 {i:reg2 i:reg3}]
%c_list $sumcollection
   {i:reg1} {i:reg2} {i:reg3}
```

get_prop

Returns a single property value for each member of the collection in a Tcl list.

Examples

```
get_prop -prop clock [find -seq *]

get_prop $listExpandedInst -prop rtl_name LOROM32X1inst

get_prop $listExpandedInst -prop location SLICE_X1y36

get_prop $listExpandedInst -prop bel C6LUT

get_prop $listExpandedInst -prop slack 0.678
```

Copyright © 2013 Synopsys, Inc.

define_collection

Creates a collection from any combination of single elements, Tcl lists, and collections. You get a warning message about empty collections if you define a collection with a leading asterisk and then define an attribute for it, as shown here:

```
set noretimesh [define_collection [find -hier -seq *uc_alu]]
define_attribute {$noretimesh} {syn_allow_retiming} {0}
```

To avoid the error message, remove the leading asterisk and change `*uc_alu` to `uc_alu`.

Example

```
set modules [define_collection {v:top} {v:cpu} $mycoll $mylist]
```

define_scope_collection

The define_scope_collection command combines set and define_collection to create a collection and assigns it to a variable.

```
define_scope_collection my_regs {find -hier -seq *my*}
```


Copyright © 2013 Synopsys, Inc.

set

Copies a collection to create a new collection. This command copies the collection but not the name, so the two are independent. Changes to the original collection do not affect the copied collection.

Syntax

set *collectionName collectionCriteria*

set *copyName* **$***collectionName*

collectionName	The name of the new collection.
collectionCriteria	Criteria for defining the elements to be included in the collection. Use this argument to embed other commands, like Tcl find and expand, as shown in the examples below, or other collection commands like define_collection, c_intersect, c_diff, c_union, and c_symdiff. Refer to the these commands for examples.
copyName	The name assigned to the copied collection.
$*collectionName*	Name of an existing collection to copy.

Examples

The following syntax examples illustrate how to use the set command:

- Use the set command to copy a collection:

  ```
  set my_mod_copy $my_module
  ```

- Use the set command with a variable name and an embedded find command to create a collection from the find command results:

  ```
  set my_module [find -view *]
  ```

- Use the set command with define_collection to create a collection:

  ```
  set my_module [define_collection {v:top} {v:cpu} $col_l $mylist]
  ```

  For more examples of the set command used with embedded Tcl collection commands, see the examples in *c_diff*, on page 1795, *c_intersect*, on page 1796*c_symdiff*, on page 1798, *c_union*, on page 1799, and *define_collection*, on page 1801.

Object Query Commands

The query commands are Synopsys SDC commands from the Desgin Compiler tool for creating collections of specific object types. Functionally, they are equivalent to the Tcl find and expand commands (*Tcl find Command*, on page 1774 and *Tcl expand Command*, on page 1791).

The Synopsys SDC collection commands are only intended to be used in the the FDC file to create collections of objects for constraints. This section describes the syntax for the object query commands supported in the FPGA synthesis tools. For complete documentation on these commands, refer to the Design Compiler documentation.

- all_clocks
- all_inputs
- all_outputs
- all_registers
- get_cells
- get_clocks
- get_nets
- get_pins
- get_ports

Note: Since all the query commands above are used to create Tcl collections of objects for constraints, they must be enclosed in [ ] to be applied. For example:

```
set_input_delay 0.5 [all_inputs] -clock clk
```

Object Query Commands and Tcl find and expand Commands

The Synopsys get* commands and all* commands are functionally similar to the the Tcl find and expand commands. The get* commands and all* commands are better suited to use with constraints and the fdc file, because they handle properties like @clock better than the Tcl find and expand commands. In certain cases, the fdc file does not support the find and expand commands, although you can still enter them in the Tcl window. See *Object Query Commands and Tcl find and expand Commands*, on page 1803 for examples.

Copyright © 2013 Synopsys, Inc.

Object Query and Tcl find/expand Examples

The following table lists parallel examples that compare how to use either the Tcl find/expand or the get/all commands to query design objects and set constraints.

Return the output pins of top-level registers clocked by clkb (e.g. inst1.inst2.my_reg.Q)	
all_registers	FDC Constraint: set_multicycle_path {4} -from [all_registers -no_hierarchy -output_pins -clock [get_clocks {clkb}]] set_multicycle_path {4} -from [get_pins -of_objects [get_cells * -filter {@clock == clkb}] -filter {@name == Q}]
find	Tcl Window: % define_collection [regsub -all {i:([^\s]+)} [join [c_list [find -inst * -filter @clock == clkfx]]] {t:\1.Q}]
Return all registers in the design clocked by the rising edge of clock clkfx	
all_registers	FDC Constraint: set_multicycle_path {3} -to [all_registers -cells -rise_clock [get_clocks {clkfx}]] set_multicycle_path {3} -to [get_cells -hier * -filter {@clock == clkfx && @clock_edge == rise}]
find	Tcl Window: find -hier -inst * -filter {@clock == clkfx && @clock_edge == rise}
Return clock pins of all registers clocked by the falling edge of cklfx	
all_registers	FDC Constraint: set_multicycle_path {2} -from [all_registers -clock_pins -fall_clock [get_clocks {clkfx}]] set_multicycle_path {2} -from [get_pins -of_objects [get_cells -hier * -filter {@clock == clkfx && @clock_edge == fall}] -filter {@name == C}]
find	Tcl Window: % find -hier -inst * -filter {@clock == clkfx && @clock_edge == fall}
Return the E pins of all instances of dffre cells (e.g. inst1.inst2.my_reg.E)	
get_pins	FDC Constraint: set_multicycle_path -to [get_pins -filter {@name == E} -of_objects [get_cells -hier * -filter {@inst_of == dffre}]
find	Tcl Window and FDC Constraint: % regsub -all {i:([^\s]+)} [join [c_list [find -hier -inst * -filter @inst_of == dffre]]] {t:\1.E}]

Copyright © 2013 Synopsys, Inc.

all_clocks

Returns a collection of clocks in the current design.

Syntax

This is the supported syntax for the all_clocks command:

all_clocks

This command has no arguments. All clocks must be defined in the design before using this command. To create clocks, you can use the create_clock command.

Example

The following constraint sets a multicycle path from all the starting points.

```
set_multicycle_path 3 -from [all_clocks]
```

all_inputs

Returns a collection of input or inout ports in the current design.

Syntax

This is the supported syntax for the all_inputs command:

all_inputs

Example

The following constraint sets a default input delay.

```
set_input_delay 3 [all_inputs]
```

all_outputs

Returns a collection of output or inout ports in the current design.

Copyright © 2013 Synopsys, Inc.

Syntax

This is the supported syntax for the all_outputs command:

all_outputs

Example

The following constraint sets a default output delay.

```
set_output_delay 2 [all_outputs]
```

all_registers

Returns a collection of sequential cells or pins in the current design.

Syntax

This is the supported syntax for the all_registers command:

all_registers

Example

The following constraint sets a max delay target for timing paths leading to all registers.

```
set_max_delay 10.0 -to [all_registers]
```

get_cells

Creates a collection of cells from the current design that is relative to the current instance.

Copyright © 2013 Synopsys, Inc.

Syntax

This is the supported syntax for the get_cells command:

get_cells
[**-hierarchical**]
[**-nocase**]
[-**regexp**]
[**-filter** *expression*]
[*pattern*]

Arguments

-hierarchical	Searches each level of hierarchy for cells in the design relative to the current instance. The object name at a particular level must match the patterns. For the cell `block1/adder`, a hierarchical search uses `"adder"` to find this cell name. By default, the search is *not* hierarchical.
-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (== and !=).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (== and !=)have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.
-filter *expressions*	Filters the collection with the specified expression. For each cell in the collection, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.
pattern	Creates a collection of cells whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option.

Copyright © 2013 Synopsys, Inc.

Examples

The following example creates a collection of cells that begin with o and reference an FD2 library cell.

```
get_cells "o*" -filter "@ref_name == FD2"
```

The following example creates a collection of cells connected to a collection of pins.

```
set pinsel [get_pins o*/cp]
get_cells -of_objects $pinsel
```

The following example creates a collection of cells connected to a collection of nets.

```
set netsel [get_nets tmp]
get_cells -of_objects $netsel
```

get_clocks

Creates a collection of clocks from the current design.

Syntax

This is the supported syntax for the get_clocks command:

get_clocks
[**-nocase**]
[-**regexp**]
[**-filter** *expression*]
[*pattern* | **-of_objects** *objects*]
[**-include_generated_clocks**]

Arguments

-nocase Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (== and !=).

-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (== and !=) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.
-filter *expressions*	Filters the collection with the specified expression. For each clock in the collection, the expression is evaluated based on the clock's attributes. If the expression evaluates to true, the clock is included in the result.
pattern	Creates a collection of clocks whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option.
-of_objects *objects*	Creates a collection of clocks that are defined for the given net or pin objects.
-include_generated_clocks	Creates a collection of clocks matching the search criteria and includes any clocks derived or generated from the source clocks found.

Examples

The following example creates a collection of clocks that match the wildcard pattern.

```
get_clocks {*BUF_1*derived_clock*}
```

The following example creates a collection of clocks that match the given regular expression.

```
get_clocks -regexp {.*derived_clock}
```

The following example creates a collection that includes clka and any generated or derived clocks of clka.

```
get_clocks -include_generated_clocks {clka}
```


Copyright © 2013 Synopsys, Inc.

get_nets

Creates a collection of nets from the current design.

Syntax

This is the supported syntax for the get_nets command:

get_nets
[**-hierarchical**]
[**-nocase**]
[-**regexp | -exact**]
[**-filter** *expression*]
[*pattern* | **-of_objects** *objects*]

Arguments

-hierarchical	Searches each level of hierarchy for nets in the design relative to the current instance. The object name at a particular level must match the patterns. For the net `block1/muxsel` a hierarchical search uses `muxsel` to find this net name. By default, the search is *not* hierarchical.
-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (== and !=).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (== and !=) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.
-filter *expressions*	Filters the collection with the specified expression. For any nets in the collection, the expression is evaluated based on the net's attributes. If the expression evaluates to true, the net is included in the result.

Copyright © 2013 Synopsys, Inc.

pattern	Creates a collection of nets whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The patterns and -of_objects arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.
-of_objects *objects*	Creates a collection of nets connected to the specified objects. Each object can be a pin, port, or cell.

Examples

The following example creates a collection of nets connected to a collection of pins.

```
set pinsel [get_pins {o_reg1.Q o_reg2.Q}]
get_nets -of_objects $pinsel
```

The following example creates a collection of nets connected to the E pin of any cell in the modulex_inst hierarchy.

```
get_nets {*.*} -filter {@pins == modulex_inst.*.E}
```

get_pins

Creates a collection of pins from the current design that match the specified criteria.

Syntax

This is the supported syntax for the get_pins command:

get_pins
[**-hierarchical**]
[**-nocase**]
[-**regexp** | **-exact**]
[**-filter** *expression*]
[*pattern* |**-of_objects** *objects* [**-leaf**]

Copyright © 2013 Synopsys, Inc.

Arguments

-hierarchical	Searches each level of hierarchy for pins in the design relative to the current instance. The object name at a particular level must match the patterns. For the cell `block1/adder/D[0]`, a hierarchical search uses `adder/D[0]` to find this pin name. By default, the search is *not* hierarchical.
-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (== and !=).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (== and !=) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed. The `-regexp` and `-exact` options are mutually exclusive; use only one.
-exact	Treats wildcards as plain characters, so the meanings of these wildcard are not interpreted. The `-regexp` and `-exact` options are mutually exclusive; use only one.
-filter *expressions*	Filters the collection with the specified expression. For each pin in the collection, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the cell is included in the result.
pattern	Creates a collection of pins whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The patterns and -of_objects arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.

Copyright © 2013 Synopsys, Inc.

-of_objects *objects*	Creates a collection of pins connected to the specified objects. Each object can be a cell or net. By default, the command considers only pins connected to the specified nets at the same hierarchical level. To consider only pins connected to leaf cells on the specified nets, use the `-leaf` option. You cannot use the `-hierarchical` option with the `-of_objects` option.
-leaf	Includes pins that are on leaf cells connected to the nets specified with the `-of_objects` option. The tool can cross hierarchical boundaries to find pins on leaf cells.

Examples

The following example creates a collection for all pins in the design.

```
get_pins -hier *.*
```

The following example creates a collection for pins from the top-level hierarchy that match the regular expression.

```
get_pins -regexp {.*\.ena}
```

The following example creates a collection for pins throughout the hierarchy that match the regular expression.

```
get_pins -hier - regexp {.*\.ena}
```

The following example creates a collection of hierarchical pin names for the library cell pin DQSFOUND, and for each instantiation of a library cell named PHASER_IN_PHY.

```
get_pins -filter {@name == DQSFOUND} -of_objects [get_cells -hier
* -filter {@inst_of == PHASER_IN_PHY}]
```

get_ports

Creates a collection of ports from that match the specified criteria.

Copyright © 2013 Synopsys, Inc.

Syntax

This is the supported syntax for the get_ports command:

get_ports
[**-nocase**]
[-**regexp**]
[**-filter** *expression*]
[*pattern*]

Arguments

-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (== and !=).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (== and !=) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.
-filter *expressions*	Filters the collection with the specified expression. For each port in the collection, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the port is included in the result.
pattern	Creates a collection of ports whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The patterns and -of_objects arguments are mutually exclusive, so only specify one of them. If you do not specify either argument, the command uses * (asterisk) as the default pattern.

Examples

The following example queries all input ports beginning with mode.

```
get_ports mode* -filter {@direction == input}
```

Synopsys Standard Collection Commands

There are a number of Synopsys standard SDC collection commands that can be included in the fdc file. These commands are not compatible with the define_scope_collection command.

The collection commands let you manipulate or operate on multiple design objects simultaneously by creating, copying, evaluating, iterating, and filtering collections. This section describes the syntax for the following collection commands supported in the FPGA synthesis tools; for the complete syntax for these commands, refer to the Design Compiler documentation.

- add_to_collection
- append_to_collection
- copy_collection
- foreach_in_collection
- get_object_name
- index_collection
- remove_from_collection
- sizeof_collection

Use these commands in the FDC constraint file to facilitate the shared scripting of constraint specifications between the FPGA synthesis and Design Compiler tools.

Copyright © 2013 Synopsys, Inc.

add_to_collection

Adds objects to a collection that results in a new collection. The base collection remains unchanged. Any duplicate objects in the resulting collection are automatically removed from the collection.

Syntax

This is the supported syntax for the add_to_collection command:

add_to_collection
[*collection1*]
[*objectSpec*]

Arguments

collection1	Specifies the base collection to which objects are to be added. This collection is copied to a resulting collection, where objects matching *objectSpec* are added to this results collection.
objectSpec	Specifies a list of named objects or collections to add. Depending on the base collection type (heterogeneous or homogeneous), the searches and resulting collection may differ. For more information, see *Heterogeneous Base Collection*, on page 1816 and *Homogeneous Base Collection*, on page 1817.

Description

The add_to_collection command allows you to add elements to a collection. The result is a new collection representing the objects added from the *objectSpec* list to the base collection. Objects are duplicated in the resulting collection, unless they are removed using the -unique option. If *objectSpec* is empty, then the new collection is a copy of the base collection. Depending on the base collection type (heterogeneous or homogeneous), the searches and resulting collection may differ.

Heterogeneous Base Collection

If the base collection is heterogeneous, then only collections are added to the resulting collection. All implicit elements of the *objectSpec* list are ignored.

Copyright © 2013 Synopsys, Inc.

Homogeneous Base Collection

If the base collection is homogeneous and any elements of *objectSpec* are not collections, then the command searches the design using the object class of the base collection.

When *collection1* is an empty collection, special rules apply to *objectSpec*. If *objectSpec* is not empty, at least one homogeneous collection must be in the *objectSpec* list (can be any position in the list). The first homogeneous collection in the *objectSpec* list becomes the base collection and sets the object class for the function.

Example

```
set result [get_cells{u*}]
get_object_name $result
```

```
==> {u:u1} {i:u2} {i:u3}
```

```
set result_1 [add_to_collection $result {get_cells {i:clkb_IBUFG}]
get_object_name $result_1
```

```
==> {i:u1} {i:u2} {i:u3} {i:clkb_IBUFG}
```

See Also

- append_to_collection

Copyright © 2013 Synopsys, Inc.

append_to_collection

Adds objects to the collection specified by a variable, modifying its value. Objects must be unique, since duplicate objects are not supported.

Syntax

This is the supported syntax for the append_to_collection command:

append_to_collection]
[*variableName*]
[*objectSpec*]

Arguments

variableName	Specifies a variable name. The objects matching *objectSpec* are added to the collection referenced by this variable.
objectSpec	Specifies a list of named objects or collections to add to the resulting collection.

Description

The append_to_collection command allows you to add elements to a collection. This command treats the *variableName* option as a collection, and appends all the elements of *objectSpec* to that collection. If the variable does not exist, it creates a collection with elements from the *objectSpec* as its value. So, a collection is created that was referenced initially by *variableName* or automatically if the *variableName* was not provided. However, if the variable exists but does not contain a collection, then an error is generated.

The append_to_collection command can be more efficient than the add_to_collection command (*append_to_collection*, on page 1818) when you are building a collection in a loop.

Copyright © 2013 Synopsys, Inc.

Example

```
set result [get_cells{u*}]
get_object_name $result
```

```
==> {u:u1} {i:u2} {i:u3}
```

```
append_to_collection result {get_cells {i:clkb_IBUFG}]
get_object_name $result
```

```
==> {i:u1} {i:u2} {i:u3} {i:clkb_IBUFG}
```

See Also

- add_to_collection

Copyright © 2013 Synopsys, Inc.

copy_collection

Duplicates the contents of a collection that results a new collection. The base collection remains unchanged.

Syntax

This is the supported syntax for the copy_collection command:

copy_collection
[*collection*1]

Arguments

collection1	Specifies the collection to be copied.

Description

The copy_collection command is an efficient mechanism to create a duplicate of an existing collection. It is sometimes more efficient and usually sufficient to simply have more than one variable referencing the same collection. However, whenever you want to copy the collection instead of reference it, use the copy_collection command.

Be aware that if an empty string is used for the *collection1* argument, the command returns an empty string. This means that a copy of the empty collection is an empty collection.

Example

```
set insts [define_collection {u1 u2 u3 u4}]
set result_copy [copy_collection $insts]
get_object_name $result_copy
```

```
==> {u1} {u2} {u3} {u4}
```

Copyright © 2013 Synopsys, Inc.

foreach_in_collection

Iterates on the elements of a collection.

Syntax

This is the supported syntax for the foreach_in_collection command:

foreach_in_collection
[*iterationVariable*]
[*collections*]
[*body*]

Arguments

iterationVariable	Specifies the name of the iteration variable. It is set to a collection of one object. Any argument that accepts collections as an argument can also accept the *iterationVariable*, as they are the same data type.
collections	Specifies a list of collections on which to iterate.
body	Specifies a script to execute for the iteration. If the body of the iteration is modifying the netlist, all or part of the collection involved in the iteration can be deleted. The foreach_in_collection command is safe for such operations. A message is generated that indicates the iteration ended prematurely.

Description

The foreach_in_collection command is a Design Compiler and PrimeTime command used to iterate on each element of a collection. This command requires the following arguments: an iteration variable (do not specify a list), the collection on which to iterate, and the script to apply for each iteration.

You can nest this command within other control structures, including another foreach_in_collection command.

You can include the command in an FDC file, but if you are using the Tcl window and HDL Analyst, you must use the standard Tcl foreach command instead of foreach_in_collection.

Copyright © 2013 Synopsys, Inc.

Example

The following examples show valid methods to reference a collection for this command:

```
set seqs[all_registers]
set port[all_inputs]

foreach_in_collection x [all_registers] {body}
foreach_in_collection x $ports {body}
foreach_in_collection x [list $seqs $ports] {body}
foreach_in_collection x {$seqs} {body}
foreach_in_collection x {$seqs $ports} {body}
```

get_object_name

Returns a list of names for objects in a collection.

Syntax

This is the supported syntax for the get_object_name command:

get_object_name
[*collection1*]

Arguments

collection1	Specifies the name of the collection that contains the requested objects.

Example

```
set c1[define_collection {u1 u2}]
get_object_name $c1
```

```
==> {u1} {u2}
```


Copyright © 2013 Synopsys, Inc.

index_collection

Creates a new collection that contains only the single object for the index specified in the base collection. You must provide an index to the collection.

Syntax

This is the supported syntax for the index_collection command:

index_collection
[*collection1*]
[*index*]

Arguments

collection1	Specifies the collection to be searched.
index	Specifies an index to the collection. Allowed values are integers from 0 to sizeof_collection -1.

Description

You can use the index_collection command to extract a single object from a collection. The result is a new collection that contains only this object. The range of indices can be from 0 to one less than the size of the collection. If the specified index is outside that range, an error message is generated.

Commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same, predictable order each time. Applications that support the sorting of collections allow you to impose a specific order on a collection.

If you use an empty string for the *collection1* argument, then any index to the empty collection is not valid. This results in an empty collection and generates an error message.

Be aware that all collections cannot be indexed.

Copyright © 2013 Synopsys, Inc.

Example

```
set c1[get_cells {u1 u2}]]
get_object_name [index_collection $c1 0]
```

```
==> {u1}
```

See Also

- sizeof_collection

Copyright © 2013 Synopsys, Inc.

remove_from_collection

Removes objects from a collection that results in a new collection. The base collection remains unchanged.

Syntax

This is the supported syntax for the remove_from_collection command:

remove_from_collection
[**-intersect**]
[*collection1*]
[*objectSpec*]

Arguments

-intersect	Removes objects from the base collection that are *not* found in *objectSpec*. By default, when this option is not specified, objects are removed from the base collection that are found in the *objectSpec*.
collection1	Specifies the base collection that is copied to a resulting collection, where objects matching *objectSpec* are removed from this results collection.
objectSpec	Specifies a list of named objects or collections to remove. The object class for each element in this list must be the same in the base collection. If the name matches an existing collection, that collection is used. Otherwise, objects are searched in the design using the object class for the base collection.

Description

The remove_from_collection command removes elements from a collection and creates a new collection.

When the -intersect option is not specified and there are no matches for *objectSpec*, the resulting collection is just a copy of the base collection. If everything in the *collection1* option matches the *objectSpec*, this results in an empty collection. When using the -intersect option, nothing is removed from the resulting collection.

Copyright © 2013 Synopsys, Inc.

Heterogeneous Base Collection

If the base collection is heterogeneous, then any elements of *objectSpec* that are not collections are ignored.

Homogeneous Base Collection

If the base collection is homogeneous and any elements of *objectSpec* are not collections, then the command searches the design using the object class of the base collection.

Examples

```
set c1[define_collection {u1 u2 u3}]]
set c2[define_collection {u2 u3 u4}]]
get_object_name [remove_from_collection $c1 $c2]

==> {u1}

get_object_name [remove_from_collection $c2 $c1]

==> {u4}

get_object_name [remove_from_collection -intersect $c1 $c2]

==> {u2} {u3}
```

See Also

- add_to_collection

Copyright © 2013 Synopsys, Inc.

sizeof_collection

Returns the number of objects in a collection.

Syntax

This is the supported syntax for the sizeof_collection command:

sizeof_collection
[*collection1*]

Arguments

collection1	Specifies the name of the collection for which the number of objects is requested. If no collection argument is specified, then the command returns 0.

Examples

```
set c1[define_collection {u1 u2 u3}]
sizeof_collection $c1

==> 3
```

SYNOPSYS®
Accelerating Innovation

APPENDIX A

Designing with Achronix

This chapter contains guidelines for synthesizing Achronix-based designs. It discusses the following topics:

- Achronix Components, on page 1830
- Output Files and Forward Annotation, on page 1850
- Achronix Technology Support, on page 1855
- Achronix Attribute and Directive Summary, on page 1861

Copyright © 2013 Synopsys, Inc.

Achronix Components

The following topics describe how the synthesis tools handle various Achronix components, and show you how to work with or manipulate them during synthesis to get the results you need:

- Achronix LRAM640 Inference, on page 1830
- Achronix BRAM80K Inference, on page 1834
- Achronix BMACC56 Inference, on page 1835
- Achronix ROMs, on page 1837
- RAM and ROM Initialization, on page 1837
- Achronix Register Inference, on page 1837
- Register Initialization for Achronix, on page 1838

Achronix LRAM640 Inference

The Achronix Speedster22iHD device supports Logic RAM (LRAM640) inference. When the syn_ramstyle attribute is not specified for a RAM instance, the synthesis software automatically infers the RAM based on size, availability of resources, and RAM mode. The software can map the RAM to logic RAM, block RAM, or registers based on the following conditions:

- LRAM640 is inferred if the RAM size is between 128 and 6400.
- BRAM80K is inferred if the RAM size is greater than 6400.
- Registers are inferred if the RAM size is less than 128.

For bigger RAM (for example, address width greater than 6 and/or data width greater than 10) the software can infer more than one LRAM640.

The following combinations of LRAM640 implementations are possible:

- Example 1: RAM With Read-Address Register
- Example 2: RAM With Output Register
- Example 3: RAM with both read-address register and output register
- Example 4: Async read RAM support with different read and write addresses
- Example 5: Async read RAM support with same read and write address

Copyright © 2013 Synopsys, Inc.

Example 1: RAM With Read-Address Register

When the syn_ramstyle attribute is specified as no_rw_check, the software infers LRAM640 without glue logic for memory collision resolution. Otherwise, LRAM640 is inferred with glue logic.

Example 2: RAM With Output Register

When the syn_ramstyle attribute is specified as no_rw_check, the software infers LRAM640 without glue logic for memory collision resolution. Otherwise, LRAM is not inferred and RAM is mapped to registers.

Copyright © 2013 Synopsys, Inc.

Example 3: RAM with both read-address register and output register

When the syn_ramstyle attribute is specified as no_rw_check, the software infers LRAM with output register packed and without glue logic for memory collision resolution. Otherwise, LRAM is inferred with input read-address register packed and with glue logic for memory collision resolution.

Example 4: Async read RAM support with different read and write addresses

Example 5: Async read RAM support with same read and write address

Example 6: Read Only Memory (ROM)

If the ROM size is between 128 and 6400, it is mapped to LRAM640.

Memory Initialization

The initial values specified using $readmemb and $readmemh statements are forward annotated as mem_init parameter value of LRAM640.

Copyright © 2013 Synopsys, Inc.

Limitations

Note the following limitations:

- True dual-port RAM cannot be mapped to LRAM640.
- If the read address register is initialized, it is not considered for packing in the LRAM640.
- Parameter mem_init_file is not supported. However, if initial values are given by data_file in RTL, those initial values are applied to LRAM640, using the parameter mem_init.

Achronix BRAM80K Inference

Achronix Speedster22iHD technologies support block RAM (BRAM80K) inference. BRAM80K is an 80K-bit dual-port memory.

BRAM Mapping

The BRAM80K can operate in the following four basic read and write modes supported by the tool:

- Writefirst, Latched Mode
- Writefirst, Registered Mode
- Nochange, Latched Mode
- Nochange, Registered Mode

Latched mode and registered mode both support reading data from the BRAM80K. In latched mode, the read address is registered and the stored data is latched into the output latches. In the registered mode, the output pipeline register is also packed into the BRAM80K with the read operation resulting in one additional latency cycle.

For Readfirst mode, the single-port and simple dual-port RAMs are mapped to registers. For the true dual-port RAMs the tool generates an error.

The tool maps the memory to block RAM by default. To disable the block RAM implementation or to specify the RAM implementation, the attribute syn_ramstyle can be used.

Copyright © 2013 Synopsys, Inc.

The synthesis tool supports the following BRAM80K memory configurations:

- True dual-port RAM
- Simple dual-port RAM
- Single-port RAM
- Read only memory (ROM)

BRAM80K is inferred irrespective of whether the output registers are uninitialized or initialized in both the latched and registered modes.

Achronix BMACC56 Inference

The synthesis tool infers BMACC56 for Speedster22iHD devices. The following logic structures can be mapped to BMACC56:

- Multipliers
- Mult-add – Multiplier followed by an adder
- Mult-sub – Multiplier followed by a subtractor
- Mult-acc – Multiplier accumulator
- MultAcc-sub –Multiplier accumulator with subtractor

By default, these logic structures are mapped to BMACC56 when all the inputs for the multiplier are greater than 2-bits wide. If the input of the multiplier is less than or equal to 2 bits, then the multiplier is mapped to logic. To override this default behavior, apply the syn_dspstyle attribute.

The following features are also supported:

- Packing input/output registers to BMACC56.

  The input/output registers for the multiplier, mult-add, mult-sub can be packed into the BMACC56. Register packing is implemented even when the registers cross hierarchy.
- Mapping both signed and unsigned multipliers to BMACC56.
- Multipliers with a size greater than 28X28 (signed) and 27X27 (unsigned) are fractured and mapped to multiple BMACC56.

Copyright © 2013 Synopsys, Inc.

Mult-add/Mult-sub Inference

The synthesis tool infers BMACC56 for Mult-add and Mult-sub by using the cascade chain. Prerequisites for inferring BMACC56 with the Mult-add/Mult-sub structures include the following:

- The multiplier input is not greater than 28X28 (signed) and 27X27 (unsigned).
- Signed multipliers have the proper sign extension.
- All the multiplier output bits feed the adder input.
- The multiplier output is not registered.

Multiplier-Accumulator Inference

The Mult-acc/MultAcc-sub structures use the adder feedback loop inside the BMACC56 block and there are no external feedback paths. Prerequisites for inferring BMACC56 with Mult-acc/MultAcc-sub structures include the following:

- The multiplier input is not greater than 28X28 (signed) and 27X27 (unsigned).
- Signed multipliers have the proper sign extension.
- All the multiplier output bits feed the adder input.
- The adder output is registered and provides feedback to the adder input for accumulation

The loadable Mult-acc can also be mapped to BMACC56. The BMACC56 is configured so that it can load the multiplier output.

Example: RTL for Loadable Accumulator Mapped to BMACC56

```
module test (clk, ld, ina, inb, dout);
parameter widtha = 28;
parameter widthb = 28;
parameter width_out = 56;

input clk, ld;
input signed [widtha-1:0] ina;
input signed [widthb-1:0] inb;
output reg signed [width_out-1:0] dout;
```

```
wire signed [width_out-1:0] mult1;

assign mult1 = ina * inb;

always@(posedge clk)
   if(ld)
      dout <= mult1;
   else
      dout <= mult1 + dout;
endmodule
```

The load signal can be registered or non-registered.

Achronix ROMs

The Achronix block RAM can also be configured as a single-port or dual-port ROM. You can map ROM to block RAM with the syn_romstyle attribute.

RAM and ROM Initialization

To initialize RAM and ROM, see *Initializing Values in RTL*, on page 1837.

Initializing Values in RTL

Initial values specified in the RTL for memories can be mapped to startup values on the FPGA. Achronix devices support power on startup values for memories. See Initializing RAMs, on page 394 in the *User Guide* for step-by-step procedures. You can initialize values in Verilog and VHDL. See:

- Initializing RAMs in Verilog, on page 394
- Initializing RAMs in VHDL, on page 395

Achronix Register Inference

The synthesis tool supports the following register inference features for the Achronix Speedster22iHD devices:

- Register initialization that supports all Achronix register primitives.

Copyright © 2013 Synopsys, Inc.

- Inference of registers that have both clock enable and async/sync control signals.

See the table below for the types of register primitives supported:

Asynchronous	Synchronous
dff, dffe[a]	
dffr[b]	dffc
dffer[c] (sr_assertion=unclocked)	dffer (sr_assertion=clocked)
dffs[d]	dffp
dffes[e] (sr_assertion=unclocked)	dffes (sr_assertion=clocked)
	dffec, dffep[f]

a. dffe is a clock enabled flip-flop.
b. dffs/dffr is a flip-flop with asynchronous set/reset signal.
c. dffer with a clock sr_assertion value is an enabled flip-flop with asynchronous reset.
d. dffs/dffr is a flip-flop with asynchronous set/reset signal.
e. dffes with a clock sr_assertion value is an enabled flip-flop with synchronous set.
f. dffec/dffep is a flip-flop with synchronous set/reset control signal.

Register Initialization for Achronix

The Synplify Pro synthesis tool supports register initialization for the Achronix Speedster22iHD devices. Initial values for registers can be specified, so that initial values specified in the RTL can be passed to the output netlist whenever there are conflicts with the default initial state of the register. Initial values are passed to the output netlist using the following primitives:

- DFF[N]
- DFF[N]E
- DFF[N][E]R
- DFF[N][E]S
- DFF[N][E]C
- DFF[N][E]P

When the reset value and the initial value of the registers are the same, the behavior for DFF[N][E][C][P][R][S] instances does not change. The default initial value of DFF[N][E][R][C] primitives is 0 and DFF[N][E]S/DFF[N][E]P primitives is 1.

See the following examples.

Case 1 - Register with no Control Signal

In this example, the register initial value = 1'b1;

```
module test (clk,d,q);
input clk;
input d;
output q;
reg q=1'b1;

always @ (posedge clk)
begin
   q <= d;
end
endmodule
```

When the register initial value = 1'b0, the same RTL is generated except the DFF does not include an init value.

Copyright © 2013 Synopsys, Inc.

Case 2 - Register with Enable

In this example, the register includes an enable with initial value = 1'b1;

```
module test (clk,en,d,q);
input clk,en;
input d;
output q;
reg q=1'b1;

always @ (posedge clk)
begin
   if(en) q <= d;
end
endmodule
```

When the register initial value = 1'b0, the same RTL is generated except the DFFE does not include an init value.

Case 3 - Register with Asynchronous Control Signals

In this example, the register includes an asynchronous reset with initial value = 1'b1;

```
module test (clk,reset,d,q);
input clk;
input reset;
input d;
output q;
reg q=1'b1;

always @ (posedge clk or posedge reset)
begin
   if(reset) q <= 0;
   else q <=d;
end
endmodule
```

Note: Clock and control signals (reset/set/enable) with -ve polarity are also supported.

Copyright © 2013 Synopsys, Inc.

Case 4 - Register with Synchronous Control Signals

In this example, the register includes an asynchronous reset with initial value = 2'b0;

```
module test (clk,set,d,q);
input clk;
input set;
input [1:0] d;
output [1:0] q;
reg [1:0] q=2'b0;

always @ (posedge clk)
begin
   if(!set) q <= 2'b11;
   else q <=d;
end
endmodule
```

Case 5 - Register with Asynchronous Control Signals and an Enable

In this example, the register includes an asynchronous reset with an enable and initial value = 1'b1;

```
module test (clk,reset,enable,d,q);
input clk;
input reset;
input enable;
input d;
output q;
reg q=1'b1;

always @ (posedge clk or posedge reset)
begin
   if(reset) q <= 0;
   else if(enable) q <=d;
end
endmodule
```

Case 6 - Register with Synchronous Control Signals and an Enable

In this example, the register includes an asynchronous reset with an enable and initial value = 2'b0;

Copyright © 2013 Synopsys, Inc.

```
module test (clk,set,enable,d,q);
input clk;
input set;
input enable;
input [1:0] d;
output [1:0] q;
reg [1:0] q=2'b00;

always @ (posedge clk)
begin
   if (enable)
   begin
      if(!set) q <= 2'b11;
      else q <=d;
   end
end
endmodule
```

Case 7 - Register with Both (async reset and async set)

In this example, the mapper errors out for all conditions (init=0/1).

Register Inference with Both Clock Enable and Control Signals

Achronix provides primitives for registers with both async/sync control signals and clock enables. The Synplify Pro tool supports inference for the following register primitives:

- DFF[N]ER
- DFF[N]ES
- DFF[N]EC
- DFF[N]EP

See the following examples, for the different types of register inference with clock enable and control signals.

Case 1 - Register with asynchronous control signals

If a register with asynchronous reset/set has higher priority than the enable, then DFFER, DFFNER, DFFES, DFFNES with sr_assertion="unclocked" is inferred.

In this example, the register has an asynchronous reset with enable and initial value =1'b1

```
module test (clk,reset,enable,d,q);
input clk;
input reset;
input enable;
input d;
output q;
reg q=1'b1;

always @ (posedge clk or posedge reset)
begin
   if(reset) q <= 0;
   else if(enable) q <=d;
end
endmodule
```


Copyright © 2013 Synopsys, Inc.

The synthesis tool infers DFFER with the parameters sr_assertion="unclocked" and init_value=1'b1. Clock and control signals (reset/set/enable) with -ve polarity are also supported.

Case 2 - Register with synchronous control signals

If a register with synchronous reset/set has higher priority than the enable, then DFFER, DFFNER, DFFES, DFFNES with sr_assertion="clocked" is inferred.

In this example, the register has a synchronous set with enable and initial value= 1'b0.

```
module test (clk,set,enable,d,q);
input clk;
input set;
input enable;
input d;
output q;
reg q=1'b0;
```

```
always @ (posedge clk or negedge set)
begin
   if(!set) q <= 1'b1;
   else if(enable) q <=d;
end
endmodule
```

The tool infers DFFES with the parameters sr_assertion="clocked" and init_value=1'b0. Clock and control signals (reset/set/enable) with -ve polarity are also supported

Copyright © 2013 Synopsys, Inc.

Case 3 - Register with clock enable

The synthesis tool infers DFF[N]EC or DFF[N]EP when the clock enable priority is higher than the synchronous reset/set.

In this example, the register has a clock enable and synchronous reset signal. The clock enable has priority over the reset signal.

```
module test (clk,reset,enable,d,q);
input clk;
input reset;
input enable;
input [1:0] d;
output [1:0] q;
reg [1:0] q=2'b11;

always @ (posedge clk)
begin
if(enable)
   begin
      if(!reset) q <= 2'b0;
      else q <=d;
   end
end
endmodule
```

The tool infers DFFEC with init value set to 1'b1. Clock and control signals (reset/set/enable) with -ve polarity are also supported

Note: The mapper infers DFFEC/DFFEP/DFFNEC/DFFNEP only when the fanout of the preset/reset signal is equal to or greater than 2 within the same hierarchy.

Copyright © 2013 Synopsys, Inc.

Output Files and Forward Annotation

After synthesis, the software generates a log file and output files for Achronix. The following describes files that forward annotate information for the Achronix back-end tools.

- Net Attributes in RTL, on page 1850
- Forward Annotation of RTL Attributes to the Netlist, on page 1853

Net Attributes in RTL

The synthesis software accepts properties or attributes for nets that have the syn_keep attribute in the RTL. Use this directive only in the RTL source code and with a single property associated with the net. These properties are forward-annotated to the Verilog netlist (VMA) file. The Achronix P&R tool retains these properties for the nets to perform certain placement-based optimizations with the specified attribute values.

Verilog Syntax and Example

object /* synthesis syn_keep = 1 <attribute>="<value>" */;

(* syn_keep = 1, <attribute>="<value>" *) object;

Here is an example showing how to use net attributes in the RTL source code.

```
module ACX_LUT4 (din0,din1,din2,din3,dout)
   /* synthesis syn_black_box */ ;
input din0, din1, din2, din3;
output dout;
parameter lut_function = 16'h0000;
endmodule

module net_attribute (clk, in, dout);
input clk, in;
output reg dout;
reg din0, din1;
wire d1_out;
assign d1_out=din1;
wire lut0_out;
```

```
(* syn_keep = 1, weight="3.0" *) wire lut1_out;
wire lut2_out;
wire lut3_out;
wire lut4_out;
wire lut5_out /* synthesis syn_keep = 1 weight=4.56778 */;

always @ (posedge clk)
begin
   din0 <=in;
   din1 <=din0;
   dout <= lut5_out;
end

// LUT Chain //
ACX_LUT4 inst0 (.din0(d1_out), .din1(1'b1), .din2(1'b0),
   .din3(1'b0), .dout(lut0_out) );
defparam inst0.lut_function=16'h000A;
ACX_LUT4 inst1 (.din0(lut0_out), .din1(1'b1), .din2(1'b0),
   .din3(1'b0), .dout(lut1_out) );
defparam inst1.lut_function=16'h000B;
ACX_LUT4 inst2 (.din0(lut1_out), .din1(1'b1), .din2(1'b0),
   .din3(1'b0), .dout(lut2_out) );
defparam inst2.lut_function=16'h000C;
ACX_LUT4 inst3 (.din0(lut2_out), .din1(1'b1), .din2(1'b0),
   .din3(1'b0), .dout(lut3_out) );
defparam inst3.lut_function=16'h000D;
ACX_LUT4 inst4 (.din0(lut3_out), .din1(1'b1), .din2(1'b0),
   .din3(1'b0), .dout(lut4_out) );
defparam inst4.lut_function=16'h000E;
ACX_LUT4 inst5 (.din0(lut4_out), .din1(1'b1), .din2(1'b0),
   .din3(1'b0), .dout(lut5_out) );
defparam inst5.lut_function=16'hA00A;

endmodule
```

The following figure displays the results of the syn_keep attribute in the RTL view.

Copyright © 2013 Synopsys, Inc.

For this example, the following attributes are forward annotated to the output netlist (VMA).

```
(* weight="3.0" *)wire lut1_out ;

(* weight=4.567780 *)wire lut5_out ;
```

Forward Annotation of RTL Attributes to the Netlist

The synthesis tool forward-annotates attributes applied to RTL objects, such as registers, modules, instantiated components, and I/O ports to their respective objects in the generated netlist.

To forward-annotate the RTL attributes/directives, you must set the synthesis directives/attributes on the RTL objects as follows:

Object	Synthesis Directive/Attribute
modules	`syn_hier="hard"`
Instantiated components	`syn_noprune`
Input/output ports	`syn_hier="hard"` on the module containing the ports
Registers	`syn_preserve`

Examples of how to specify the attributes in the RTL as shown below:

Using Verilog

Here are some Verilog examples for specifying attributes:

- Example 1 – `(* syn_preserve = 1, weight="3.0" *) reg my_reg;`
- Example 2 –
  `(* weight="6.0" *) reg my_reg /* synthesis syn_preserve=1 */;`
- Example 3 –
  `(* syn_preserve = 1 *) reg my_reg/* synthesis weight=7.7 */;`
- Example 4 – `reg my_reg /* synthesis syn_preserve = 1 weight=4 */;`

Using VHDL

Here are some VHDL examples for specifying attributes:

- `syn_preserve` – boolean
- `syn_preserve` of `en_gate` – signal is true
- `priority` – string
- priority of `en_gate` – signal is low

Copyright © 2013 Synopsys, Inc.

Note: Using the synthesis directives/attributes along with the RTL attributes can prevent certain optimizations; this can impact the QoR.

See the *Summary of Attributes and Directives*, on page 1169 for a detailed description of how to use the synthesis directives/attributes.

Limitations

RTL attributes that begin with "." or "syn" are not forward-annotated to the netlist.

Achronix Technology Support

This section provides guidelines for using the synthesis tool with devices for the Speedster22iHD technology. The topics include:

- Device Mapping Options, on page 1855
- Tcl set_option Command, on page 1856
- Tcl project Command, on page 1860

Device Mapping Options

Device mapping options are synthesis algorithms to optimize your design. To set these options, select Project -> Implementation Options and set the options on the Device tab. You can set other options on other tabs of this dialog box. For descriptions of optimization options and constraints, see *Options Panel*, on page 197 and *Constraints Panel*, on page 200.

The following table lists device mapping options (Device tab) for the Speedster22iHD technology.

Clock Conversion	Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details.
Disable I/O Insertion	When enabled, it prevents automatic I/O insertion during synthesis. By default, it is disabled. See *Disable I/O Insertion*, on page 1859 for details.
Fanout Guide	Sets a guideline for the fanout limit. The default is 100. For tips on setting and using fanout limits, see Setting Fanout Limits, on page 444 and Controlling Buffering and Replication, on page 446in the User Guide.
Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.

Copyright © 2013 Synopsys, Inc.

Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Retime Registers Forward	Forward retime registers across combinatorial logic towards the output to improve area QoR.
Time Borrowing	Allows timing for a register to give time to the previous cycle or borrow from the next cycle. See *Time Borrowing*, on page 1859 for more information. For the Speedster22iHD device, Time Borrowing is OFF by default.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Interface Timing for Compile Points*, on page 1101, for details.

You can also use the corresponding set_option Tcl command to enable/disable these options. See *Tcl set_option Command*, on page 1856 for more information.

Tcl set_option Command

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles. The set_option Tcl command lets you specify the same implementation options as you do through the dialog box displayed in the Project view with Project -> Implementation Options.

The following table lists the options for the Speedster22iHD technology. See *set_option*, on page 1679 or enter help set_option in the Tcl Script window for a full list of syntax and options.

OPTION	DESCRIPTION
Target Technology Options	
-technology *keyword*	Sets the target technology for the implementation. Achronix library supported includes: AchronixSpeedster22iHD.
-part *partName*	Specifies a part for the implementation. Refer to the Implementation Options dialog box for available choices (*Implementation Options Command*, on page 194).
-speed_grade *value*	Sets the speed grade for the implementation. Refer to the Implementation Options dialog box for available choices (*Implementation Options Command*, on page 194).
-package *packageName*	Specifies the package for the implementation. Refer to the Implementation Options dialog box for available choices (*Implementation Options Command*, on page 194).
Optimization Options	
-disable_io_insertion 1 | 0	Controls I/O insertion during synthesis. The default is 0, where I/Os are inserted automatically. To disable I/O insertion, set it to 1. See *Disable I/O Insertion*, on page 1859 for a description.
-fanout_guide *value*	Sets the fanout limit. Default is 10000. See Setting Fanout Limits, on page 444 for details.
-pipe 1 | 0	Determines whether you run designs at a faster frequency by moving registers after the multiplier into the multiplier. The default is 0. See Pipelining, on page 426 of the *User Guide* for information on how to use it.
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
-resource_sharing 1|0	Enables/disables resource sharing. See Sharing Resources, on page 448 in the *User Guide* for information about using it.
-retime_registers_forward 1 | 0	Forward retime registers across combinatorial logic towards the output to improve area QoR.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-retiming 1 | 0	Determines whether you use retiming to optimize your design. When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). You must also enable -pipe. See Retiming, on page 430 of the *User Guide* f or information on how to use it.
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
-symbolic_fsm_compiler 1 | 0	Enables/disables the FSM compiler. Controls the use of FSM synthesis for state machines. The default is false (FSM Compiler disabled). Value can be 1 or true, 0 or false. See Running the FSM Compiler, on page 455 in the *User Guide* for details of its use.
-time_borrow 1|0	Allows timing for a register to give time to the previous cycle or borrow from the next cycle. See *Time Borrowing*, on page 1859 for more information. For the Speedster22iHD device, Time Borrowing is OFF by default.
-update_models_cp 1 | 0	Determines whether changes to a locked compile point force the parents to be remapped to use the new timing model of the child. See *Compile Point Remapping in Achronix Designs*, on page 1859, for details. • 1 Forces remapping of parents • 0 Does not remap parents

Disable I/O Insertion

I/O pads are automatically inserted into the design. If you manually instantiate I/Os, I/Os are inserted only for the pins that need them.

If you do not want to insert *any* I/Os in the design, select the Disable I/O Insertion check box in the Implementation Options dialog box. This is useful for bottom-up compiles, because you can check the area your logic blocks use before you synthesize the whole design. If you disable I/O insertion, you will not get any I/Os in the design unless you manually instantiate them.

Time Borrowing

Achronix designs can implement time borrowing which allows timing for a register to give time to the previous cycle or borrow from the next cycle. For example, if the data input of a register has negative slack and the output has positive slack, the synthesis software can redistribute some of the excess slack from the output to the input. Paths with positive slack can borrow as much time from the previous or successive paths as required. This option helps to improve the average maximum frequency for the design.

Compile Point Remapping in Achronix Designs

Compile points are smaller synthesis units, which lets you break up a larger design into smaller units and do incremental synthesis. See *Compile Point Synthesis*, on page 1104 and *Compile Point Types*, on page 1092 for information about the flow and compile points.

The Update Compile Point Timing Data option determines whether changes to a compile point force the remapping of its parents, taking into account the new timing model of the child. The term *child* refers to a compile point that is contained inside another; the term *parent* refers to the compile point that contains the child. These terms are thus not used here in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points.

Copyright © 2013 Synopsys, Inc.

The following table describes the settings for the Update Compile Point Timing Data option:

Disabled	(Default). Only compile points that have changed are remapped, and their remapping does *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used to map and optimize its parents. If the option is disabled and the *interface* of a locked compile point is changed, the immediate parent of the compile point must be changed accordingly. In this case, both are remapped, and the *updated* timing model of the child is used when remapping this parent.
Enabled	Compile point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account because the option was disabled. The timing model of a compile point is updated when either of the following is true: • The compile point is remapped, and the Update Compile Point Timing Data option is enabled. • The interface of the compile point is changed.

Tcl project Command

You can use the project Tcl command to specify the same result file formats and filenames as are available from the Implementation Results panel of the Implementation Options dialog box.

This table lists the options for the project Tcl command for Speedster22iHD technology. See *project*, on page 1662 for complete syntax and options for this command, or type `help project` in the Tcl Script window:

Option	Description
-result_file *value*	Specifies the name of the synthesized netlist for the implementation.
-result_format vma	Specifies the synthesis result file format for the implementation. The format must be vma (lower-case letters).

Achronix Attribute and Directive Summary

The following table summarizes the synthesis and Achronix-specific attributes and directives available with the Achronix Technology. Complete descriptions and examples can be found in Chapter 16, *Attribute and Directive Syntax*.

Attribute/Directive	Description
black_box_pad_pin	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
full_case	Specifies that a Verilog case statement has covered all possible cases.
loop_limit	Specifies a loop iteration limit for for loops.
parallel_case	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
pragma translate_off/pragma translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
syn_allow_retiming	Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.
syn_black_box	Defines a black box for synthesis.
syn_direct_enable	Identifies which signal to use as the enable input to an enable flip-flop when multiple candidates are possible.
syn_dspstyle	Determines whether the multiplier logical structures are mapped into BMACC56 or to logic.
syn_encoding	Specifies the encoding style for state machines.
syn_enum_encoding	Specifies the encoding style for enumerated types (VHDL only).

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_hier	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
syn_isclock	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_keep	Prevents the internal signal from being removed during synthesis and optimization.
syn_looplimit	Specifies a loop iteration limit for while loops.
syn_maxfan	Overrides the default fanout guide for an individual input port, net, or register output.
syn_max_memsize_reg	Allows you to specify the maximum number of registers that can be mapped to an inferred RAM above the limit for which the tool errors out.
syn_netlist_hierarchy	Determines if the VMA output netlist is flat or hierarchical.
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.
syn_pipeline	Specifies that registers be moved into multipliers and ROMs in order to improve frequency.
syn_preserve	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
syn_probe	Adds probe points for testing and debugging.
syn_ramstyle	Determines the way in which RAMs are implemented.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
syn_replicate	Disables replication.
syn_resources	Specifies resources used in black boxes.
syn_romstyle (Achronix)	Determines how ROM architectures are implemented.
syn_sharing	Specifies resource sharing of operators.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_state_machine	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n>	Defines timing clock to output delay through the black box. The *n* indicates a value between 1 and 10.
syn_tpd<n>	Specifies timing propagation for combinational delay through the black box. The *n* indicates a value between 1 and 10.
syn_tristate	Specifies that a black-box pin is a tristate pin.
syn_tsu<n>	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.
syn_unconnected_inputs	You can specify that input pins of an instance be left floating (unassigned in the RTL).
syn_useenables	Prevents generation of registers with clock enable pins.
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

APPENDIX B

Designing with Altera

The following topics describe how to design and synthesize with different Altera technologies:

- Basic Support for Altera Designs, on page 1866
- Input for Altera Designs, on page 1876
- Altera Components, on page 1878
- Altera RAMs and ROMs, on page 1882
- Altera LPMs, on page 1898
- DSP Blocks for Stratix, Arria, and Cyclone, on page 1904
- Altera Optimizations, on page 1910
- Altera Reports, on page 1917
- Forward-annotation to Altera Tools, on page 1920
- Integration with Altera Tools and Processes, on page 1923
- Stratix, Arria, and Cyclone Families, on page 1927
- MAX II and MAX V Families, on page 1931
- MAX Family, on page 1935
- Altera Attribute and Directive Summary, on page 1944

Copyright © 2013 Synopsys, Inc.

Basic Support for Altera Designs

The following topics briefly describe flows, constraints, and attributes supported by the synthesis tools for Altera designs:

- Altera Device Support, on page 1866
- Supported Synthesis Design Flows for Altera Designs, on page 1867
- Synthesis Constraints and Attributes for Altera Designs, on page 1867

For information on support in specific areas, see the other sections in this chapter.

Altera Device Support

The synthesis tools create technology-specific netlists for various Altera FPGA technologies. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Implementation Options dialog box.

You can use the Synplify Premier software for all Altera technologies in logic synthesis mode, but physical synthesis is not available for older Altera technologies. If you have the Synplify Premier tool and the Physical Synthesis switch does not appear on the Options tab of the Implementation Options dialog box for the selected technology, then physical synthesis for that technology is not supported.

See the following for device-specific information:

- Stratix, Arria, and Cyclone Families, on page 1927
- MAX II and MAX V Families, on page 1931
- MAX Family, on page 1935

After synthesis, the tool generates netlists that are used as input to the Altera place-and-route tool. The accepted netlists are updated periodically, so check the most current version of the place-and-route tool. The software generates `vqm` netlist files as input for the Quartus II place-and-route tool.

Copyright © 2013 Synopsys, Inc.

Supported Synthesis Design Flows for Altera Designs

The following summarizes the synthesis design flows supported for Altera designs. All flows may not be available for all technologies:

Logic synthesis	See *Logic Synthesis Design Flow*, on page 36
Logic synthesis with floorplan	(Synplify Premier with Design Planner). See *Design Plan-Based Logic Synthesis*, on page 42 in the *User Guide*.
Logic synthesis with fast synthesis	(Synplify Premier). See Chapter 19, *Fast Synthesis* in the *User Guide*.
Logic synthesis with enhanced optimizations	(Synplify Premier). See *Logic Synthesis with Enhanced Optimization*, on page 40 in the *User Guide*.
Graph-based physical synthesis	(Synplify Premier). See *Altera Graph-Based Physical Synthesis*, on page 46 in the *User Guide*.
Graph-based physical synthesis with a floorplan	(Synplify Premier with Design Planner). See *Altera Graph-Based Physical Synthesis with Design Planner*, on page 49 in the *User Guide*.
Floorplan-based physical synthesis	(Synplify Premier with Design Planner). See *Design Plan-based Physical Synthesis*, on page 51 in the *User Guide*.

The synthesis tools also support other flows which allow better integration with the Altera components, flows, or back-end tools. See *Integration with Altera Tools and Processes*, on page 1923 for information.

Synthesis Constraints and Attributes for Altera Designs

The synthesis tools let you specify timing constraints, general HDL attributes, and Altera-specific attributes to improve your design. You can manage the attributes and constraints in the SCOPE interface. Altera has vendor-specific I/O standard constraints it supports for synthesis. For a list of supported I/O standards, see *Altera I/O Standards*, on page 1867.

Altera I/O Standards

The following tables show the applicable I/O standards for the Stratix, Stratix GX, Stratix II, and Stratix II GX families with their equivalent Quartus I/O standards and for the Cyclone and Cylone II families with their equivalent Quartus I/O standards.

Copyright © 2013 Synopsys, Inc.

Stratix and Stratix GX I/O Standards

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix	Stratix GX
AGP1X	AGP 1X	X	X
AGP2X	AGP 2X	X	X
CTT	CTT	X	X
DIFF_HSTL_15_Class_I	Differential 1.5-V HSTL Class I	X	X
DIFF_HSTL_15_Class_II	Differential 1.5-V HSTL Class II		
DIFF_HSTL_18_Class_I	Differential 1.8-V HSTL Class I		
DIFF_HSTL_18_Class_II	Differential 1.8-V HSTL Class II		
DIFF_SSTL_18_Class_I	Differential 1.8-V SSTL Class I		
DIFF_SSTL_18_Class_II	Differential 1.8-V SSTL Class II		
DIFF_SSTL_2_Class_I	Differential SSTL-2		X
DIFF_SSTL_2_Class_II	Differential 2.5-V SSTL Class II	X	
GTL	GTL	X	X
GTL+	GTL+	X	X
HSTL_Class_I			X
HSTL_Class_II			X
HSTL_12	1.2-V HSTL		
HSTL_15_Class_I	1.5-V HSTL Class I	X	X
HSTL_15_Class_II	1.5-V HSTL Class II	X	X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix	Stratix GX
HSTL_18_Class_I	1.8-V HSTL Class I	X	X
HSTL_18_Class_II	1.8-V HSTL Class II	X	X
HyperTransport	HyperTransport	X	X
LVCMOS_15	1.5V	X	
LVCMOS_18	1.8V	X	X
LVCMOS_25	2.5V	X	X
LVCMOS_33	LVCMOS	X	X
LVDS	LVDS	X	X
LVPECL	Differential LVPECL	X	X
LVTTL	LVTTL	X	X
PCI33	3.3-V PCI	X	X
PCI-X_133	3.3-V PCI-X	X	X
PCML	PCML	X	X
SSTL_18_Class_I	SSTL-18 Class I	X	X
SSTL_18_Class_II	SSTL-18 Class II	X	X
SSTL_2_Class_I	SSTL-2 Class I	X	X
SSTL_2_Class_II	SSTL-2 Class II	X	X
SSTL_3_Class_I	SSTL-3 Class I	X	X
SSTL_3_Class_II	SSTL-3 Class II	X	X

Copyright © 2013 Synopsys, Inc.

Stratix II and Stratix II GX I/O Standards

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix II	Stratix II GX
DIFF_HSTL_15_Class_I	Differential 1.5-V HSTL Class I	X	X
DIFF_HSTL_15_Class_II	Differential 1.5-V HSTL Class II	X	X
DIFF_HSTL_18_Class_I	Differential 1.8-V HSTL Class I	X	X
DIFF_HSTL_18_Class_II	Differential 1.8-V HSTL Class II	X	
DIFF_SSTL_18_Class_I	Differential 1.8-V SSTL Class I		X
DIFF_SSTL_18_Class_II	Differential 1.8-V SSTL Class II	X	X
DIFF_SSTL_2_Class_I	Differential 2.5-V SSTL Class I	X	X
DIFF_SSTL_2_Class_II	Differential 2.5-V SSTL Class II	X	X
HSTL_15_Class_I	1.5-V HSTL Class I	X	X
HSTL_15_Class_II	1.5-V HSTL Class II	X	X
HSTL_18_Class_I	1.8-V HSTL Class I	X	X
HSTL_18_Class_II	1.8-V HSTL Class II	X	X
HyperTransport	HyperTransport	X	
LVCMOS_15	1.5V	X	X
LVCMOS_18	1.8V	X	X
LVCMOS_25	2.5V	X	X
LVCMOS_33	3.0-V LVCMOS	X	X
LVDS	LVDS	X	X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix II	Stratix II GX
LVPECL	Differential LVPECL	X	X
LVTTL	3.0-V LVTTL	X	
PCI33	3.3-V PCI	X	X
PCI-X_133	3.3-V PCI-X	X	X
SSTL_18_Class_I	SSTL-18 Class I	X	X
SSTL_18_Class_II	SSTL-18 Class II	X	X
SSTL_2_Class_I	SSTL-2 Class I	X	X
SSTL_2_Class_II	SSTL-2 Class II	X	X

Stratix III and Stratix IV I/O Standards

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV
1_2	1.2 V	X	
1_5	1.5 V	X	
1_8	1.8 V	X	
2_5	2.5 V	X	
DIFF_HSTL_12_I	Differential 1.2-V HSTL Class I	X	X
DIFF_HSTL_12_II	Differential 1.2-V HSTL Class II	X	X
DIFF_HSTL_15_I	Differential 1.5-V HSTL Class I	X	
DIFF_HSTL_15_II	Differential 1.5-V HSTL Class II	X	
DIFF_HSTL_15_Class_I	Differential 1.5-V HSTL Class I		X
DIFF_HSTL_15_Class_II	Differential 1.5-V HSTL Class II		X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV
DIFF_HSTL_18_Class_I	Differential 1.8-V HSTL Class I	X	X
DIFF_HSTL_18_Class_II	Differential 1.8-V HSTL Class II	X	X
DIFF_SSTL_15_I	Differential 1.5-V SSTL Class I		X
DIFF_SSTL_15_II	Differential 1.5-V SSTL Class II		X
DIFF_SSTL_15_Class_I	Differential 1.5-V SSTL Class I	X	
DIFF_SSTL_15_Class_II	Differential 1.5-V SSTL Class II	X	
DIFF_SSTL_18_Class_I	Differential 1.8-V SSTL Class I	X	X
DIFF_SSTL_18_Class_II	Differential 1.8-V SSTL Class II	X	X
DIFF_SSTL_2_Class_I	Differential 2.5-V SSTL Class I	X	X
DIFF_SSTL_2_Class_II	Differential 2.5-V SSTL Class II	X	X
HCSL	HCSL		X
HSTL_12_I	1.2-V HSTL Class I	X	X
HSTL_12_II	1.2-V HSTL Class II	X	X
HSTL_15_Class_I	1.5-V HSTL Class I	X	X
HSTL_15_Class_II	1.5-V HSTL Class II	X	X
HSTL_18_Class_I	1.8-V HSTL Class I	X	X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV
HSTL_18_Class_II	1.8-V HSTL Class II	X	X
LVCMOS	3.0-V LVCMOS	X	
LVCMOS_12	1.2 V		X
LVCMOS_15	1.5 V		X
LVCMOS_18	1.5 V		X
LVCMOS_25	2.5 V		X
LVCMOS_33	3.3-V LVCMOSV		X
LVDS	LVDS		X
LVDS_E_1R	LVDS_E_R1	X	X
LVDS_E_3R	LVDS_E_3R	X	X
LVPECL	Differential LVPECL		X
LVTTL	3.0-V LVTTL	X	
LVTTL_33	3.3-V LVTTL		X
MINI_LVDS	mini-LVDS		X
MINI_LVDS_1R	mini-LVDS_1R	X	X
MINI_LVDS_3R	mini-LVDS_3R	X	X
PCI	3.0-V PCI	X	X
PCI_X	3.0-V PCI-X	X	X
PCML_12	1.2-V PCML		X
PCML_14	1.4-V PCML		X
PCML_15	1.5-V PCML		X
PCML_25	2.5-V PCML		X
RSDS	RSDS		X
RSDS_E_1R	RSDS_E_1R	X	X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Stratix III	Stratix IV
RSDS_E_3R	RSDS_E_3R	X	X
SSTL_15_I	SSTL-15 Class I	X	X
SSTL_15_II	SSTL-15 Class II	X	X
SSTL_18_Class_I	SSTL-18 Class I	X	X
SSTL_18_Class_II	SSTL-18 Class II	X	X
SSTL_2_Class_I	SSTL-2 Class I	X	X
SSTL_2_Class_II	SSTL-2 Class II	X	X

Cyclone I/O Standards

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone	Cyclone II
DIFF_HSTL_15_Class_I	Differential 1.5-V HSTL Class I		X
DIFF_HSTL_15_Class_II	Differential 1.5-V HSTL Class II		X
DIFF_HSTL_18_Class_I	Differential 1.8-V HSTL Class I		X
DIFF_HSTL_18_Class_II	Differential 1.8-V HSTL Class II		X
DIFF_SSTL_18_Class_I	Differential 1.8-V SSTL Class I		X
DIFF_SSTL_18_Class_II	Differential 1.8-V SSTL Class II		X
DIFF_SSTL_2_Class_I	Differential SSTL-2	X	X
DIFF_SSTL_2_Class_II	Differential 2.5-V SSTL Class II		X
HSTL_15_Class_I	1.5-V HSTL Class I		X
HSTL_15_Class_II	1.5-V HSTL Class II		X
HSTL_18_Class_I	1.8-V HSTL Class I		X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	Quartus I/O Standard	Cyclone	Cyclone II
HSTL_18_Class_II	1.8-V HSTL Class II		X
LVCMOS_15	1.5V	X	X
LVCMOS_18	1.8V	X	X
LVCMOS_25	2.5V	X	X
LVCMOS_33	LVCMOS	X	X
LVDS	LVDS	X	X
LVPECL	Differential LVPECL		X
LVTTL	LVTTL	X	X
MINI_LVDS	mini-LVDS		X
PCI33	3.3-V PCI	X	X
PCI-X_133	3.3-V PCI-X		X
RSDS	RSDS	X	X
SSTL_18_Class_I	SSTL-18 Class I		X
SSTL_18_Class_II	SSTL-18 Class II		X
SSTL_2_Class_I	SSTL-2 Class I	X	X
SSTL_2_Class_II	SSTL-2 Class II	X	X
SSTL_3_Class_I	SSTL-3 Class I	X	
SSTL_3_Class_II	SSTL-3 Class II	X	

See Also:

- *Industry I/O Standards*, on page 496 for a list of industry I/O standards.
- See *Altera Attribute and Directive Summary*, on page 1944 for a list of Altera attributes and directives.

Copyright © 2013 Synopsys, Inc.

Input for Altera Designs

The following describe how to handle different kinds of input to Altera designs:

- Altera Verilog Libraries, on page 1876
- Altera VHDL Libraries, on page 1876
- IP Core Support, on page 1876
- QSF Constraints, on page 1878
- Initial Values for Altera RAMs, on page 1878

Altera Verilog Libraries

The synthesis tool automatically loads the Altera library files, so you do not need to add these files to your project file. To automatically load other common and family-specific Verilog libraries, select the correct Quartus II place-and-route tool version from the drop-down menu on the Implementation Results tab of the Implementation Options dialog box. The Altera macro files are in *install_dir*/lib/altera/*quartusVersion*.

Altera VHDL Libraries

The Altera macro files are in *install_dir*/lib/altera/*quartusVersion*. Unlike Verilog designs where the appropriate macro files automatically become available when you select the Quartus version, in VHDL designs you must manually specify a non-default library by adding a line to the `prj` file.

The following example shows the syntax to be added to the `prj` file to link in a user-instantiated LPM component called compinv.vhd from the Quartus II 10.0 library, when Quartus II 9.1 is the default library for the design:

```
add_file -vhdl -lib X:/Altera/quartus_II100/quartus/lpm
   "./compinv.vhd"
```

IP Core Support

In a typical design flow, IP is instantiated as black boxes in the VHDL/Verilog source. EDIF files for these modules are supplied to the place-and-route tool to complete the design.

Copyright © 2013 Synopsys, Inc.

This design flow has the following limitations:

- The Synopsys FPGA software does not have timing models for the black boxes which limits its capability to perform realistic timing analysis and optimizations on the remainder of the design. Although you can add timing models for black boxes, this process is tedious and error prone even for very small blocks.
- The Synopsys FPGA software does not have resource usage summaries for the black boxes, which limits its capability to correctly report the resources needed for the complete design. Without this information, optimizations including RAM/ROM to block RAM mapping and clock buffer insertion cannot be effectively performed.

The following briefly describe how to work with various kinds of Altera IP:

Stratix Clearbox Megafunctions	The synthesis software can implement megafunctions generated by the Altera Megawizard tool as Clearboxes. It uses the underlying structural information from the Clearbox model for timing and resource allocation. For details of this flow, see *Implementing Megafunctions with Clearbox Models*, on page 1002 of the *User Guide*.
SOPC Builder Cores	The synthesis tools can incorporate IP cores generated with SOPC Builder in Altera designs. For details of the procedure, see *Including Altera Processor Cores Generated in SOPC Builder*, on page 1023 of the *User Guide*.
Grey Box Megafunctions	You can incorporate encrypted IP as grey boxes in your Altera designs. The synthesis tool uses the grey box netlist information for timing and resource allocation, but does not include the grey box in the output. For details of the procedure, see *Implementing Megafunctions with Grey Box Models*, on page 1012 in the *User Guide*. You can also import the megafunction as an IP package (*Including Altera MegaCore IP Using an IP Package*, on page 1019 in the *User Guide*), but it is recommended that you use the grey box netlist method instead.
SOPC Builder Components	The synthesis tool lets you use components generated with SOPC Builder in your Altera designs. For details of the procedure, see *Working with SOPC Builder Components*, on page 1029 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

QSF Constraints

The synthesis tools include functionality to translate Altera QSF constraints to the SDC format. See *Translating Altera QSF Constraints*, on page 193 in the *User Guide* for details.

Initial Values for Altera RAMs

You can specify startup values for RAMs in Altera designs. See *Initializing RAMs*, on page 394 in the *User Guide* for step-by-step procedures on setting initial values in Verilog or VHDL code.

Altera Components

The following topics describe how the synthesis tools handle various Altera components, and show you how to work with or manipulate them during synthesis to get the results you need:

- Altera Macros, on page 1879
- Altera Black Boxes, on page 1879
- Altera ATOM Primitives
- Altera LPMs, on page 1898
- Altera Multipliers, on page 1880
- Special Registers in Altera Designs, on page 1880
- Altera PLLs, on page 1880
- Stratix ShiftTaps, on page 1881
- Altera Pad Cells, on page 1881

See *Altera RAMs and ROMs*, on page 1882, *Altera LPMs*, on page 1898, and *DSP Blocks for Stratix, Arria, and Cyclone*, on page 1904 for information about other Altera components.

Copyright © 2013 Synopsys, Inc.

Altera Macros

You can select macros from the Altera library of predefined macros and instantiate them as black boxes. You can instantiate global buffers for clocks, resets, sets, and other heavily loaded signals. Make sure to use the appropriate macro library, located in the *install_dir*/lib/altera directory. You can have Verilog or VHDL macros in Altera designs. Refer to the Altera documentation for supported macros.

Verilog

For Verilog designs, the tool automatically adds the Verilog component declaration file for the target technology to your project from the lib/altera/*quartusVersion* directory. To make sure that the correct version of the component declaration file is included, select the version of Quartus that you intend to use from the Implementation Results tab of the Implementation Options panel.

For example, if you are using Stratix III with Quartus 11.1, select Quartus II 11.1 from the drop-down menu to use the stratixiii.v file from the quartus_II111 directory to access the port and parameter definitions for the primitives.

VHDL

For VHDL designs, instantiate Altera VHDL macro library elements as predefined black boxes into your design. Add the following library and use clauses to the top of the source files in which you instantiate the macros:

```
library family ;
use altera.family.all;
```

In this example, *family* is the name of the library family such as stratixiigx or cycloneiii (for a list of family names, see the location.map file in the /lib/vhd directory).

Altera Black Boxes

You can create an instance of any externally defined macro, including a user-defined macro or Altera macro such as an I/O or flip-flop, by instantiating a black box in your Verilog or VHDL source code. These black boxes are empty Verilog module descriptions and VHDL component declarations. For a procedure to instantiate black boxes, see *Defining Black Boxes for Synthesis, on page 330* in the *User Guide*.

Altera ATOM Primitives

Enable or disable mapping to ATOM primitives with the Map Logic to ATOMs option (Project->Implementation Options->Device). Mapping to ATOM primitives is supported only by the MAX architectures.

Copyright © 2013 Synopsys, Inc.

When enabled, the synthesis tool maps technology components directly to Quartus II ATOM primitives (like logic elements, memory elements, and I/O cells), thus reducing the total runtime for placement and routing. Using the Verilog netlist as input, Quartus II builds the logic database, skips synthesis, and performs placement and routing. By mapping to the ATOM primitives, the synthesis tool is able to more precisely control the logic implementation in various Altera device families, and more accurately estimate timing.

Altera Multipliers

Use the syn_multstyle attribute if you need to allocate the multiplier resources in your design. Set it to lpm_mult to infer LPM functions, and block_mult to infer DSP blocks.

Special Registers in Altera Designs

The compiler can infer additional register primitive types so that the mapper can implement a better solution, when it targets Altera technologies:

Registers with clock enables	The compiler identifies and extracts clock-enable logic for registers. The RTL view displays registers with enables as special primitives with an extra pin for the enable signal. The special primitives are dffe, dffre, dffse, dffrse, dffpatre and dffpatrse. Set syn_direct_enable in the HDL source file (not in the SCOPE spreadsheet or a constraint file) to direct the compiler to infer desirable clock enables. For further syntax details see *syn_direct_enable*, on page 1248. By default, registers without clock enables will be inferred.
Registers with synchronous sets/resets	The compiler identifies and extracts registers with synchronous sets and resets, and passes them to the mapper. The special primitives are sdffr, sdffs, sdffrs, sdffpat, sdffre, sdffse and sdffpate. The compiler does this automatically and does not require a directive.

Altera PLLs

The synthesis software recognizes Altera PLL components, or altplls. You generate the component files with the Altera Megawizard tool and then add constraints before you synthesize. For details of the procedure, see *Working with Altera PLLs*, on page 760 of the *User Guide*.

Stratix ShiftTaps

The synthesis tools can implement shift registers with taps as ShiftTap megafunctions in Stratix designs. A ShiftTap is a parameterized array of shift registers with intermediate tap points. This configuration is inferred as the Stratix megafunction altshift_tap in the `vqm` or `vm` file that Quartus maps into either altsyncram or registers. see *Inferring Shift Registers*, on page 405 of the *User Guide* for information on implementing these components.

Altera Pad Cells

The synthesis tools automatically insert pad cells when known primitives (lcell, register, ram_block, mlab, MAC) are connected to top-level ports. However, the tool does not insert pad cells when the pins of a black box or Quartus megafunction with unknown contents are connected to top-level ports. If the black box or Megafunction already contains pad cells, this prevents “double pad cell” failures during P&R.

To override this behavior and insert pad cells on black boxes or megafunctions, include an enable_io_map.txt file in the implementation directory. This file specifies how you want to handle pad cells, and uses the following syntax to do so:

To insert all pad cells, use...	*cellName*, **any**
To insert a pad cell on a specific pin, use...	*cellName*, *pinName*
To insert pad cells on all inputs, use...	*cellName*, **input**
To insert pad cells on all outputs, use...	*cellName*, **output**

For example:

```
altsyncram, any

test0, in0

test1, input

test2, output\
```


Copyright © 2013 Synopsys, Inc.

Altera RAMs and ROMs

The following describe how the synthesis tools infer and implement Altera RAMs and ROMs. For information about other Altera components, see *Altera Components*, on page 1878.

- Altera RAMs, on page 1882
- Stratix, Arria, and Cyclone RAMs, on page 1882
- Stratix V Byte-Wide Write Enable RAM, on page 1884
- Stratix III and Stratix IV RAMs with Control Signals, on page 1895
- Cyclone III RAMs with Control Signals, on page 1896
- Altera ROMs, on page 1896

See *Altera Components*, on page 1878, *Altera LPMs*, on page 1898, and *DSP Blocks for Stratix, Arria, and Cyclone*, on page 1904 for descriptions of other Altera components.

Altera RAMs

From the HDL source code, the synthesis tool automatically infers RAMs and generates single-port or simple dual-port RAMS. This section contains an overview; for details refer to the RAM inference sections in the HDL language chapters: *Verilog Coding for Altera RAM Inference*, on page 717, *VHDL Guidelines for RAM Inference*, on page 940, and *VHDL Coding for Altera RAM Inference*, on page 948.

Stratix, Arria, and Cyclone RAMs

This table describes RAM inference for Stratix, Arria, and Cyclone devices. In general, the tool recognizes RTL RAMs and maps them to the altsyncram mega function (RAMs are mapped to Stratix II, Stratix III, Stratix IV, or Stratix V RAM blocks).

Copyright © 2013 Synopsys, Inc.

RAMs with...	Mapped to...
Synchronous read and write	RAM. The read operation can be synchronized by registering either the read address or the output of the RAM.
Asynchronous read or write	Logic
Only one address bus	RAM (generally in single-port mode) However, because old data cannot be obtained in single-port mode, whenever only the output of a RAM is registered, the RAM is mapped in dual-port mode.
Two address buses (one each for read and write)	RAM (dual-port mode) Glue logic is created if new data must be obtained in dual-port mode. You can disable the creation of glue logic by setting the syn_ramstyle value to no_rw_check. See *Stratix, Arria and Cyclone Dual-Port Verilog RAM Examples*, on page 720 and *Stratix, Arria and Cyclone Dual-Port VHDL RAM Example*, on page 948 for dual-port RAM descriptions and examples.
Two address buses, (one for read and write, the other for read only)	RAM (in BIDIR mode)

You can also use the syn_ramstyle attribute (*syn_ramstyle*, on page 1467) to map the RAM to particular resources. For example:

Stratix, Stratix GX, Stratix II (and HardCopy II), Stratix II GX	MRAM, M4K, M512
Stratix III (and HardCopy III), Stratix IV (and HardCopy IV)	M144K, M9K
Stratix V, Cyclone V, Arria V	M20K
Cyclone, Cyclone II	M4K
Cyclone III, Cyclone IV E, Cyclone IV GX	M9K
Arria GX, Arria II GX	MRAM, M512

For information on ROM inference for Stratix, Arria, and Cyclone, see *Altera ROMs*, on page 1896.

Copyright © 2013 Synopsys, Inc.

Stratix V Byte-Wide Write Enable RAM

The synthesis tool can infer byte-wide write enable RAM for single-port and simple dual-port RAM. The tool maps the byte-wide write enable RAM to the Altera stratixv_ram_block or MLABs for improved performance by merging RAM and reducing its area. For configurations using address registers or when the RAM is too large for MLABs, the tool maps to the stratixv_ram_block. However, you can also use the attributes /* synthesis syn_ramstyle = "block_ram" */ or /* synthesis syn_ramstyle = "MLAB" */ to help the tool guide your preferences. See *syn_ramstyle*, on page 1467 for more information.

For coding examples, see:

- Example1: Simple Dual-port RAM
- Example 2: Simple Dual-port RAM (SystemVerilog Compatible)
- Example 3: Single-port Write First RAM

Limitations

Note the following limitations:

- Simple dual-port RAM can pack up to a maximum of 5 bytes of data.
- The RTL memory can have any configuration of data size up to a maximum of 5 bytes; however, read and write data size must be a multiple of 8-bit bytes. Thus the merged block RAM can only have up to 5 bytes of data width.

Note: True dual-port RAM with byte-wide write enable is not supported.

Example1: Simple Dual-port RAM

A Verilog and VHDL example for simple dual-port RAM are provided:

- Example 1A: Simple Dual-port RAM (Verilog)
- Example 1B: Simple Dual-port RAM (VHDL)

Example 1A: Simple Dual-port RAM (Verilog)

The Verlog code in this example contains a 2-dimensional RAM with byte-wide write enable.

Copyright © 2013 Synopsys, Inc.

```
module test (clk,we,waddr,raddr, be,wdata,q);
input clk,we;
input [5:0] waddr,raddr;
input [3:0] be;
input [31:0] wdata;
output reg [31:0] q;

reg [31:0] mem [63:0]
   /* xsynthesis xsyn_ramstyle = "xblock_ram, xno_rw_check" */;

//reg [5:0] waddr_reg;

always @(posedge clk) //write process
begin

//waddr_reg <= waddr;
   if (we)
   begin
      if (be[0])
      mem[waddr][7:0] <= wdata[7:0];
      if (be[1])
      mem[waddr][15:8] <= wdata[15:8];
      if (be[2])
      mem[waddr][23:16] <= wdata[23:16];
      if (be[3])
      mem[waddr][31:24] <= wdata[31:24];
   end
      q[31 : 0] <= mem[raddr];
end
endmodule
```

Example 1B: Simple Dual-port RAM (VHDL)

The VHDL code in this example contains a 2-dimensional RAM with byte-wide write enable.

```
-- Simple Dual-Port RAM with different read/write addresses and
   --single read/write clock
-- and with a control for writing single bytes into the memory
   --word; byte enable

library ieee;
use ieee.std_logic_1164.all;
library work;

entity byte_enabled_simple_dual_port_ram is
```


Copyright © 2013 Synopsys, Inc.

```
   generic (
      ADDR_WIDTH : natural := 10;
      BYTE_WIDTH : natural := 8;
      BYTES : natural := 4);

   port (
      we, clk : in  std_logic;
      be      : in  std_logic_vector (BYTES - 1 downto 0);
      wdata   : in  std_logic_vector(BYTE_WIDTH - 1 downto 0);
      waddr   : in  integer range 0 to 2 ** ADDR_WIDTH -1 ;
      raddr   : in  integer range 0 to 2 ** ADDR_WIDTH - 1;
      q       : out std_logic_vector(BYTES*BYTE_WIDTH-1 downto 0));
end byte_enabled_simple_dual_port_ram;

architecture rtl of byte_enabled_simple_dual_port_ram is
--Here is the two dimensional RAM array of a third dimensional
--"BYTE_WIDTH data
   --  build up 2D array to hold the memory
   type word_t is array (0 to BYTES-1) of
std_logic_vector(BYTE_WIDTH-1 downto 0);
   type ram_t is array (0 to 2 ** ADDR_WIDTH - 1) of word_t;
   -- delcare the RAM
   signal ram : ram_t;
   signal q_local : word_t;

begin  -- rtl
   -- Re-organize the read data from the RAM to match the output
   unpack: for i in 0 to BYTES - 1 generate
      q(BYTE_WIDTH*(i+1) - 1 downto BYTE_WIDTH*i) <= q_local(i);
   end generate unpack;

   process(clk)
   begin
      if(rising_edge(clk)) then
         if(we = '1') then
            if(be(0) = '1') then
               ram(waddr)(0) <= wdata;
            end if;
            if be(1) = '1' then
               ram(waddr)(1) <= wdata;
            end if;
            if be(2) = '1' then
               ram(waddr)(2) <= wdata;
            end if;
            if be(3) = '1' then
```

```
                    ram(waddr)(3) <= wdata;
                 end if;
              end if;
              q_local <= ram(raddr);
           end if;
        end process;
     end rtl;
```

For the examples above, the following single-port RAM are shown in the HDL Analyst RTL view.

For the examples above, the RTL byte-wide write enable RAM are shown mapped to the Altera stratixv_ram_block component in the Technology view. The two stratixv_ram_block blocks are generated. These blocks were filtered and the

Copyright © 2013 Synopsys, Inc.

portabyteenamask for the blocks were expanded to register/ports. This view shows how the byte-enable masks are connected. Since the VHDL version of the simple dual-port RAM example increased the address width from 6 to 10, two RAM blocks are needed, whereas for the Verilog version only a single RAM block is necessary.

Example 2: Simple Dual-port RAM (SystemVerilog Compatible)

The following SystemVerilog code example is equivalent to Example 1 above, except that it contains a 3-dimensional RAM with byte-wide write enable. For 3-dimensional RAM, make sure to enable the System Verilog switch on the Verilog tab of the Implementation Options panel.

```
module ram_infer_altera
(
   input we, clk,
   input [5:0] waddr, raddr, // address width = 6
   input [3:0] be, // 4 bytes per word
   input [31:0] wdata, // byte width = 8, 4 bytes per word
   output reg [31:0] q // byte width = 8, 4 bytes per word
);
// use a multi-dimensional packed array
// to model individual bytes within the word

reg [3:0][7:0] ram[0:63];// # words = 1 << address width
always@(posedge clk)
begin
   if(we)
      begin
         if(be[0]) ram[waddr][0] <= wdata[7:0];
         if(be[1]) ram[waddr][1] <= wdata[15:8];
         if(be[2]) ram[waddr][2] <= wdata[23:16];
         if(be[3]) ram[waddr][3] <= wdata[31:24];
      end
   q <= ram[raddr];
end
endmodule
```

For the example above, the RTL byte-wide write enable RAM are shown mapped to the Altera stratixv_ram_ram component in the Technology view. This block was filtered and the portabyteenamask was expanded to register/ports. This view highlights the fact that the byte-enable masks are connected to a single RAM block.

Copyright © 2013 Synopsys, Inc.

Example 3: Single-port Write First RAM

A Verilog and VHDL code example is provided that contains a single-port write first RAM with the byte enables written in the RTL code, for which the active byte enables solely determine the read activity of the RAM. For this type of RAM, you must register the byte enables for read in the RTL code. This example uses Altera "new_data_no_nbe_read" mode, which means that when the byte enables are not active no read occurs from the memory. Instead, whatever was last read, independent of the current memory address is read out.

- Example 3A: Single-port Write First RAM (Verilog)
- Example 3B: Single-port Write First RAM (VHDL)

Example 3A: Single-port Write First RAM (Verilog)

The Verilog code in this example contains a single-port write first RAM with byte enables.

```
module test (clk,we,address, be,wdata,q);
input clk,we;
input [8:0] address;
input [3:0] be;
input [31:0] wdata;
output reg [31:0] q;

reg [31:0] mem [511:0]
   /* synthesis syn_ramstyle = "block_ram, no_rw_check" */;

reg [8:0] address_reg;
reg [3:0] be_reg;

always @(posedge clk) //write process
begin

   if (we)
   begin
      if (be[0])
      mem[address][7:0] <= wdata[7:0];
      if (be[1])
      mem[address][15:8] <= wdata[15:8];
      if (be[2])
      mem[address][23:16] <= wdata[23:16];
      if (be[3])
      mem[address][31:24] <= wdata[31:24];
   end
end

wire [31:0] mem_data_out;
assign mem_data_out = mem[address_reg];

always @(posedge clk) //read process
begin
address_reg <= address;
be_reg <= be;
   if (be_reg[0] )
      q[7 : 0] <= mem_data_out[7 : 0];

   if (be_reg[1] )
      q[15 : 8] <= mem_data_out[15 : 8];

   if (be_reg[2] )
      q[23 : 16] <= mem_data_out[23 : 16];

   if (be_reg[3] )
```


Copyright © 2013 Synopsys, Inc.

```
        q[31 : 24] <= mem_data_out[31 : 24];

end //end of always

endmodule
```

Example 3B: Single-port Write First RAM (VHDL)

The VHDL code in this example contains a single-port write first RAM with byte enables.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test is
   port (
      clock : in std_logic;
      address : in std_logic_vector (8 downto 0);
      byteena : in std_logic_vector (3 downto 0);
      data : in std_logic_vector (31 downto 0);
      q : out std_logic_vector (31 downto 0)
      );
end test;

architecture rtl of test is
   type memory is array(512 downto 0) of std_logic_vector
      (31 downto 0);
   signal mem : memory := (others => (others => '0'));
   signal address_reg : std_logic_vector (8 downto 0);
   signal byteena_reg : std_logic_vector (3 downto 0);

attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is
   "block_ram, no_rw_check";

begin
   mem_write : process (clock)
   begin
      if (clock'event and clock = '1') then

   -- Write into memory when byte enabled
         if (byteena(0) = '1') then
            mem(conv_integer(address))(7 downto 0)
               <= data(7 downto 0);
```

```
            end if;
            if (byteena(1) = '1') then
               mem(conv_integer(address))(15 downto 8)
                  <= data(15 downto 8);
            end if;
            if (byteena(2) = '1') then
               mem(conv_integer(address))(23 downto 16)
                  <= data(23 downto 16);
            end if;
            if (byteena(3) = '1') then
               mem(conv_integer(address))(31 downto 24)
                  <= data(31 downto 24);
            end if;
         end if;
      end process mem_write;

   mem_read : process (clock)
      begin
         if (clock'event and clock = '1') then
         --Read output is updated only when byte enable is ON
            address_reg <= address;
            byteena_reg <= byteena;
            if (byteena_reg(0) = '1') then
               q(7 downto 0) <= mem(conv_integer(address_reg))
                  (7 downto 0);
               end if;
               if (byteena_reg(1) = '1') then
                  q(15 downto 8) <= mem(conv_integer(address_reg))
                  (15 downto 8);
               end if;
               if (byteena_reg (2) = '1') then
                  q(23 downto 16) <= mem(conv_integer(address_reg))
                  (23 downto 16);
               end if;
               if (byteena_reg (3) = '1') then
                  q(31 downto 24) <= mem(conv_integer(address_reg))
                  (31 downto 24);
               end if;
         end if;
      end process mem_read;

   end architecture rtl;
```

For the examples above, the following HDL Analyst RTL view shows that the byte enables for read are registered for the single-port write first RAM.

Copyright © 2013 Synopsys, Inc.

In the Technology view below, this portion of the design shows an example of Altera new_data_no_nbe_read mode for byte-wide write enable RAM. When the byte enables are inactive the read does not occur in memory; instead, whatever was last read for that byte enable is output.

Stratix III and Stratix IV RAMs with Control Signals

The enable pins on Altera Stratix III and Stratix IV RAM blocks provide built-in functionality for control signals. You can set up your Verilog VHDL code so that the synthesis tool extracts control signals for single-port and simple dual-port RAMs with a single common clock, and maps the logic to Stratix III and Stratix IV LutRAMs instead of registers. This avoids inferring extra logic, and improves resource utilization and quality of results (QoR). The control signals can be read enables, write enables and clock enables. See *Verilog Coding for StratixIII and StratixIV RAMs with Control Signals*, on page 724 for details.

Copyright © 2013 Synopsys, Inc.

Cyclone III RAMs with Control Signals

You can extract the control signals and take advantage of the built-in functionality for control signals provided with Cyclone III RAMs. The synthesis tool extracts control signals for single-port, simple dual-port and true dual-port RAMs and maps the logic to Cyclone III dedicated RAMs instead of registers.

See *Verilog Coding for Cyclone III RAMs with Control Signals*, on page 730 for details and examples.

You can specify the inference of individual RAMs as registers and logic, by setting the syn_ramstyle attribute (Verilog or VHDL) to registers. You can also use this attribute to map the RAM to particular resources. For syntax details, see *syn_ramstyle*, on page 1467.

Altera ROMs

ROMs that would normally be implemented as logic can be implemented as RAMs. See *syn_romstyle (Altera)*, on page 1502 for syntax details.

Copyright © 2013 Synopsys, Inc.

The following table shows how ROMs are inferred and implemented in different Altera technologies.

Technology	Synthesis infers...	Implementation
Stratix Stratix GX Stratix II HardCopy II Stratix II GX Stratix III HardCopy III Stratix IV HardCopy IV Stratix V Cyclone Cyclone II Cyclone III Cyclone IV E Cyclone IV GX Cyclone V Arria GX Arria II GX Arria II GZ Arria V	*technology*_ram_block RAM primitive (for synchronous ROMs only).	RAM. A synchronous ROM can have any of the following configurations: • Only read address registered • Only ROM output registered • Both read address and ROM output registered Each register can optionally have an asynchronous clear (aclr) and/or clock enable (clken) control signal. When the read address register has an aclr, the aclr is moved to the output of the ROM, because in these technologies a RAM core does not allow an asynchronous reset of the read address. When only the ROM output is registered, the synthesis tool automatically retimes the output register of the ROM to its address inputs during the mapping phase.
Stratix and Cyclone or newer families	M9K, M20K, or M144K	RAM. Asynchronous ROM can be mapped to RAM resources.
Stratix III and newer families	MLAB	RAM. Asynchronous ROM are mapped to MLAB resources.

From your HDL source code, the synthesis tool automatically infers ROMs and maps them to RAM primitives. The automatic mapping only occurs if the ROM size or address meets the minimum specifications. If the specifications are not met, the ROM is implemented in logic.

Copyright © 2013 Synopsys, Inc.

Altera LPMs

The Altera Library of Parameterized Modules (LPMs) contains technology-independent logic functions parameterized for scalability and adaptability. The library is derived from LPM version 2.2.0, which offers architecture-independent design entry for all Quartus II-supported devices. For step-by-step procedures, see *Working with LPMs*, on page 411 in the *User Guide*.

There are three methods for instantiating Altera LPMs:

- As black boxes (Verilog or VHDL). Start by defining the LPM using the Altera MegaWizard megafunction coding style. For synthesis, set the LPM as a black box and instantiate it in your design. Supply the original LPM megafunction file along with the `vqm` file as input for the place-and-route tool. See *Instantiating Altera LPMs as Black Boxes*, on page 412 in the *User Guide* for details.
- Using the Verilog LPM library provided. See *Instantiating Altera LPMs Using a Verilog Library*, on page 418 in the *User Guide* for a procedure.
- Using the provided VHDL prepared components that are provided (see *Prepared LPM Components Provided by Synopsys (VHDL)*, on page 1898 for a list). Prepared LPM Components use generics instead of attributes to specify different design parameters. You instantiate the components and assign (map) the ports and the values for the generics. Refer to the Altera place-and-route documentation for ports and generics that require mapping. See *Instantiating Altera LPMs Using VHDL Prepared Components*, on page 416 in the *User Guide* for a detailed procedure.

Prepared LPM Components Provided by Synopsys (VHDL)

The following VHDL prepared LPM components are provided by Synopsys. Their VHDL source code definitions are in the *install_dir*/lib/vhd/lpm.vhd file.

altcam_syn	altcdr_rx	altcdr_tx	altclklock
altclklock_syn	altddio_bidir	altddio_in	altddio_out
altdpram	altlvds_rx	altlvds_tx	altqpram
csfifo	dcfifo	lpm_abs	lpm_add_sub
lpm_and	lpm_bustri	lpm_clshift	lpm_compare

Copyright © 2013 Synopsys, Inc.

lpm_components	lpm_constant	lpm_counter	lpm_decode
lpm_divide	lpm_ff	lpm_fifo	lpm_fifo_dc
lpm_inv	lpm_latch	lpm_mult	lpm_mux
lpm_ram_dp	lpm_ram_dq	lpm_ram_io	lpm_or
lpm_rom	lpm_shiftreg	lpm_xor	scfifo

Altera DDR I/O Inference

The synthesis software maps Altera DDR register logic to the altddio_in and altddio_out megafunction. DDR registers can capture and/or send data at twice the rate of the clock or data strobe to a memory device or other high-speed interface application, for which the data is clocked at both edges of the clock. The DDR registers interface with DDR SDRAM, DDR2 SDRAM, RLDRAM II, QDR SRAM, and QDRII SRAM memory devices. Also, you can use the DDR I/O registers as a SERDES bypass mechanism in LVDS applications.

Example 1: altddio_in

The altddio_in megafunction only infers DDR registers with asynchronous control signals. This megafunction is inferred when there is a latch on the output of the negedge register. Note, the altddio_in megafunction does not infer registers with synchronous control signals, clock enable, or inverted clocks.

```
`timescale 100 ps/100 ps
`ifdef SYNTHESIS
module test(in1, clk,q);
`else
module test_RTL(in1, clk,q);
`endif

input  in1;

input  clk;
output q;

reg  q0,q1;
reg  l_r;

always @(posedge clk)
begin
```


Copyright © 2013 Synopsys, Inc.

```
      q0 <= in1;
   end
   always @(negedge clk)
   begin
      l_r <= in1;
   end

   always@(clk or l_r)
   if(clk)
      q1 <= l_r;

   assign q = q0 & q1;

   endmodule
```

The RTL and Technology views below show how altddio_in inferencing is implemented after synthesis.

Example 2: altddio_out

The altddio_out megafunction infers DDR registers for most combinations of asynchronous and synchronous control signals. This megafunction is inferred when both the registers are triggered from the posedge of the clock.

```
--#######################################################
-- The test case infers altddio_out instance for Altera.
--#######################################################
--Control signals used:
--#######################################################
-- sync rst
--#######################################################

library ieee;
use ieee.std_logic_1164.all;
entity test2 is
port
   (clk : in std_logic;
   rst : in std_logic;
   q0 : out std_logic;
   d1 : in std_logic;
   d2 : in std_logic);
end test2;
architecture behavioral of test2 is
signal d1_i_0,d2_i_0 : std_logic;

begin
   alt0 : process (clk,d1,d2,rst)
begin

-----without any control--------------
   if (clk'event and clk='1') then
      if rst ='1' then
         d1_i_0 <= '0';
      else
         d1_i_0 <= d1;
      end if ;
   end if;

   if (clk'event and clk='1') then
      if rst = '1' then
         d2_i_0 <= '0';
      else
         d2_i_0 <= d2;
      end if ;
   end if;
```


Copyright © 2013 Synopsys, Inc.

```
end process;

q0 <= d1_i_0 when clk ='1'  else d2_i_0;
--------------------------------------
end behavioral;
```

The RTL and Technology views below show how altddio_out inferencing is implemented after synthesis.

Copyright © 2013 Synopsys, Inc.

DSP Blocks for Stratix, Arria, and Cyclone

Altera Stratix, Arria, and Cyclone families of devices have dedicated DSP blocks that can be configured as multipliers, multiplier adders or multiplier accumulators. The DSP blocks are specified in a `vqm` netlist file as the Altera mega functions altmult_add, altmult_accum, and lpm_mult. Structural VHDL and Verilog simulation netlists also contain these functions.

The synthesis tools automatically recognize and extract these functions, then pass them to Quartus II, which maps them directly to the dedicated DSP blocks. This section describes the following:

- DSP Block Inference, on page 1904
- Attributes and Directives for DSP Block Inference, on page 1907
- Synchronous and Asynchronous Reset Packing for DSP Blocks, on page 1908
- Formal Verification with DSP Blocks, on page 1909

DSP Block Inference

The following table summarizes when DSP blocks are inferred.

Configuration	The synthesis tool implements...
Multiply-only mode	
A standalone signed or unsigned multiplier: `x = a*b;` It can be one of the following: • 36x36-bit multiplier (36-bit inputs, at most) • 18x18-bit multiplier (18-bit inputs, at most) • 9x9-bit multiplier (9-bit inputs, at most)	The lpm_mult megafunction

Copyright © 2013 Synopsys, Inc.

Configuration	The synthesis tool implements...
Multiply-add mode without input shift register	

2, 3, or 4 multipliers (all signed or all unsigned) feeding an adder. The outputs of the second and/or fourth multipliers can be negated.
`x = a*b + c*d;`
`x = a*b - c*d;`
`x = a*b + c*d + e*f;`
`x = a*b - c*d + e*f;`
`x = a*b + c*d - e*f;`
`x = a*b - c*d - e*f;`
`x = (a*b + c*d) + (e*f + g*h);`
`x = (a*b - c*d) + (e*f + g*h);`
`x = (a*b + c*d) + (e*f - g*h);`
`x = (a*b - c*d) + (e*f - g*h);`
	The altmult_add megafunction.
	The multiplier inputs and outputs can be registered. The output of the final adder can also be registered. The registers can have an asynchronous reset and/or enable. A maximum of four clocks (with or without enables) and four asynchronous resets can be used in an altmult_add to control the registers.
	The width of each multiplier bus is rounded up to the next highest multiple of 9 bits, and the larger of the two widths is taken as the size of the multiplier. Unused most significant bits (MSBs) are automatically sign extended.

Multiply-add with non-multiplier input. For example:
`a*b + c`
`a*b + c*d + e`
`a*b + c + d`
`a + b*c`
`a*b + c + d + e*f`
	Transformed and packed into a multiply-add DSP block by multiplying the input variable by 1.
	`a*b + c*1`
	`a*b + c*d + e*1`
	`a*b + c*1 + d*1`
	`a*1 + b*c`
	`a*b + c*1 + d*1 + e*f`

Copyright © 2013 Synopsys, Inc.

Configuration	The synthesis tool implements...
Multiply-add mode with input shift register	
Scan chains are formed in one of these ways: • Plain multiplier input registers • Input registers followed by adder • Scan chain formed by the input register of an accumulator. A DSP block has a *scan chain* when a multiplier register is driven by the output of the input register of another multiplier. The start of a scan chain is an input register driven by external signals. The final multiplier input registers can have scan-outs, which can be used to drive only another multiplier input register that can go into another DSP block.	The altmult_add megafunction when possible. It implements the altmult_add megafunction for scan chains formed by the input registers of a multiplier, regardless of whether it is followed by an adder. The tool implements the altmult_accum megafunction for scan chains formed by the input registers of an accumulator.
Multiply-accumulate mode with synchronous load	
	The altmult_accum megafunction whenever possible. The accumulator is loaded with the output of a multiplier without first clearing the accumulator. The synchronous load data must come from the same source feeding the adder. The source must be a non-negated multiplier; it can be signed or unsigned, registered or unregistered.

Copyright © 2013 Synopsys, Inc.

Configuration	The synthesis tool implements...
Multiply-accumulate mode without sload_mux	

1 multiplier driving an accumulator
`x = x + a*b;`
`x = x - a*b;`
	The altmult_accum megafunction.
	The multiplier can be signed or unsigned.
	The multiplier input and outputs can be registered. The accumulator register can have an asynchronous reset and/or enable.
	The accumulator width can be between 4 and 52; the multiplier width must be between 4 and 18.
	The width of each multiplier bus is rounded up to the next highest multiple of 9 bits, and the larger of the two widths is taken as the size of the multiplier. Unused most significant bits (MSBs) will automatically be set to zero, sign extended.

Attributes and Directives for DSP Block Inference

DSP block inference is influenced by the following attributes and directives:

syn_useioff syn_preserve	Registers with either of these attributes are not packed into MACs.
syn_keep	A syn_keep directive between the adder and the multiplier prevents inference of multipliers altmult_add and altmult_accum, but not plain multipliers.
syn_multstyle (Stratix only)	This attribute is used to choose between an LPM implementation and logic. The default implementation uses dedicated multiplier blocks (MAC); you specify logic or lpm_mult for alternative implementations. A multiplier with syn_multstyle attribute value of logic is mapped to logic, not packed into a MAC.

Copyright © 2013 Synopsys, Inc.

Synchronous and Asynchronous Reset Packing for DSP Blocks

The software packs asynchronous and synchronous resets into the Altera DSP block. The following code is an example of packing asynchronous resets.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity rst_sync is
port(
   clk,rst : in std_logic;
   Port_In : in std_logic_vector(15 downto 0);
   Port_In1 : in std_logic_vector(15 downto 0);
   Port_In2 : in std_logic_vector(15 downto 0);
   Port_In3 : in std_logic_vector(15 downto 0);
   Port_Out   : out std_logic_vector(31 downto 0) );
end rst_sync;

architecture rtl of rst_sync is
signal dly,dly1,dly2,dly3 : std_logic_vector(15 downto 0);
signal mult1, mult2 : std_logic_vector(31 downto 0);
signal dly_mult1, dly_mult2 : std_logic_vector(31 downto 0);
signal sum : std_logic_vector(31 downto 0);

begin
   process(clk)
   begin
      if(rising_edge(clk)) then
         if (rst='1') then
            dly  <= (others => '0');
            dly1 <= (others => '0');
            dly2 <= (others => '0');
            dly3 <= (others => '0');
         else
            dly  <= Port_In;
            dly1 <= Port_In1;
            dly2 <= Port_In2;
            dly3 <= Port_In3;
         end if;
      end if;
   end process;

mult1 <= dly * dly1;
mult2 <= dly2 * dly3;
```

```
   process(clk)
   begin
      if(rising_edge(clk)) then
         if (rst='1') then
            dly_mult1  <= (others => '0');
            dly_mult2 <= (others => '0');
         else
            dly_mult1  <= mult1;
            dly_mult2 <= mult2;
         end if;
      end if;
   end process;

sum <= dly_mult1 + dly_mult2;

   process(clk)
   begin
      if(rising_edge(clk)) then
         if (rst='1') then
            Port_Out  <= (others => '0');
         else
            Port_Out  <= sum;
         end if;
      end if;
   end process;
end rtl;
```

Formal Verification with DSP Blocks

When formal verification is enabled, the DSP block maps to LPM_MULT. See *Altera Optimizations*, on page 1910 for more information.

Copyright © 2013 Synopsys, Inc.

Altera Optimizations

This section describes optimizations that are specific to Altera.

- Core Voltage, on page 1910
- Companion Parts, on page 1910
- Fanout Limits in Altera Designs, on page 1911
- I/O Insertion in Altera Designs, on page 1912
- Pipelining in Altera Designs, on page 1913
- Retiming in Altera Designs, on page 1913
- Altera Gated and Generated Clocks, on page 1913
- Sequential Optimizations in Altera Designs, on page 1914
- Verification Mode in Altera Designs, on page 1914
- Compile Point Remapping in Altera Designs, on page 1915

For more information on generating a verification (`vif`) file using the synthesis tool, see *Overview of the VIF Flow*, on page 893 in the *User Guide*.

Core Voltage

Certain Stratix III parts
You can specify a core voltage value for some parts. See *Specifying Core Voltage in Stratix III Designs*, on page 766 in the *User Guide* for details.

Companion Parts

Stratix II, Stratix III, and Stratix IV; HardCopy II, HardCopy III, or HardCopy IV
You can specify an associated HardCopy companion part for a Stratix part, and vice versa. See *Specifying HardCopy and Stratix Companion Parts*, on page 765 in the *User Guide* for information on specifying companion parts.

Copyright © 2013 Synopsys, Inc.

Fanout Limits in Altera Designs

Large fanouts can cause large delays and routability problems. The synthesis tool maintains reasonable fanouts automatically, and also allows you to specify maximum fanouts. There are two ways to specify fanout limits: the Fanout Guide option (*Fanout Guide Option*, on page 1911), and the syn_maxfan attribute (*The syn_maxfan attribute for Altera Designs*, on page 1911).

The software first reduces fanout by replicating the driver of the high fanout net and splitting the net into segments. This replication can affect the number of register bits and ATOMs in your design. If replication is not possible, the signals are buffered. Buffering increases intrinsic delay and consumes more resources, and is therefore not used until a slightly higher fanout limit has been reached than is specified. You might want to instantiate global buffers for clocks with high fanouts to save resources.

Click View Log to display the net buffering report in the log file. The report shows how many nets were buffered or had their sources replicated, and the number of segments created for the net.

Fanout Guide Option

To access this option, select Project->Implementation Options->Device. Specify an integer value for Fanout Guide. During technology mapping, the synthesis tool tries to keep the fanout to less than the specified fanout guide. However, this value is only a *guideline*, not a hard limit. This means that the synthesis tool might not always respect it, if the limit imposes constraints that interfere with optimization.

The syn_maxfan attribute for Altera Designs

The syn_maxfan attribute controls the maximum fanout of an instance, net, or port. The limit specified by this attribute is treated as a hard or soft limit depending on where you specify it. The following rules determine how the tool treats the fanout limit

- A global fanout limit serves as a soft limit (or guide) and may not be honored if the limit degrades performance. You set a global fanout limit either by setting it as an implementation option (Fanout Guide) or by specifying the syn_maxfan attribute on a top-level module or view.
- A syn_maxfan attribute applied on a module or view serves as a soft limit for the scope of the module. The module fanout limit overrides the global fanout guide limit for the scope of the module.

Copyright © 2013 Synopsys, Inc.

- When you specify a syn_maxfan attribute on an instance that is not a primitive as inferred by the compiler, the limit serves as a soft limit and can be propagated down the hierarchy.
- When you specify a syn_maxfan attribute on a port, net, or register (or any primitive instance), the limit is considered a hard limit. Note that the syn_maxfan attribute does not prevent the instance from being optimized away. Any design rule violations that result from buffering or replication are the responsibility of the user.

Buffering vs Replication

The fanout of an instance or a net can be reduced by either replicating the driver of the net that violates the maxfan limit or by creating a buffer tree. Replication or buffering depends on where the syn_maxfan attribute is specified and also on other attributes such as syn_replicate. The set of rules given below describes when the software replicates or creates a buffer tree.

- When a syn_maxfan attribute is set on a port or on a net driven by a port (or an I/O pad), a buffer tree is always created to honor the maxfan limit.
- When the value of syn_replicate is 0, the software creates a buffer tree. Note that the syn_replicate attribute must be used in conjunction with the syn_maxfan attribute; if not, syn_replicate has no meaning. The syn_replicate attribute is used only to turn off the replication.
- When a syn_keep or syn_preserve attribute is specified on a net driver, a buffer tree is created.
- When a net driver is not a primitive gate or register, a buffer tree is created. (only registers and ATOMs are replicated).

I/O Insertion in Altera Designs

I/O pads are automatically inserted into the design, unless you disable it. If you manually instantiate I/Os, I/Os are inserted only for the pins that need them.

If you do not want to insert any I/Os in the design, enable the Disable I/O Insertion check box (Project->Implementation Options->Device). This is useful for bottom-up compiles, because you can check the area your logic blocks use before you synthesize the whole design. If you disable I/O insertion, you will not get any I/Os in the design unless you manually instantiate them.

Copyright © 2013 Synopsys, Inc.

Register Packing in Altera Designs

When a register drives an output, or an input drives a register, you can pack the register into the I/O. You can use the syn_useioff attribute to control register packing. See *Packing I/O Cell Registers in Altera Designs*, on page 763 in the *User Guide* for the procedure.

Pipelining in Altera Designs

Pipelining allows designs with multipliers to operate at a faster frequency by moving registers after a multiplier into the multiplier. If you have suitable registers, the software can automatically pipeline multipliers.

You enable/disable pipelining with the Pipelining device mapping option (Project->Implementation Options->Device) or by setting the syn_pipeline attribute (syn_pipeline, on page 1442). See *Pipelining*, on page 426 in the *User Guide* for a step-by-step procedure on using pipelining.

Retiming in Altera Designs

Retiming is the process of automatically moving registers (register balancing) across combinational gates/LUTs to improve timing, while ensuring identical logic behavior.

You enable/disable global retiming with the Retiming device mapping option (Project->Implementation Options->Device). You can use the syn_allow_retiming attribute (*syn_allow_retiming*, on page 1216) to enable or disable retiming for individual flip-flops.

Pipelining is automatically enabled when retiming is enabled. Use the syn_pipeline attribute (*syn_pipeline, on page 1442*) to control pipelining. See *Retiming*, on page 430 in the *User Guide* for a procedure.

Altera Gated and Generated Clocks

The Clock Conversion option on the GCC (Synplify Pro) or GCC & Prototyping Tools tab (Synplify Premier), when enabled, attempts to remove the clock gating from synchronous logic and to convert generated clocks to the original clock with an enable.

Copyright © 2013 Synopsys, Inc.

Gated Clocks

Gated clocks are converted by removing the enable logic from the clock path. Extracting the enable logic allows clock constraints to be applied globally and eliminates broken paths. Converting gated clocks also makes use of the dedicated clock networks to simplify ASIC prototyping. For information on using the Clock Conversion option with gated clocks, see *Working with Gated Clocks, on page 466* in the *User Guide*.

Generated Clocks

Generated clocks are converted to a circuit with an enable to improve performance by reducing clock skew. For more information on using the Clock Conversion option with generated clocks, see *Optimizing Generated Clocks, on page 499* in the *User Guide*.

Sequential Optimizations in Altera Designs

The Disable Sequential Optimization option determines whether sequential optimizations are performed. By default, sequential optimizations are performed. Note that unused registers are always removed from the design regardless of this option setting. Disabling sequential optimizations (checking the Disable Sequential Optimizations box) can increase delay times and area requirements, and effectively disables the FSM Compiler. The Tcl equivalent for this option is -no_sequential_opt 1|0.

Verification Mode in Altera Designs

Verification mode determines whether optimizations are disabled so that the output is compatible with verification tools. This option is only supported in the Synplify Pro and Synplify Premier tools. You control verification mode with the Verification Mode option in the Device tab of the Implementation Options dialog box. Or you can use the equivalent Tcl command: set_option -verification_mode 1|0 .

Note that although they are similar, Verification Mode differs from the Disable Sequential Optimizations option. The Disable Sequential Optimizations option only disables sequential optimizations, but Verification Mode disables other FPGA optimizations as well.

Copyright © 2013 Synopsys, Inc.

The following describes the effects of setting the Verification Mode option:

Enabled	Ensures that synthesis output files are compatible with the verification tool by disabling various sequential optimizations that cannot be easily verified. It disables boundary optimizations and sequential optimizations, like the inference of resettable SRLs. When you enable Verification Mode, you must also disable the Retiming and Pipelining options and remove any assigned syn_pipeline and syn_allow_retiming attributes. Enabling this mode also enables the Write Verification Interface Format (VIF) File option, which generates a `vif` file. This file allows you to formally verify the equivalence of your input RTL description with the post-synthesis netlist. Enable this option only when this verification is needed. For more information on generating a verification (`vif`) file using the synthesis tool, see *VIF Formal Verification Flows*, on page 893.
Disabled	Produces synthesis output that performs the standard default optimizations when verification mode is turned off.

Effect of Verification Mode on Multiplier Mapping

For Altera Stratix designs, the Verification Mode setting affects how multipliers are mapped. When this option is disabled, the multipliers are mapped to device-specific components:

Verification Mode	Multiplier Mapping
On	All mapped to altmult_add or altmult_accum
Off	Stratix - stratix_mac_mult and stratix_mac_out Stratix GX - stratixgx_mac_mult and stratixgx_mac_out Stratix II - stratixii_mac_mult and stratixii_mac_out Stratix II GX - stratixiigx_mac_mult and stratixiigx_mac_out Stratix III - stratixiii_mac_mult and stratixiii_mac_out Stratix IV - stratixiv_mac_mult and stratixiv_mac_out

Compile Point Remapping in Altera Designs

Compile points are smaller synthesis units, which let you break up a larger design into smaller units and do incremental synthesis. See *Synthesizing Compile Points*, on page 1109 and *Compile Point Basics*, on page 1088 for information about the flow and compile points.

Copyright © 2013 Synopsys, Inc.

The Update Compile Point Timing Data option determines whether changes to a compile point force the remapping of its parents, taking into account the new timing model of the child. The term *child* refers to a compile point that is contained inside another; the term *parent* refers to the compile point that contains the child. These terms are thus not used here in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is the parent of all compile points.

The following table describes the settings for the Update Compile Point Timing Data option:

Disabled	(Default). Only compile points that have changed are remapped, and their remapping does *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used to map and optimize its parents. If the option is disabled and the *interface* of a locked compile point is changed, the immediate parent of the compile point must be changed accordingly. In this case, both are remapped, and the *updated* timing model of the child is used when remapping this parent.
Enabled	Compile point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account because the option was disabled. The timing model of a compile point is updated when either of the following is true: • The compile point is remapped, and the Update Compile Point Timing Data option is enabled. • The interface of the compile point is changed.

Copyright © 2013 Synopsys, Inc.

Altera Reports

Synopsys FPGA tools generate resource usage and timing reports. For the MAX family, the software only generates a resource usage report. To view these reports, click the View Log button in the Project view or select View ->View Log File from the menu. In addition to the standard reports described in this section, you can generate custom timing reports when using some devices – see *Altera Custom Timing Reports*, on page 1919. The Synplify tool does not support custom timing reports.

- Altera Resource Usage Reports, on page 1917
- Altera Net Buffering Reports, on page 1919
- Altera Custom Timing Reports, on page 1919

Altera Resource Usage Reports

The log file for Altera designs includes various reports: a resource usage report, a timing report, and a net buffering report. To view these synthesis reports, click View Log in the Project view. The resource usage reports include this information:

- Cell usage, including the following:
  - Carry chains, flip-flops, logic element ATOMs (*Stratix and Cyclone*)
  - LUTs and DSP blocks
- Memory cell usage

MRAM	Stratix, Stratix GX, Stratix II/HardCopy II, Stratix II GX, Arria GX, Arria II GX
M4K	Stratix, Stratix GX, Stratix II/HardCopy II, Stratix II GX, Cyclone, Cyclone II
M512	Stratix, Stratix GX, Stratix II/HardCopy II, StratixII GX, Arria GX, Arria II GX
M144K	Stratix III/HardCopy III, Stratix IV/HardCopy IV
M9K	Stratix III/HardCopy III, Stratix IV/HardCopy IV, Cyclone III, Cyclone IV E, Cyclone IV GX
M20K	Stratix V
M10K	Cyclone V, Arria V

Copyright © 2013 Synopsys, Inc.

- Number of I/Os
- Utilization percentage

See *Stratix II GX Resource Usage Report Example*, on page 1918 and *Cyclone III Resource Usage Report Example*, on page 1918 for examples of this report.

Stratix II GX Resource Usage Report Example

```
##### START OF AREA REPORT #####[</a>

Design view:work.eight_bit_uc(verilog)
Selecting part EP2SGX30CF780C3

Total combinational functions 213
ALUT usage by number of inputs
        7 input functions  5
        6 input functions  55
        5 input functions  36
        4 input functions  30
        [=3 input functions  87
ALUTs by mode
        normal mode            188
        extended LUT mode      5
        arithmetic mode        20
        shared arithmetic mode 0
Total registers 180
Total Estimated Packed ALMs 184
I/O pins 26

Total user instantiated Altsyncrams:      1
DSP Blocks:     0  (0 nine-bit DSP elements).
DSP Utilization: 0.00% of available 16 blocks (128 nine-bit).
ShiftTap:       0  (0 registers)
MRAM:           0  (0% of 1)
M4Ks:           1  (0% of 144)
M512s:          1  (0% of 202)
Total ESB:      1792 bits

##### END OF AREA REPORT #####]
```

Cyclone III Resource Usage Report Example

```
##### START OF AREA REPORT #####[</a>

Design view:work.eight_bit_uc(verilog)
Selecting part EP3C5F256C6
```

```
Total combinational functions 316
Logic element usage by number of inputs
        4 input functions  142
        3 input functions  124
        [=2 input functions  50
Logic elements by mode
        normal mode            295
        arithmetic mode        21

Total registers 179
I/O pins 26

Total user instantiated Altsyncrams:      1
DSP Blocks:     0  (0 nine-bit DSP elements).
DSP Utilization: 0.00% of available 23 blocks (46 nine-bit).
ShiftTap:       0  (0 registers)
Total ESB:      1792 bits

##### END OF AREA REPORT #####
```

Altera Net Buffering Reports

Click View Log to display the net buffering report in the log file. The report shows how many nets were buffered or had their sources replicated, and the number of segments created for the net. For an explanation of buffering and replication, see *Buffering vs Replication*, on page 1912.

Altera Custom Timing Reports

In addition to the default timing report, you can generate custom timing reports for some technologies. You must have a Synplify Premier or Synplify Pro license to generate custom timing reports. The default timing report is part of the log file (*projectName*.srr) that is located in the results directory. This report provides a clock summary, I/O timing summary, and detailed timing information about paths you specify.

The custom timing report lets you run targeted timing analysis. For example, you can specify start and end points of paths and generate a report for only those sections of the design. You generate a custom timing report with the Analysis->Timing Analyst command. See *Generating Custom Timing Reports with STA*, on page 729 in the *User Guide* for a procedure. The generated custom timing report is stored in a text file with a `ta` extension. In addition, the tool generates a corresponding output netlist file, which allows you to view just the targeted paths in a customized Technology view.

Copyright © 2013 Synopsys, Inc.

Forward-annotation to Altera Tools

The following describe forward-annotation to the Altera place-and-route tools:

- Forward-annotation Formats for Stratix II Constraints, on page 1921
- Forward-annotation of Altera Compile Point Constraints, on page 1922
- VQM Filenames for Place-and-route Tools, on page 1922

Forward-annotation for Altera Designs

The synthesis tools automatically generate constraint files for forward-annotation that are specific to the P&R tool you are using (`tcl` for Quartus II). If you want to disable this option, click Implementation Options, go to the Implementation Results panel, and disable Write Vendor Constraint File.

The tool forward-annotates constraints like delays and clock relationships.

define_clock	Includes clock constraints generated by auto constraining.
define_false_path	
define_input_delay define_output_delay	The synthesis tool forward-annotates the define_input_delay and define_output_delay constraints to Quartus II if you use the syn_forward_io_constraints attribute. A value of 1 enables forward-annotation (default behavior), while a value of 0 disables forward annotation. Use this attribute on the top level VHDL or Verilog file, or place it on the global object in the Attributes panel in the SCOPE spreadsheet. For details about the syntax for this attribute, see syn_forward_io_constraints, on page 1330.
define_multicycle_path	Quartus II only supports forward-annotation of multicycle -from and -to register constraints, but not -through constraints. The software forward-annotates a -from A constraint as a from A to * constraint. Bus constraints are forward-annotated in a bit-blasted format, which could result in a large number of constraints in the Tcl file. For -from -to constraints that are applied to the entire bus, you can use wildcards in square brackets to compress the number of constraints; for example, -from A[*] -to B[*]. If the brackets are removed, the software does not compress the constraints. Single-ended constraints (-from, -to) are not compressed.
define_path_delay	

Copyright © 2013 Synopsys, Inc.

In addition to these constraints, the synthesis tool forward-annotates relationships between different clocks. For PLLs, the software forward-annotates the inputs when you define them in the SCOPE window. The software only forward-annotates the PLL inputs, because the P&R tool does its own propagation.

Altera Max+Plus II Constraint Files

The constraint file generated for Altera Max+Plus II place-and-route tools has an .acf file extension. The following constraints are forward-annotated to Altera Max+Plus II in this file:

- define_clock
- define_input_delay
- define_output_delay

Forward-annotation Formats for Stratix II Constraints

Stratix II and Stratix II GX
You have a choice of constraint formats, depending on which timing analyzer you are targeting in the Quartus tool.

TimeQuest Timing Analyzer	`scf` constraints file
Classic Timing Analyzer	`tcl` constraints file

To specify the correct output format for the constraints to be forward-annotated, enable or disable the Use TimeQuest Timing Analyzer option on the Device tab of the Implementation Options dialog box or use the set_option -timequest 1|0 Tcl command.

In addition to these constraints, the synthesis tool forward annotates relationships between different clocks.

For PLLs, the software forward-annotates the inputs when you define them in the SCOPE window. For more information about this, see the *User Guide*. The software only forward-annotates the PLL inputs, because the back-end tool does its own propagation.

Copyright © 2013 Synopsys, Inc.

Forward-annotation of Altera Compile Point Constraints

Constraints applied to instances inside a compile point are forward-annotated. Constraints applied to the interface (ports and bit ports) of a compile point are not forward-annotated.

VQM Filenames for Place-and-route Tools

Altera requires that the name (without the filetype extension) of a VQM file be identical to the top-level design name inside the file. Make sure that your synthesis result file has the same name as your top-level Verilog module or VHDL entity.

The synthesis result filename defaults to your HDL source filename with the `edf` extension, so give the top-level module/entity the same name as the source file, but without the extension. If you do not have identical names, you get this error message:

"Can't find top level cell *filename*"

To keep track of the files generated by Quartus II, create a subdirectory for the VQM result files before running the place-and-route tool.

Integration with Altera Tools and Processes

The following describe how the synthesis tools support various tools and flows for Altera designs:

- Altera Place-and-route Tools, on page 1923
- Quartus II Incremental Compilation Flow, on page 1924
- Clearbox Support in Synplify Pro and Synplify Premier, on page 1924
- Greybox Support in Synplify Pro and Synplify Premier, on page 1926
- Stratix, Arria, and Cyclone Families, on page 1927

Altera Place-and-route Tools

The synthesis tools support many versions of the Altera place-and-route tools, and automatically generate the appropriate Altera library files for the tool and version you are using. The `vqm` format varies slightly from version to version, so you must specify the Quartus tool version and set the output format on the Implementation Results panel of the Implementation Options dialog box.

Copyright © 2013 Synopsys, Inc.

When you set these options, the synthesis tools automatically generate the appropriate vqm files for the version you are using for placement and routing. You can set P&R to run automatically after synthesis or launch the P&R UI from the Options menu. See *Working with Quartus II*, on page 769 and *Running P&R Automatically after Synthesis*, on page 842 of the *User Guide* for details.

Quartus II Incremental Compilation Flow

The Altera Quartus II Incremental Compilation feature preserves design data so that it can make incremental place-and-route updates when there are changes like HDL changes to a small number of modules, pin relocation, attribute changes, or timing constraint changes. The Synplify Pro and Synplify Premier tools support this Altera methodology by allowing you to create partitions called compile points in the synthesis design. You can then compare the previous and current implementation of a partition to determine if it needs to be updated, without redoing the rest of the design.

For information about how to run this flow, see the following sections in the *User Guide*:

- *Running Altera Quartus II Incrementally*, on page 846
- *Synthesizing Compile Points*, on page 1109

Clearbox Support in Synplify Pro and Synplify Premier

Most Quartus megafunctions do not come with timing information. Without this information, the synthesis tools cannot do any timing-driven optimizations at the megafunction boundary. Providing timing information for the Quartus megafunction library is difficult because most of the relevant megafunctions are highly parameterized. For several megafunctions, timing changes dramatically when a single parameter changes.

Typically the synthesis tools treat megafunctions without timing information as black boxes. This approach prevents the tool from doing boundary optimizations. For example, the tool might be able to move registers to increase the FMAX of a design, but the registers in a pipelined LPM_MULT are fixed.

Altera Clearbox models provide for the conversion of megafunctions to structural VHDL or Verilog. Using these models eliminates the need to black box the Quartus megafunctions. The Clearbox structural content is fully visible

Copyright © 2013 Synopsys, Inc.

and has known primitives like dffe or latch in addition to the WYSIWYG Clearbox primitives (see *Clearbox WYSIWYG Cells*, on page 1925). There are no undefined parameters, except for the ones remaining on the ATOMs. The synthesis tool uses the timing information from the model to perform boundary optimizations on the megafunctions.

Clearbox Flow Support

There are two ways in which the synthesis tools can get the Clearbox information and optimize timing:

- Read the information from a Clearbox netlist added to the design
  Support: All Altera technologies.
- For some supported Altera families the synthesis tools automatically generate the timing models from the Quartus Clearbox information.
  Synplify Pro Support: Arria GX, Arria II, Arria V, Cyclone V, Stratix II, Stratix III, Stratix IV, Stratix V.
  Synplify Premier Support: Stratix II, Stratix III, and Stratix V

  The tools can use the Clearbox information for optimization, and then include the architecture-specific primitives in the output netlist. For procedures on using Clearbox models, see *Implementing Megafunctions with Clearbox Models*, on page 1002 in the *User Guide*.

Clearbox WYSIWYG Cells

The following WYSIWYG primitives are generated by Clearbox tools. The synthesis tools obtain the information about the internals of the megafunctions and the timing from each input to output.

- lcell
- mac_mult
- mac_out
- ram_block

Copyright © 2013 Synopsys, Inc.

Greybox Support in Synplify Pro and Synplify Premier

The Synplify Pro and Synplify Premier tools support Altera grey box models. 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 Altera grey box timing models. The synthesis tools can use the resource information for optimizations, but does not write the underlying logic out to the output netlist. For details about implementing megafunctions using grey box timing models, see *Implementing Megafunctions with Grey Box Models*, on page 1012 in the *User Guide*.

Stratix, Arria, and Cyclone Families

This section provides guidelines for using the synthesis tool with Stratix (and corresponding HardCopy), Arria, and Cyclone devices.

- Stratix, Arria, and Cyclone Device Mapping Options, on page 1927
- Stratix, Arria, and Cyclone File Format Options, on page 1930
- Altera Attribute and Directive Summary, on page 1944

Stratix, Arria, and Cyclone Device Mapping Options

You can specify device mapping options in the Implementation Options dialog box or with the Tcl set_option command.

Device Mapping Options

This table alphabetically lists the options you can set on the Device tab:

Option	Description
Altera Models	Specifies if Clearbox models are used for instantiated and inferred Altera megafunctions. • on calls Clearbox for inferred and instantiated Megafunctions. • off does not call Clearbox for inferred and instantiated Megafunctions. • clearbox_only calls Clearbox for inferred Megafunctions only. For more information, see *Implementing Megafunctions with Clearbox Models*, on page 1002.
Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views and Design Planner, and use them to create collections using the Tcl Find command. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
Clock Conversion	Performs gated- and generated-clock optimizations when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

Option	Description
Companion Part	Lets you specify a HardCopy companion part. See *Specifying HardCopy and Stratix Companion Parts*, on page 765 in the *User Guide*.
Core Voltage	*Stratix III* Lets you specify a core voltage. See *Specifying Core Voltage in Stratix III Designs*, on page 766 in the *User Guide*.
Disable I/O Insertion	Enables or disables I/O insertion during synthesis. The default is false, so I/O ATOMs are inserted. See *I/O Insertion in Altera Designs*, on page 1912 for details.
Disable Sequential Optimizations	Enables or disables the sequential optimizations for the design. See *Sequential Optimizations in Altera Designs*, on page 1914.
Enhanced Optimization	In Synplify Premier logic synthesis mode, determines whether the tool runs additional optimizations. See *Logic Synthesis with Enhanced Optimization*, on page 40 in the *User Guide* for more information.
Fanout Guide	Guide for fanout-based optimizations which results in buffering or replication of the net driver. See *Setting Fanout Limits*, on page 444 in the *User Guide* for more information. The default is 500.
Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Compile Point Remapping in Altera Designs*, on page 1915 for details. The Synplify tool does not support compile points.

Copyright © 2013 Synopsys, Inc.

Option	Description
Use TimeQuest Timing Analyzer	*Stratix II and Stratix II GX* Determines which timing analyzer to use. Enable it to use the Quartus II TimeQuest Timing Analyzer, and disable it to use the Classic Timing Analyzer. See *Forward-annotation Formats for Stratix II Constraints*, on page 1921 for additional information.
Verification Mode	Determines whether synthesis output files are compatible with the verification tools. To use this feature, you must have a Synplify Pro or Synplify Premier license. • 1 Ensures that synthesis output files are compatible with the verification tools. • 0 When disabled, produces synthesis output that performs the standard default optimizations. See *Verification Mode in Altera Designs*, on page 1914.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Altera Architectures*, on page 1937.

set_option Device Options

This is the syntax for setting device options for Stratix, Cyclone, and Arria devices.

set_option -technology *keyword* **-part** *partName* **-part_companion** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | **0** or **-disable_io_insertion 1** | **0**
-enable_designware 1|**0**
-enhanced_optimization 1|**0**
-fix_gated_and_generated_clocks 1|**0**
-maxfan *value*
-no_sequential_opt 1|**0**
-resolve_multiple_driver 1|**0**
-run_prop_extract 1|**0**
-RWCheckOnRam 1|**0**

Copyright © 2013 Synopsys, Inc.

-syn_altera_model on|off|clearbox_only
-timequest 1|0
-update_models_cp 1|0
-verification_mode 1|0
-voltage *value*

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-automatic_compile_point 1|0
-fast_synthesis 1|0
-physical_synthesis 1|0
-pipe 1|0
-resource_sharing 1|0
-retiming 1|0
-symbolic_fsm_compiler 1|0 or **-autosm 1|0**
-use_fsm_explorer 1|0

Stratix, Arria, and Cyclone File Format Options

Use one of the following methods to set options for result file formats and design filenames:

User Interface	• Select Project->Implementation Options to display the Implementation Options dialog box. • Go to the Implementation Results panel and specify options for results files.
Tcl	Use the project Tcl command to specify the format and file name. See *project Command for Altera Architectures*, on page 1943 for details of the arguments.

Copyright © 2013 Synopsys, Inc.

MAX II and MAX V Families

This section provides guidelines for using the synthesis tool with MAX II and MAX V devices. See the following:

- Device Mapping Options for MAX II and MAX V, on page 1931
- MAX II and MAX V File Format Options, on page 1933
- Altera Attribute and Directive Summary, on page 1944

Device Mapping Options for MAX II and MAX V

You can specify device mapping options in the Implementation Options dialog box or with the Tcl set_option command.

Device Mapping Options

This table alphabetically lists the options available on the Device tab of the Implementation Options dialog box for MAX II and MAX V devices:

Option	Description
Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views and Design Planner, and use them to create collections using the Tcl Find command. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
Clock Conversion	Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details. The Clock Conversion option is only available with the *Synplify Pro* and *Synplify Premier* products.
Disable I/O Insertion	Enables or disables I/O insertion during synthesis. The default is false, so I/O ATOMs are inserted. See *I/O Insertion in Altera Designs*, on page 1912.

Copyright © 2013 Synopsys, Inc.

Option	Description
Disable Sequential Optimizations	Enables or disables the sequential optimizations for the design. See *Sequential Optimizations in Altera Designs*, on page 1914.
Fanout Guide	Guide for fanout-based optimizations which results in buffering or replication of the net driver. The default is 500. See *Setting Fanout Limits*, on page 444 in the *User Guide* for more information.
Map Logic	Maps logic directly to Altera ATOM primitives. See *Altera ATOM Primitives*, on page 1879 for more information. By default, this is enabled.
Perform Cliquing	Enables/disables the grouping of logic functions, for example all logic critical to speed. If enabled, the compiler keeps cliqued members together and forward-annotates them to Quartus II. The default value is (enabled). Does not apply to MAX II devices.
Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Update Compile Point Timing Data	Determines if changes to a locked compile point force remapping of its parents, taking into account the new timing model of the child. See *Compile Point Remapping in Altera Designs*, on page 1915, for an explanation. This feature is not available in the Synplify tool.

Copyright © 2013 Synopsys, Inc.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box.w *set_option Command for Altera Architectures*, on page 1937.

set_option Device Options

This is the syntax for setting device options for MAX II and MAX V devices.

set_option -technology *keyword* **-part** *partName* **-speed_grade** *value*
-package *packageName*
-block 1|0 or **-disable_io_insertion 1 | 0**
-clique 1 | 0 or **-cliquing 1 | 0**
-fix_gated_and_generated_clocks 1|0
-map_logic 1|0
-maxfan *value*
-no_sequential_opt 1|0
-resolve_multiple_driver 1|0
-run_prop_extract 1|0
-RWCheckOnRam 1|0
-update_models_cp 1|0

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-pipe 1|0
-resource_sharing 1|0
-retiming 1|0
-symbolic_fsm_compiler 1|0 or **-autosm 1|0**
-use_fsm_explorer 1|0

MAX II and MAX V File Format Options

Use one of the following methods to set options for result file formats and design filenames:

Copyright © 2013 Synopsys, Inc.

User Interface	• Select Project->Implementation Options to display the Implementation Options dialog box. • Go to the Implementation Results panel and specify options for results files.
Tcl	Use the project Tcl command to specify the format and file name. See *project Command for Altera Architectures*, on page 1943 for details of the arguments.

MAX Family

This section provides guidelines for using the synthesis tool with MAX devices. The topics include:

- Device Mapping Options for Altera MAX, on page 1935
- File Format Options for MAX, on page 1937
- Altera Attribute and Directive Summary, on page 1944

Device Mapping Options for Altera MAX

You can specify device mapping options in the Implementation Options dialog box or with the Tcl set_option command.

Device Mapping Options

The Altera MAX technologies are supported only by the Synplify and Synplify Pro software. The following table is an alphabetical list of the MAX options on the Device panel:

Option	Description
Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views and Design Planner, and use them to create collections using the Tcl Find command. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
Disable Sequential Optimizations	Enables or disables the sequential optimizations for the design. See *Sequential Optimizations in Altera Designs*, on page 1914.
Map logic	Controls mapping to Altera LCELLs (logic cells). When enabled, it maps logic directly to LCELLs, instead of to primitive gates.
Maximum Cell Fanin	Sets the maximum fanin for synthesis.
Percent of designs to optimize for timing	Specifies the percentage of nets to optimize during synthesis. See *Tips for Optimization*, on page 422 of the *User Guide*.
Soft LCELLs	Controls the insertion of soft buffers during synthesis.

Copyright © 2013 Synopsys, Inc.

Option	Description
Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

set_option Device Mapping Options

You can use the set_option command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Altera Architectures*, on page 1937.

set_option Device Options

This is the syntax for setting options for MAX devices. For descriptions of these Tcl options, see *set_option Command for Altera Architectures*, on page 1937.

set_option -technology *keyword* **-part** *partName* **-part_companion** *partName*
-speed_grade *value* **-package** *packageName*
-area_delay_percent *value*
-domap 1|0
-fanin_limit *value*
-map_logic 1|0
-no_sequential_opt 1|0
-resolve_multiple_driver 1|0
-run_prop_extract 1|0
-RWCheckOnRam 1|0
-soft_buffers 1|0

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
- **-pipe 1|0**
- **-resource_sharing 1|0**
- **-retiming 1|0**
- **-symbolic_fsm_compiler 1|0** or **-autosm 1|0**
- **-use_fsm_explorer 1|0**

File Format Options for MAX

Use one of the following methods to set options for result file formats and design filenames:

User Interface	• Select Project->Implementation Options to display the Implementation Options dialog box. • Go to the Implementation Results panel and specify options for results files.
Tcl	Use the project Tcl command to specify the format and file name. See *project Command for Altera Architectures*, on page 1943 for details of the arguments.

set_option Command for Altera Architectures

The set_option Tcl command offers an alternative way to set options that are available on the Device and Options tabs on the Implementation Options dialog box. You can save options set with this command in a file and re-execute them. For a complete list of all available set_option arguments, see *set_option*, on page 1679, or enter help set_option in a Tcl Script window

The following table describes the subset of options available for different Altera technologies. All options are not available for all technologies.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
Target Technology Options	
-technology *keyword*	Specifies the target technology. See *Technology Keywords*, on page 1942 for a list of valid keywords.
-part *partName*	Specifies a part for the implementation. Refer to Project->Implementation Options->Device or to the Altera documentation for available choices.
-part_companion *partName*	Specifies a HardCopy companion part package for certain technologies. Refer to Project -> Implementation Options->Device for available choices.
-speed_grade *value*	Sets the speed grade. Refer to Project-> Implementation Options->Device or to the Altera documentation for available choices.
-package *packageName*	Specifies the package. Refer to Project-> Implementation Options->Device or to the Altera documentation for available choices.
Optimization Options	
-area_delay_percent *value*	*MAX family* Specifies the percentage of nets to optimize during synthesis. Use only when map_logic is `true`. See *Tips for Optimization*, on page 422 of the *User Guide*. Default: 0
-automatic_compile_point 1|0	*Stratix, Arria, and Cylone families* Enables/disables automatic compile points, which can be mapped in parallel using multiprocessing. When enabled, the tool analyzes a design and automatically identifies modules that can be defined as compile points.
-block 1 | 0 **-disable_io_insertion 1 | 0**	*All families except MAX* Enables/disables I/O insertion during synthesis. See *I/O Insertion in Altera Designs*, on page 1912. Default: 0 (Inserts I/Os)

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-clique 1 | 0 **-cliquing 1 | 0**	*MAX II and MAX V families* Enables/disables the grouping of logic functions, for example all logic critical to speed. If enabled, the compiler keeps cliqued members together and forward-annotates them to Quartus II. Default: 1 (enabled)
-domap 1 | 0	*MAX family* Turns on direct mapping to LCELLs during synthesis. Default: 1 (enabled).
-enhanced_optimization 1|0	*Synplify Premier* *Stratix, Arria, and Cylone families* Enables/disables additional optimizations during logic synthesis to get better QoR. When disabled, the tool runs standard logic synthesis without the additional optimizations. See *Logic Synthesis with Enhanced Optimization*, on page 40 of the *User Guide* for details. Default: 1 (enabled)
-fanin_limit *value*	*MAX family* Sets the maximum fanin during synthesis. Use only when `map_logic` is true. To improve routability, reduce the value. Default: 40
-fast_synthesis 1|0	*Synplify Premier* *Stratix, Arria, and Cylone families* Enables/disables some mapper optimizations to improve logic synthesis runtimes. See Chapter 19, *Fast Synthesis* of the *User Guide* for details.
-fix_gated_and_generated_clocks 1|0	*All families except MAX, MAX II, and MAX V.* Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-map_logic 1 | 0	*MAX family* Turns on direct mapping to ATOMs or LCELLs during synthesis. See *Altera ATOM Primitives*, on page 1879 for more information. Default: 1 (direct mapping enabled)
-maxfan *value*	*All families except MAX* Establishes a limit for fanout-based optimizations See *Fanout Limits in Altera Designs*, on page 1911 for more information.
-no_sequential_opt 1|0	Enables or disables the sequential optimizations for the design. See *Sequential Optimizations in Altera Designs*, on page 1914 for details. Default: 0 (sequential optimizations enabled)
-physical_synthesis 1|0	*Synplify Premier* *Stratix, Arria, and Cyclone families* Enables/disables physical synthesis, which includes global placement, placement-aware optimizations, and graph-based placement.
-pipe 1 | 0	Runs designs at a faster frequency by moving registers located outside of a multiplier module back into the module to create pipeline stages. See *Pipelining*, on page 426 in the *User Guide* for a procedure.
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
-resource_sharing 1|0	Enables or disables resource sharing. See *Sharing Resources*, on page 448 in the *User Guide* for information about using it.
-retiming 1 | 0	When enabled (1), registers may be moved into combinational logic to improve performance. Enabling -retiming also enables -pipe. See *Retiming*, on page 430 in the *User Guide* for a procedure. Default: 0 (retiming disabled)
-run_prop_extract 1 | 0	Determines whether the tool extracts properties for annotation. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-RWCheckOnRam 1 | 0	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
-soft_buffers 1 | 0	*MAX family* Controls the insertion of soft buffers during synthesis. Buffers are inserted when the value is true. Default: true
-symbolic_fsm_compiler 1 | 0 **-autosm 1 | 0**	Enables or disables the FSM compiler. Controls the use of FSM synthesis for state machines. FSM synthesis is enabled when the value is 1 or true. See *Running the FSM Compiler*, on page 455 in the *User Guide* for details of its use. Default: 0 or false (FSM synthesis disabled)
-syn_altera_model on|off | clearbox_only	Determines whether Clearbox is called for Altera Megafunctions during synthesis. For more information, see *Implementing Megafunctions with Clearbox Models*, on page 1002.
-timequest 1|0	*Stratix II and Stratix II GX* Determines whether to use the Altera Quartus II TimeQuest Timing Analyzer (1) or the Classic Timing Analyzer (0). The Quartus II TimeQuest Timing Analyzer is the default for Stratix IV, Stratix III, Arria GX, and Arria II GX devices.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-update_models_cp 1 | 0	*All families except MAX* Determines whether changes to locked compile points force the remapping of their parents (1), so that the new timing model of the child is taken into account. See *Compile Point Remapping in Altera Designs*, on page 1915, for details.
-use_fsm_explorer 1|0	Enables or disables the FSM Explorer. See *Running the FSM Explorer*, on page 459 in the *User Guide* for information about using it.
-verification_mode 1|0	*Stratix, Arria, and Cylone families* Enables or disables the Verification Mode option. See *Verification Mode in Altera Designs*, on page 1914.
-voltage *value*	*Stratix III* Sets a core voltage for Stratix III devices that support speed grade 4. See *Core Voltage*, on page 1910.

Technology Keywords

The following table lists the -technology keyword for different Altera technologies:

Family	Valid -technology Keywords
Arria	ARRIA-GX, ARRIAII-GX, ARRIAII-GZ, ARRIAV
Cyclone	CYCLONE, CYCLONEII, CYCLONEIII, CYCLONEIV-E, CYCLONEIV-GX, CYCLONEV
MAX	MAX3000
MAX II	MAXII, MAXV
Stratix	STRATIX, STRATIXGX, STRATIX_HC, STRATIXII, STRATIXII-GX, STRATIXIII, STRATIXIV, STRATIXV, HARDCOPYII, HARDCOPYIII, HARDCOPYIV

Copyright © 2013 Synopsys, Inc.

project Command for Altera Architectures

Use the Tcl project command to specify file format options. These options are the equivalents of the ones you set in the Implementation Options dialog box. The following table lists the options used to specify file formats for Altera devices. For a complete list of project command options, see project, on page 1662, or type help project in a Tcl Script window:

Option	Description
-result_format vqm	Specifies the synthesis result file format for the implementation. The format must be vqm (lower-case letters).
-result_file *filename*	Specifies the name of the synthesized netlist for the implementation.

Copyright © 2013 Synopsys, Inc.

Altera Attribute and Directive Summary

The table below summarizes attributes available with Altera technology.

Attribute/Directive	Description
altera_io_powerup	Controls the power-up mode for I/O registers that do not have predefined preset or clear conditions.
altera_logiclock_location	Assigns a compile point to an Altera LogicLock region, specifying the location of the region). This attribute is not available in Synplify.
altera_logiclock_size	Assigns a compile point to an Altera LogicLock region, specifying the size of the region. This attribute is not available in Synplify.
black_box_pad_pin (D)	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins (D)	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
full_case (D)	Specifies that a Verilog case statement has covered all possible cases.
loop_limit (D)	Specifies a loop iteration limit for for loops.
parallel_case (D)	Specifies a parallel, multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
syn_allow_retiming	Determines whether registers may be moved into combinational logic to improve performance. This attribute is not available in Synplify.
syn_allowed_resources	Specifies allowed resources for compile points. The value assigned to a given compile point includes the resources used by its children (at all levels). This attribute is not available in Synplify.
syn_black_box (D)	Defines a black box for synthesis.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_direct_enable (D and A)	Identifies the signal to use as the enable input to an enable flip-flop, when multiple candidates are possible.
syn_edif_name_length	Determines the length of port names in an Altera output EDIF file.
syn_encoding	Specifies the encoding style for state machines.
syn_enum_encoding (D)	Specifies the encoding style for enumerated types (VHDL only).
syn_forward_io_constraints	Enables I/O constraints to forward-annotate to the place-and-route tool.
syn_force_seq_prim (D)	Indicates that the fix gated clocks algorithm can be applied to the associated primitive. The Synplify software does not support this feature.
syn_gatedclk_clock_en (D)	Specifies the name of the enable pin within a black box. The Synplify software does not support this feature.
syn_gatedclk_clock_en_polarity (D)	Indicates the polarity of the clock enable port on a black box, so that the software can apply the algorithm to fix gated clocks. The Synplify software does not support this feature.
syn_hier	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
syn_highrel_ioconnector	Adds I/O connectors that interface to modules with distributed TMR or DWC, which allows you to access the signals within its boundaries ensuring their inputs and outputs are not a single-point of failure.
syn_isclock (D)	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_keep (D)	Prevents the internal signal from being removed during synthesis and optimization.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_loc	Specifies pin locations for I/O pins and cores, and forward-annotates this information to the place-and-route tool.
syn_maxfan	Overrides the default fanout guide for an individual input port, net, or register output.
syn_multstyle	Determines whether multipliers are implemented in logic or hardware blocks.
syn_netlist_hierarchy	Determines whether the output netlist is flat or hierarchical.
syn_noarrayports	Prevents the ports in the output netlist from being grouped into arrays, and leaves them as individual signals.
syn_noclockbuf	Turns off automatic clock buffers for entire modules or nets, or just for specific input ports.
syn_no_compile_point	Ignores modules as compile points in the Automatic Compile Point flow.
syn_noprune (D)	Controls the automatic removal of instances with outputs that are not driven.
syn_pipeline	Specifies that registers be moved into multipliers to improve frequency. The Synplify software does not support pipelining.
syn_preserve (D)	Enables or prevents sequential optimizations, which eliminate redundant registers and registers with constant drivers.
syn_probe	Adds probe points for testing and debugging. The Synplify software does not support probe insertion.
syn_radhardlevel	Enables the triple modular redundancy and voting logic in the design.
syn_ramstyle	Determines how RAMs are implemented.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_replicate	Controls the replication of registers.
syn_resources	Specifies the resources available for black boxes. It is attached to Verilog black-box modules or VHDL architectures or component definitions.
syn_romstyle (Altera)	Determines how ROM architectures are implemented.
syn_rw_conflict_logic	Allows synthesis to NOT automatically infer a block RAM when unnecessary, and ensures that the tool does NOT insert bypass logic around the RAM to prevent a simulation mismatch.
syn_sharing (D)	Specifies resource sharing of operators.
syn_srlstyle	Specifies register packing for Stratix altshift_tap mega function.
syn_state_machine (D)	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n> (D)	Defines the clock to output delay timing through a black box. The *n* indicates a value between 1 and 10.
syn_tpd<n> (D)	Specifies timing propagation for combinational delay through a black box. The *n* indicates a value between 1 and 10.
syn_tristate(D)	Specifies that a black box pin is a tristate pin.
syn_tsu<n> (D)	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.
syn_useenables	Prevents generation of registers with clock enable pins.
syn_useioff (Altera)	Determines whether flip-flops are packed in the I/O ring to improve input/output path timing.
syn_vote_loops	Use this attribute when a loop exists in the module for which distributed TMR is applied.
translate_off/translate_on (D)	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

(D) = directive. (D) and (A) = both directive and attribute. All others are attributes.

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

APPENDIX C

Designing with Lattice

This chapter discusses the following topics for synthesizing Lattice designs:

- Lattice Technology Support, on page 1950
- IP Encryption Flow for Lattice, on page 1951
- Lattice Features, on page 1952
- Lattice Constraints, Attributes, and Options, on page 1976
- Device Mapping Options, on page 1979
- Design Options, on page 1983
- Implementation Results, on page 1985
- ORCA Devices, on page 1986
- LatticeECP/EC and Later Devices, on page 1989
- LatticeSC/SCM and LatticeXP2/XP Devices, on page 1993
- MachXO and Platform Devices, on page 1997
- ispGAL, ispGDX, and ispLSI Devices, on page 2001
- ispXPGA Devices, on page 2004
- ispXPLD5000MX and ispLSI5000VE Devices, on page 2007
- ispMACH and MACH Devices, on page 2011
- Lattice iCE65 and iCE40 Devices, on page 2014
- set_option Command for Lattice Architectures, on page 2017
- Lattice Attribute and Directive Summary, on page 2021

Copyright © 2013 Synopsys, Inc.

Lattice Technology Support

This section describes general guidelines for using the synthesis tool and Lattice place-and-route (P&R) tool with Lattice devices. Refer to:

- Supported Device Families, on page 1950
- Netlist Format, on page 1951

Supported Device Families

The synthesis tool creates technology-specific netlists for the following Lattice technologies:

FPGA

- iCE65/iCE40
- ispXPGA
- LatticeECP3
- LatticeECP2S/ECP2MS/ECP2M/ECP2
- LatticeECP
- LatticeEC
- LatticeSCM
- LatticeSC
- LatticeXP2/XP
- MachXO2/XO
- ORCA FPSC, Series 2, Series3, Series 4
- Platform Manager

Copyright © 2013 Synopsys, Inc.

CPLD

- ispGAL
- ispGDX/GDX2
- ispLSI1K, 2K 5K, 8K, ispLSI5000VE
- ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG
- ispXPLD 5000MX
- MACH

New devices are added on an ongoing basis. For the most current list of supported devices, select Project->Implementation Options and check the list of technologies on the Device tab.

Netlist Format

The synthesis tool outputs EDIF netlist files for use with the Lattice ispLever, Diamond, or iCEcube2 P&R application. These files have .edn and .edf extensions.

IP Encryption Flow for Lattice

Lattice technologies fully support the Synopsys FPGA IP encryption flow. This IP encryption flow is a design flow that encourages interoperability while protecting IP implementations using encryption/decryption technology licensed from RSA Securities. This flow offers the following advantages: interoperability, protection of IP, reuse of IP, and a standard flow for IP encryption.

The Lattice IP encryption scheme uses the flow described in *Working with Lattice IP*, on page 1042. Use this procedure for details on how to incorporate Lattice encrypted IP in your design.

Copyright © 2013 Synopsys, Inc.

Lattice Features

This section describes some supported Lattice features:

- Block RAM Support, on page 1952
- SYNCORE RAMs, on page 1962
- Macros and Black Boxes, on page 1963
- Programmable I/O Cell Latches, on page 1963
- Initial Value Support for Registers, on page 1963
- Hierarchical Attribute Support for Module Generation, on page 1964
- Selective Clock Enable Inference, on page 1965
- Lattice iCE65 and iCE40 Components, on page 1965
- Forward Annotation, on page 1973
- Supported Timing Constraints, on page 1973
- Synthesis Reports, on page 1974

Block RAM Support

The LatticeEC/ECP device families support three Block RAM primitives:

- SP8KA – a single-port RAM with one read and one write port. The parameter WRITEMODE can be set to either NORMAL, WRITETHROUGH, or READBEFOREWRITE.
- DP8KA – a dual-port RAM with read and write ports. The parameters WRITEMODE_A and WRITEMODE_B can be set to either NORMAL, WRITETHROUGH, or READBEFOREWRITE.
- PDP8KA – a pseudo dual-port RAM with a read-only port and a write-only port. There is no memory access mode for this primitive. If the same location is being read and written to in the same cycle, the RAM behaves as WRITETHROUGH. The table shows the pin functions on the Block RAM primitive.

Copyright © 2013 Synopsys, Inc.

Function	SP8KA	DP8KA	PDP8KA
Clock Enable	CE	CEA, CEB	CEW, CER
Clock	CLK	CLKA	CLKW, CLKR
Chip Select	CS[2:0]	CSA[2:0], CSB[2:0]	CSW[2:0], CSR[2:0]
Reset	RST	RSTA, RSTB	RST
Write Enable	WE	WEA, WEB	WE
Address	AD[12:0]	ADA[12:0], ADB[12:0]	ADW[12:0], ADR[12:0]
Write Data	DI[17:0]	DIA[17:0], DIB[17:0]	DI[35:0]
Read Data	DO[17:0]	DOA[17:0], DOB[17:0]	DO[35:0]

Lattice Block RAM Specifications

- The Lattice mapper does not support the following features for the RAM primitives:
  - CS decode when more than one RAM primitive is used
  - GSR support on RAMS
- The RAM size configuration is set by the DATA_WIDTH parameter.
  - SP8KA – DATA_WIDTH
  - DP8KA – DATA_WIDTH_A, DATA_WIDTH_B
  - PDP8KA – DATA_WIDTH_W, DATA_WIDTH_W
- Block RAM does not have bus ports. Instead of DIA[17:0], the ports are DIA0...DIA17.
- The Address bus is always left justified.
- The Data bus is always right justified.

Copyright © 2013 Synopsys, Inc.

Single Address RAM Styles

RAMN_RW

The read address and write address are the same with the exception of a flip-flop on the read data out. Based on the RTL coding style, WRITEFIRST, READFIRST, and NO_CHANGE modes apply. The mapper always uses the SP8KA Block RAM primitive.

Synchronous Set/Reset: Because the Lattice RAM primitives do not support a reset pattern, the RAM is inferred only if the output is registered to a reset flip-flop. Set flip-flops and pattern reset flip-flops are not supported.

Enable: An output flip-flop with an enable cannot be mapped to the clock enable of the block RAM primitive because it will affect the write enable. An exception occurs when the enable is a compliment of the write enable.

WRITE_FIRST Style

The Block RAM is set to WRITE_FIRST mode. Bypass logic is not required.

READ_FIRST Style

The Block RAM is set to READ_FIRST mode. Bypass logic is not required.

NO_CHANGE Style

The Block RAM is set to NO_CHANGE mode. Bypass logic is not required.

The ports are DOUT, ADDR, DIN, WE, CLK, OCLK, RST, EN

```
ramn_rw_generics[] = {
    { "family", 's' },
    { "width", 'i' },
    { "addrwidth", 'i' },
    { "depth", 'i' },
    { "dout_reg", 'b' },
    { "din_reg", 'b' },
    { "rst_data", 's' },
    { "wr_mode", 's' },
    { "addr_reg", 'b' },
};
```

RAM_RWP

The read address is the pipelined write address. The RTL style is WRITEFIRST. No bypass logic is required. The mapper always uses the SP8KA Block RAM primitive.

Copyright © 2013 Synopsys, Inc.

- **Synchronous Set/Reset:** If the address flip-flop has a set/reset signal, the RAM is considered as a generic case of RAM_R_W and mapped using dual-port RAMs.
- **Enable:** An address flip-flop with an enable cannot be mapped to the clock enable of the Block RAM primitive because it will affect the write enable. In such cases the RAM is considered a generic case of RAM_R_W and mapped using dual-port RAMs.

The ports are DOUT, ADDR, DIN, WE, CLK, OCLK

```
rw_generics[] = {
   { "family", 's' },
   { "width", 'i' },
   { "addrwidth" 'i' },
   { "depth", 'i' },
   { "addr_reg”, 'b' },
   { "din_reg", 'b' },
   { "dout_reg", 'b' },
};
```

RAM_RW

The read and write address is the same and can only be implemented in a dual-port Block RAM. The write address for the Block RAM is the output of the addr_r flip-flop. The read address for the Block RAM is tapped from the input of the address flip-flop. Block RAM mode is set to READFIRST. Bypass logic is built to handle read/write collisions, unless disabled by the syn_no_rw_check attribute. The mapper always uses the DP8KA Block RAM primitive.

- **Synchronous Set/Reset:** is allowed on the address flip-flop. The read address to block RAM is ANDed with reset or ORed with set. Because the write address to the Block RAM is fed from the output of the address flip-flop, the flip-flop can support set/reset.
- **Enable:** is allowed on the address flop.

The ports are DOUT, ADDR, DIN, WE, CLK, OCLK

```
rw_generics[] = {
   { "family", 's' },
   { "width", 'i' },
   { "addrwidth", 'i' },
   { "depth", 'i' },
   { "addr_reg", 'b' },
   { "din_reg", 'b' },
   { "dout_reg", 'b' },
};
```

Dual Address RAM Styles

RAMN_R_W

The read and write addresses are independent with a flip-flop on the read data out. Based on the RTL coding style, WRITEFIRST, READFIRST, and NO_CHANGE modes apply. The mapper always uses the DP8KA Block RAM primitive.

- **Synchronous Set/Reset:** Because the Lattice RAM primitives do not support a reset pattern, the RAM is inferred only if the output is registered to a reset flip-flop. Set flip-flops and pattern reset flip-flops are not supported.
- **Enable:** An output flip-flop with an enable cannot be mapped to the clock enable of the block RAM primitive because it affects the write enable. An exception occurs when the enable is a compliment of the write enable.

READ_FIRST Style

The write port is set to READ_FIRST mode. Hence bypass logic is not required.

Copyright © 2013 Synopsys, Inc.

The ports are DOUT, RADDR, DIN, WADDR, WE, CLK, OCLK, W_EN, EN, RST

```
ramn_r_w_generics[] = {
   { "family", 's' },
   { "width", 'i' },
   { "addrwidth", 'i' },
   { "depth", 'i' },
   { "dout_reg", 'b' },
   { "din_reg", 'b' },
   { "rst_data", 's' },
   { "wr_mode", 's' },
   { "raddr_reg", 'b' },
   { "waddr_reg", 'b' },
};
```

RAMN_RW_R

There are two independent read addresses, and both are registered. The mapper uses the DP8KA Block RAM primitive. Based on the RTL coding style, WRITEFIRST, READFIRST, and NO_CHANGE modes are available.

- **Synchronous Set/Reset:** Because Lattice RAM primitives do not support a reset pattern, the RAM is inferred only if the output is registered to a reset flip-flop. Set flip-flops and pattern reset flip-flops are not supported.

WRITE_FIRST Style

The RTL style for the read/write port is WRITE_FIRST. The other RTL read-only port has no implied mode. The write port of the Block RAM is set to WRITE_FIRST mode. Bypass logic is not required.

READ_FIRST Style

Both the RTL read ports have an implied READ_FIRST mode. The write port of the Block RAM is set to READ_FIRST. Bypass logic is not required.

NO_CHANGE Style

The Block RAM write port is set to NO_CHANGE mode. Bypass logic is not required.

Copyright © 2013 Synopsys, Inc.

The ports are R_DOUT, RADDR, W_DOUT, DIN, WADDR, WE, CLK, R_OCLK, W_EN, EN, R_RST, W_RST

```
ramn_rw_r_generics[] = {
   { "family", 's' },
   { "width", 'i' },
   { "addrwidth", 'i' },
   { "depth", 'i' },
   { "rdout_reg", 'b' },
   { "wdout_reg", 'b' },
   { "raddr_reg", 'b' },
   { "din_reg", 'b' },
   { "waddr_reg", 'b' },
   { "r_rst_data", 's' },
   { "w_rst_data", 's' },
   { "r_wr_mode", 's' },
   { "w_wr_mode", 's' },
};
```

RAM_R_W

The read and write addresses are independent. The RTL style is WRITEFIRST. The mapper always uses the DP8KA Block RAM primitive. Bypass logic is built to handle read/write collisions, unless disabled by the syn_no_rw_check attribute. The read address should be registered.

- **Synchronous Set/Reset:** is allowed on the address flip-flop. The read address to Block RAM is ANDed with reset or ORed with set. Because the write address to the Block RAM is fed from the output of the address flip-flop, the flip-flop can support set/reset.
- **Enable:** is allowed on the address flip-flop. Read address to Block RAM is modified.

The ports are DOUT, RADDR, DIN, WADDR, WE, CLK, OCLK

```
genericinfo r_w_generics[] = {
    { "family", 's' },
    { "width", 'i' },
    { "addrwidth", 'i' },
    { "depth", 'i' },
    { "raddr_reg", 'b' },
    { "din_reg", 'b' },
    { "dout_reg", 'b' },
    { "waddr_reg", 'b' },
};
```

RAM_RWP_R

There are two independent read addresses, and both are registered. The write address and one of the read addresses are pipelined. The mapper always uses the DP8KA Block RAM primitive. The RTL style is WRITEFIRST. The read/write port of the block RAM is programmed to WRITEFIRST, and does not require bypass logic. The read-only port of the Block RAM requires bypass logic.

Copyright © 2013 Synopsys, Inc.

The ports are R_DOUT, RADDR, W_DOUT, DIN, WADDR, WE, CLK, R_OCLK, W_OCLK

```
rw_r_generics[] = {
   { "family", 's' },
   { "width", 'i' },
   { "addrwidth", 'i' },
   { "depth", 'i' },
   { "rdout_reg", 'b' },
   { "wdout_reg", 'b' },
   { "raddr_reg", 'b' },
   { "din_reg", 'b' },
   { "waddr_reg", 'b' },
};
```

SYNCORE RAMs

The SYNCORE Memory Compiler is available under the IP Wizard to help you generate HDL code for your specific RAM implementation requirements. For information on using the SYNCORE Memory Compiler, see *Specifying RAMs with SYNCore*, on page 939 in the *User Guide*.

Macros and Black Boxes

You can instantiate Lattice macros, such as gates, counters, flip-flops and I/Os, by using the synthesis tool-supplied Lattice macro libraries, which contain predefined Lattice black-box macros. There are Verilog and VHDL libraries. For information about using these macro libraries, see *Instantiating Lattice Macros*, on page 772 in the *User Guide*. For general information about instantiating black boxes, see *Instantiating Black Boxes in Verilog*, on page 749 and *Instantiating Black Boxes in VHDL*, on page 961.

Programmable I/O Cell Latches

The tool automatically infers programmable I/O cell (PIC) latches for the LatticeSC/SCM, LatticeXP/XP2, and LatticeEC/ECP device families. Use syn_keep if you do not want to infer a PIC. See *Inferring Lattice PIC Latches*, on page 775 in the *User Guide* for details and examples.

Initial Value Support for Registers

You can specify INIT values for registers, which include the following:

- Registers or latches (including PIC)

  For registers with control signals such as async set/reset, the mapper generates a warning message when the INIT value does not match the set/reset value.

- Counters and shift registers
- Registers associated with DSPs

  This may impact register packing.

  – For registers with control signals and an INIT value of either 1 or 0, the mapper ignores the initial value and register packing is not impacted.

  – For registers without control signals and an INIT value of 1, the mapper honors the initial value and register packing will not happen.

- Registers associated with RAMs

  This may impact register packing and block RAM inferencing. For Lattice EBRs, registers with INIT value of 1 are not packed into the RAM.

Copyright © 2013 Synopsys, Inc.

Synthesis software infers the following Lattice register primitives with power-up value of '1'/'0':

- FD1S3AX
  (Positive-edge Triggered D Flip-flop, GSR used for Clear with power-up value 0)
- FD1S3AY
  (Positive-edge Triggered D Flip-flop, GSR used for Preset with power-up value 1)
- FD1P3AX
  (Positive-edge Triggered D Flip-flop with Positive Level Enable, GSR used for Clear with power-up value 0)
- FD1P3AY
  (Positive-edge Triggered D Flip-flop with Positive Level Enable, GSR used for Preset with power-up value 1)

Constant propagation is affected only when the value on the data pin does not match the INIT value and retiming should not be affected.

Hierarchical Attribute Support for Module Generation

You can define attribute values at the module level which apply to elements at lower levels in the hierarchy. The Module Generator will generate PCS, PLL, DLL, and MACO type blocks, which can include single or multiple levels of hierarchy. Although modules contain multiple levels, attribute values are set at the top level.

Example

```
module PPLL (CLKIN, FB, MCLK, NCLK, LOCK, INTFB) /* synthesis
   syn_black_box */;
input CLKIN, FB;
output MCLK, NCLK, INTFB, LOCK;
parameter VCOTAP = "1";
endmodule // EXPLLA
```

```
module macro (clki, clko);
input clki;
output clko;
parameter VCOTAP = "1";
defparam u1.VCOTAP = VCOTAP;
PPLL u1 (.CLKIN(clki), .FB(fb), .MCLK(clko), .NCLK(),
   .INTFB(fb), .LOCK());
endmodule // macro

module top (clki, clko1, clko2);
input clki;
output clko1, clko2;
defparam u1.VCOTAP = "2";
defparam u2.VCOTAP = "6";
macro u1 (.clki(clki), .clko(clko1));
macro u2 (.clki(clki), .clko(clko2));
endmodule // top
```

Selective Clock Enable Inference

For CPLD devices, you can extract clock enable to enhance performance using the syn_useenables attribute. See syn_useenables, on page 1559 for more information.

Lattice iCE65 and iCE40 Components

The following topics describe how the synthesis tool handles various Lattice iCE65 and iCE40 components, and shows you how to work with or manipulate them during synthesis to get the results you need:

- Lattice iCE65 and iCE40 RAM, on page 1965
- Lattice iCE65 and iCE40 ROM, on page 1966
- Timing Propagation Through PLLs, on page 1967
- Lattice iCE65 and iCE40 Latches, on page 1970

Lattice iCE65 and iCE40 RAM

Lattice iCE65 and iCE40 technologies support single-port or dual-port synchronous memory.

Copyright © 2013 Synopsys, Inc.

RAM Mapping

Lattice iCE65 and iCE40 RAM cannot resolve simultaneous read or write modes:

- The synthesis software applies glue logic to avoid simultaneous read/write conflicts with the following modes.
  - Writefirst
  - Nochange
- For Readfirst mode, a simulation mismatch occurs since the software cannot apply glue logic.

Readfirst mode is not supported, so map these RAM to registers. You can disable automatic RAM inference or specify the RAM implementation with the syn_ramstyle attribute. For information on inferring and implementing RAM, see Inferring RAMs, on page 369 of the *User Guide*.

Block RAMs can be configured as:

- Single-port memory
- Simple dual-port memory

Lattice iCE65 and iCE40 ROM

You can map ROM to block RAM with the syn_romstyle attribute.

RAM and ROM Initialization

You can initialize RAMs and ROMs by initializing values in RTL. See *Initializing Values in RTL*, on page 1966.

Initializing Values in RTL

Initial values specified in the RTL for memories can be mapped to startup values on the FPGA. Lattice iCE65 and iCE40 devices support power on startup values for memories. See Initializing RAMs, on page 394 in the *User Guide* for step-by-step procedures. You can initialize values in Verilog and VHDL. See:

- Initializing RAMs in Verilog, on page 394
- Initializing RAMs in VHDL, on page 395

Copyright © 2013 Synopsys, Inc.

Timing Propagation Through PLLs

The synthesis tool can propagate timing constraints through instantiated PLLs for P-series devices of the Lattice iCE65 and iCE40 technologies.

The iCE65 devices support PLL primitives: SB_PLL_CORE, SB_PLL_PAD, and SB_PLL_2_PAD. The iCE40 devices support PLL primitives: SB_PLL40_CORE, SB_PLL40_PAD, SB_PLL40_2_PAD, SB_PLL40_2F_CORE and SB_PLL40_2F_PAD; with different Feedback Path modes (for example, Simple Mode, Delay Mode, Phase and Delay Mode, External Mode). Note that only Fixed value is supported for Delay adjustment mode; Dynamic value is not supported currently.

When using a PLL for on-chip clock generation, you only need to define the clock at the primary inputs. The synthesis software propagates clocks through the PLL and calculates derived clock frequency, with phase and delay offset. Clocks at the outputs of a PLL are generated automatically as needed, while taking into account any phase shift or frequency change.

The synthesis tool assigns all the PLL outputs to the same clock group. However, only the clocks at the PLL inputs are forward-annotated to the `scf` file. The PLL configuration is generated from iCEcube software.

Example of the SB_PLL_CORE (Simple Mode)

It is strongly recommended that the PLL configuration be generated from the PLL configuration tool of the iCEcube software. An example of a PLL CORE using Simple Mode is shown in the Verilog code below:

```
//PLL in Simple mode

module test_pll
   (REFERENCECLK,
    PLLOUTCORE,
    PLLOUTGLOBAL,
    RESET,
    LOCK);

input REFERENCECLK;
input RESET;    /* To initialize the simulation properly, */
   /* the RESET signal (Active Low) must be asserted at the */
   /* beginning of the simulation */
output PLLOUTCORE;
output PLLOUTGLOBAL;
output LOCK;
```


Copyright © 2013 Synopsys, Inc.

```
SB_PLL_CORE mypll_inst(.REFERENCECLK(REFERENCECLK),
   .PLLOUTCORE(PLLOUTCORE),
   .PLLOUTGLOBAL(PLLOUTGLOBAL),
   .RESET(RESET),
   .BYPASS(1'b0),
   .LOCK(LOCK));

//\\ Fin=60, Fout=120;
defparam mypll_inst.DIVR = 4'b0000;
defparam mypll_inst.DIVF = 6'b000111;
defparam mypll_inst.DIVQ = 3'b010;
defparam mypll_inst.FILTER_RANGE = 3'b100;
defparam mypll_inst.FEEDBACK_PATH = "SIMPLE";
defparam mypll_inst.PLLOUT_PHASE = "NONE";
defparam mypll_inst.ENABLE_ICEGATE = 1'b0;
endmodule
```

This SB_PLL_CORE is displayed in the RTL Analyst View.

Example of the SB_PLL40_2F_PAD (Delay Mode)

An example of a PLL CORE using Delay Mode is shown in the Verilog code below:

```
//PLL in Delay mode

module test_pll (PACKAGEPIN,
   PLLOUTCOREA,
   PLLOUTCOREB,
   PLLOUTGLOBALA,
   PLLOUTGLOBALB,
   RESETB);

input PACKAGEPIN;
input RESETB; /* To initialize the simulation properly, the */
   /* RESET signal (Active Low) must be asserted at the */
   /* beginning of the simulation */
output PLLOUTCOREA;
output PLLOUTCOREB;
output PLLOUTGLOBALA;
output PLLOUTGLOBALB;

SB_PLL40_2F_PAD mypll_inst(.PACKAGEPIN(PACKAGEPIN),
   .PLLOUTCOREA(PLLOUTCOREA),
   .PLLOUTCOREB(PLLOUTCOREB),
   .PLLOUTGLOBALA(PLLOUTGLOBALA),
   .PLLOUTGLOBALB(PLLOUTGLOBALB),
   .RESETB(RESETB),
   .BYPASS(1'b0));

defparam mypll_inst.DIVR = 4'b0001;
defparam mypll_inst.DIVF = 6'b000011;
defparam mypll_inst.DIVQ = 3'b010;
defparam mypll_inst.FILTER_RANGE = 3'b010;
defparam mypll_inst.FEEDBACK_PATH = "DELAY";
defparam mypll_inst.DELAY_ADJUSTMENT_MODE_FEEDBACK = "FIXED";
defparam mypll_inst.DELAY_ADJUSTMENT_MODE_RELATIVE = "FIXED";
defparam mypll_inst.PLLOUT_SELECT_PORTA = "GENCLK_HALF";
defparam mypll_inst.PLLOUT_SELECT_PORTB = "GENCLK";
defparam mypll_inst.FDA_FEEDBACK = 4'b0011;
defparam mypll_inst.FDA_RELATIVE = 4'b0111;
defparam mypll_inst.ENABLE_ICEGATE_PORTA = 1'b0;
defparam mypll_inst.ENABLE_ICEGATE_PORTB = 1'b0;

endmodule
```

This SB_PLL40_2F_PAD is displayed in the RTL Analyst View.

Copyright © 2013 Synopsys, Inc.

Lattice iCE65 and iCE40 Latches

The synthesis software implements latches using LUTs for the specified iCE65 and iCE40 technologies. Types of latches supported include:

- Simple latch
- Latch with asynchronous/synchronous set
- Latch with asynchronous/synchronous reset

Example 1: Verilog RTL for a Latch with Asynchronous Set

```
module test(set,en,d,q) ;
input set,en;
input d;

output q;
reg q;

always @ (en or d or set)
begin
```

```
      if(set) q <= 1;
      else if(en)
         q <= d;
   end
   endmodule
```

The following figure shows the schematic representations for a latch with asynchronous set.

Example 2: Verilog RTL for a Latch with Synchronous Reset

```
module test(rst,en,d,q) ;
input rst,en;
input d;

output q;
```


Copyright © 2013 Synopsys, Inc.

```
reg q;

always @ (en or d or rst)
begin
   if(en)
   if(rst) q <= 0;
   else  q <= d;
end
endmodule
```

The following figure shows the schematic representations for a latch with synchronous reset.

RTL View

Technology View

Forward Annotation

The synthesis tool generates Lattice-compliant constraint files from selected constraints that are forward-annotated (read in and then used) by the Lattice ispLever and Diamond place-and-route program. The Lattice constraint file uses the `.lpf` extension. This constraint file must be imported into the Lattice flow.

For Lattice iCE65 and iCE40 devices, the software generates the output files (`edf` and `scf`). The `scf` constraint file forward-annotates timing constraints for the Lattice iCEcube2 place-and-route tool.

By default, Lattice constraint files are generated from the synthesis tool constraints. You can then forward-annotate these files to the place-and-route tool. To disable this feature, deselect the Write Vendor Constraint File box (on the Implementation Results tab of the Implementation Options dialog box).

For a detailed procedure, see *Forward-Annotating Lattice Constraints*, on page 784 in the *User Guide*.

Supported Timing Constraints

The constraint file generated for the Lattice ispLEVER place-and-route tool uses the $DESIGN_synplify.lpf file. Follow these steps to forward-annotate your synthesis constraint file to ispLEVER:

1. Set your constraints in the SCOPE spreadsheet (see *SCOPE Tabs*, on page 471).
2. Run your design. The synthesis tool creates the $DESIGN_synplify.lpf file in the same directory as your results files.
3. Open the Lattice ispLEVER place-and-route tool. Run the Map stage, which reads in the $DESIGN_synplify.lpf file.
4. Run the PAR and BIT stages in Foundry.

The following constraints are forward-annotated to ispLEVER in this file:

- define_false_path
- define_multicycle_path
- define_reg_input_delay

Copyright © 2013 Synopsys, Inc.

- define_reg_output_delay
- global frequency

Synthesis Reports

The synthesis tool generates synthesis reports for Lattice designs that include a resource usage report and a timing report.

Resource Usage Report

The resource usage report lists the number of each type of cell used, and the number of look-up tables and registers.

For example, in the ORCA series 2, 3, 4 device families, four 4-input look-up tables and four registers fit into each *function unit* (PFU). After placement and routing, the ispLEVER software issues a report with the number of occupied PFUs. An occupied PFU indicates that at least one look-up table or register was used. Because the synthesis tool report references LUTs, you can compare the synthesis report and the place-and-route report on LUTs.

Example: Lattice ORCA 4 Resource Usage Report

```
Resource Usage Report
Part: 4e02-2

Register bits: 206 of 5616 (11%)
I/O cells:       26

Details:
BMZ6:           24
FADD4:          2
FD1P3AX:        45
FD1P3AY:        35
FD1S3AX:        49
FD1S3AY:        3
FD1S3IX:        15
FL1S3AX:        1
FSUB4:          2
GSR:            1
IBM:            2
INV:            1
OFS1P3DX:       24
ORCALUT4:       443
```

```
ORCALUT5:        37
PFUMX:           63
RCE32X4:         2
VHI:             1
VLO:             1
ghost:           1
Found clock eight_bit_uc|clock with period 1000.00ns
```

Timing Report

Timing reports located in the log file, are generated during synthesis for Lattice technologies. (see *Timing Reports*, on page 448). To view the log file, click View Log.

Copyright © 2013 Synopsys, Inc.

Lattice Constraints, Attributes, and Options

This section explains how to implement Lattice constraints, attributes, and options. Refer to:

- Constraints and Attributes, on page 1976
- Disable I/O Insertion Globally or Port by Port, on page 1977
- Device Mapping Options, on page 1979

Constraints and Attributes

Specify timing constraints, general HDL attributes, and Lattice-specific attributes to improve your design. Manage these files with the SCOPE constraints and attributes editor. Lattice has vendor-specific I/O standard constraints it supports for synthesis. For a list of supported I/O standards, see *Lattice iCE65/iCE40 I/O Standards*, on page 1976.

Lattice iCE65/iCE40 I/O Standards

The following table contains the applicable I/O standards for Lattice iCE65/iCE40 devices.

Lattice iCE65/iCE40 I/O Standards

SB_LVCMOS	SB_MDDR2
SB_LVCMOS15_2	SB_MDDR4
SB_LVCMOS15_4	SB_MDDR8
SB_LVCMOS18_2	SB_MDDR10
SB_LVCMOS18_4	SB_SSTL2_CLASS_1
SB_LVCMOS18_8	SB_SSTL2_CLASS_2
SB_LVCMOS18_10	SB_SSTL18_FULL
SB_LVCMOS25_4	SB_SSTL18_HALF
SB_LVCMOS25_8	SB_LVDS_INPUT

NOTE: The I/O standards apply for L04, L08, and P04 devices. The L01 device only supports SB_LVCMOS.

Copyright © 2013 Synopsys, Inc.

Lattice iCE65/iCE40 I/O Standards

SB_LVCMOS25_12	SB_SUBLVDS_INPUT
SB_LVCMOS25_16	
SB_LVCMOS33_8	

NOTE: The I/O standards apply for L04, L08, and P04 devices. The L01 device only supports SB_LVCMOS.

See Also:

- *Industry I/O Standards*, on page 496 for a list of industry I/O standards.
- *Chapter 6, SCOPE Constraints Editor* for the timing constraints.
- *Lattice Attribute and Directive Summary, on page 2021* for a list of attributes.

Disable I/O Insertion Globally or Port by Port

The syn_force_pads attribute command tells the synthesis tool whether to enable or disable I/O pad insertion on a global or port-by-port basis.

The syntax is:

- syn_force_pads=1 – Enables pad insertion.
- syn_force_pads=0 – Disables pad insertion.

See syn_force_pads, on page 1327 for more information.

Copyright © 2013 Synopsys, Inc.

I/O Buffer Support

The synthesis tool supports unconnected I/O buffers created for top-level ports. When you set syn_force_pads=1 either globally or on a port-by-port basis, the software does not optimize away the unconnected buffers. This enables the synthesis tools to do the following:

- Preserve user-instantiated pads
- Insert pads on unconnected ports
- Insert bi-directional pads on bi-directional ports instead of converting them to input ports.
- Insert output pads on unconnected outputs.

This feature is supported for Lattice ECP/EC, LatticeSC/SCM, Lattice MACH, LatticeXP, and ORCA device families.

Copyright © 2013 Synopsys, Inc.

Device Mapping Options

To achieve optimal design results, set the correct implementation options. These options control the following:

- target technology and related parameters
- mapping options to use during synthesis
- output and output formats,
- results directories

You can set options in the Implementation Options dialog box or by typing the set_option command and its arguments in the Tcl window. To open the Implementation Options dialog box, select Project->Implementation Options in the Project view.

One or more of the following options appear in all Lattice devices.

- Fanout Guide, on page 1980
- Disable I/O Insertion, on page 1980
- GSR Resource, on page 1980
- Map Logic to Macrocells, on page 1981
- Maximum Cell Fanin, on page 1981
- Maximum Terms/Macrocell, on page 1981
- Percentage of Design to Optimize for Timing, on page 1982
- Pipelining, on page 1984
- Retiming, on page 1984
- Compile Point Remapping in Lattice Designs, on page 1982

Copyright © 2013 Synopsys, Inc.

Fanout Guide

Large fanouts can cause large delays and routability problems. During technology mapping, the synthesis tool automatically maintains reasonable fanout limits, keeping the fanout under the limit you specified. For details about this option, see *Setting Fanout Limits*, on page 444 in the *User Guide*. This feature is supported for Lattice iCE65/iCE40, LatticeECP/EC, Lattice ispXPGA, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager device families.

Disable I/O Insertion

The synthesis tool inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist, unless you disable I/O insertion. You can override which I/O pads are used by directly instantiating specific I/O pads. If you manually insert I/O pads, you only insert them for the pins that require them. For further details, see *Controlling I/O Insertion in Lattice Designs*, on page 783in the *User Guide*.

GSR Resource

The LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, Platform Manager, and ORCA technology families have a resource for a GSR (global set reset) signal. The resource is a prerouted signal that goes to the reset input of each flip-flop in the FPGA. The GSR is connected to all flip-flops that are tied to the GSR line, regardless of any other defined reset signals. When the GSR is activated, all registers are reset.

The synthesis tool creates a GSR instance to access the resource, if it is appropriate for your design. Using this resource instead of general routing for a reset signal can have a significant positive impact on the routability and performance of your design.

If there is a single reset in your design, the synthesis tool connects that reset signal to a GSR instance. It forces this even if some flip-flops do not have a reset. Usually, flip-flops without set or reset can be safely initialized because the reset is only used when the device is turned on. If this is not the case, you must turn off the forced use of GSR by the synthesis tool.

Copyright © 2013 Synopsys, Inc.

To turn off the use of GSR, select no for the Force GSR Usage option on the Device tab of the Implementation Options dialog box. This option is no by default; when off all flip-flops must have resets, and all the resets use the same signal before the tool uses GSR.

Map Logic to Macrocells

When this option is set (checked), the design is mapped to macrocells cells (instead of primitive gates) during synthesis. The default value is unchecked which can sometimes give a smaller result.

If you set this option, the following three additional options are available: Maximum Cell Fanin, Maximum Terms/Macrocell, and Percentage of Design to Optimize for Timing. This feature is supported for ispLSI1K, 2K, 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH technologies.

Maximum Cell Fanin

The Maximum Cell Fanin option sets the maximum fanin during synthesis. You can try to improve routability by reducing the fanin limit. The default value is 20. The Maximum Cell Fanin option is available only when the Map logic to macrocells option is enabled. This feature is supported for ispLSI1K, 2K 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH technologies.

Maximum Terms/Macrocell

The Maximum Terms/Macrocell option sets the maximum number of terms per macrocell during synthesis. The default value is 16. This feature is supported for ispLSI1K, 2K, 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH technologies.

Copyright © 2013 Synopsys, Inc.

Percentage of Design to Optimize for Timing

The Percent of design to optimize for timing option is available only when the Map logic to macrocells option is enabled. This option sets the percentage of nets you want optimized during synthesis. The default value is 0 percent. See *Tips for Optimization*, on page 422 of the *User Guide*. This feature is supported for ispLSI1K, 2K, 5K, 8K, ispLSI5000VE, ispLSI5K, ispMACH 4000B, 4000C, 4000V, 4000ZC, 4000ZE, 5000B, 5000VG, and MACH technologies.

Compile Point Remapping in Lattice Designs

The compile point feature is supported for Lattice iCE65/iCE40, LatticeECP/EC, LatticeSC/SCM, LatticeXP2/XP, MachXO, and Platform Manager device families. The Synplify software does not support compile points.

Compile points are smaller synthesis units, which lets you break up a larger design into smaller units and do incremental synthesis. See Synthesizing Compile Points, on page 1109 and Compile Point Basics, on page 1088 for information about the flow and compile points.

The Update Compile Point Timing Data option determines whether changes to a compile point force the remapping of its parents, taking into account the new timing model of the child. The term *child* refers to a compile point that is contained inside another; the term *parent* refers to the compile point that contains the child. These terms are thus not used here in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points.

Copyright © 2013 Synopsys, Inc.

The following table describes the settings for the Update Compile Point Timing Data option:

Disabled	(Default). Only compile points that have changed are remapped, and their remapping does *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used to map and optimize its parents. If the option is disabled and the *interface* of a locked compile point is changed, the immediate parent of the compile point must be changed accordingly. In this case, both are remapped, and the *updated* timing model of the child is used when remapping this parent.
Enabled	Compile point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account because the option was disabled. The timing model of a compile point is updated when either of the following is true: • The compile point is remapped, and the Update Compile Point Timing Data option is enabled. • The interface of the compile point is changed.

Design Options

The following options are located on the Options tab of the Implementation Options dialog box.

- FSM Compiler, on page 1983
- Resource Sharing, on page 1984
- Pipelining, on page 1984
- Retiming, on page 1984
- Gated and Generated Clocks, on page 1985

FSM Compiler

Turning on the FSM Compiler enables special FSM optimizations. See *Optimizing State Machines*, on page 454 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Resource Sharing

When Resource Sharing is enabled, synthesis uses resource sharing techniques. A resource sharing report is available in the log file (see *Resource Usage Report*, on page 612). See *Sharing Resources*, on page 448 in the *User Guide*.

Pipelining

Pipelining multipliers and ROMs allows your design to operate at a faster frequency. The synthesis tool moves registers that follow a multiplier or ROM into the multiplier or ROM, provided that you have registers in your RTL code. The Synplify software does not support pipelining. You can pipeline multipliers and ROMs for the Lattice iCE65/iCE40, LatticeECP/EC, Lattice SC/SCM, LatticeXP, MachXO, and Platform Manager technology families if the ROM is bigger than 512 words.

- To invoke pipelining and apply it globally, enable the Pipelining option in the Project view.
- To apply this attribute locally, add the syn_pipeline attribute to the registers driven by the multipliers or ROMs. See syn_pipeline, on page 1442 for the syntax.

For detailed information about using pipelining, see *Pipelining*, on page 426 of the *User Guide*.

Retiming

Retiming is a powerful technique for improving the timing performance of sequential circuits. The retiming process moves storage devices (flip-flops) across computational elements with no memory (gates/LUTs) to improve the performance of the circuit. Retiming is supported for Lattice iCE65/iCE40, LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager technology families. Also, the Synplify software does not support retiming.

Set global retiming either from the Project view check box or from the Device tab under the Implementation Options button. Use the syn_allow_retiming attribute to enable or disable retiming for individual flip-flops. See syn_allow_retiming, on page 1216 for syntax details, or *Retiming*, on page 430 of the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Pipelining is automatically enabled when retiming is enabled. Use the syn_pipeline attribute to control pipelining. See syn_pipeline, on page 1442 for syntax details.

Gated and Generated Clocks

The Clock Conversion option on the GCC (Synplify Pro) or GCC & Prototyping Tools tab (Synplify Premier), when enabled, attempts to remove the gated clocks from synchronous logic and to convert generated clocks to the original clock with an enable. Gated- and generated-clock conversion is is not available for the CPLD technologies

Gated Clocks

Gated clocks are converted by removing the enable logic from the clock path. Extracting the enable logic allows clock constraints to be applied globally and eliminates broken paths. Converting gated clocks also makes use of the dedicated clock networks to simplify ASIC prototyping. For information on using the Clock Conversion option with gated clocks, see *Working with Gated Clocks*, on page 466 in the *User Guide*.

Generated Clocks

Generated clocks are converted to a circuit with an enable to improve performance by reducing clock skew. For more information on using the Clock Conversion option with generated clocks, see *Optimizing Generated Clocks*, on page 499 in the *User Guide*.

Implementation Results

On the Implementation Results tab of the Implementation Options dialog box, two file formats: edif and src, are available depending on your design's device family.

You can also use the project Tcl command to specify the result file format.

```
project -result_format edif
```


Copyright © 2013 Synopsys, Inc.

ORCA Devices

This section describes guidelines for ORCA devices using the following options in the synthesis tool:

- ORCA Device Mapping Options, on page 1986
- ORCA Attributes, on page 1986
- ORCA set_option Tcl Command, on page 1986

ORCA Device Mapping Options

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project->Implementation Options). You can set the following:

- Fanout Guide, on page 1980
- Disable I/O Insertion, on page 1980
- GSR Resource, on page 1980

ORCA Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with ORCA devices.

ORCA set_option Tcl Command

You can use the set_option Tcl command to specify the same device mapping options as are available through the Implementation Options dialog box (Project->Implementation Options).

This section provides information on specific options for the architectures. For a complete list of options, see *set_option*, on page 1679.

The set_option command supports the following options for ORCA devices.

Copyright © 2013 Synopsys, Inc.

Option	Description
Disable I/O insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Fanout Guide	Sets the fanout limit guideline for the open Project. The default fanout limit is 100. For information about setting fanout limits, see *Fanout Guide*, on page 1980.
Force GSR Usage	Enables (yes) or disables (no) forced usage of the global Set/Reset routing resources for the open Project. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is auto. For additional information about GSR, see *GSR Resource*, on page 1980.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

Copyright © 2013 Synopsys, Inc.

set_option Device Options

This is the syntax for setting device options for ORCA FPSC and ORCA Series 2, 3, and 4 devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | **0** or **-disable_io_insertion 1** | **0**
-fanout_guide *value*
-force_gsr yes | **no** | **auto**
-maxfan *value*
-resolve_multiple_driver 1|**0**
-RWCheckOnRam 1|**0**

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-resource_sharing 1|**0**
-symbolic_fsm_compiler 1|**0** or **-autosm 1**|**0**

Copyright © 2013 Synopsys, Inc.

LatticeECP/EC and Later Devices

This section describes guidelines for using the synthesis tool with LatticeECP/EC and later technologies. The topics include:

- LatticeECP/EC and Later Device Mapping Options, on page 1989
- LatticeECP/EC and Later Attributes, on page 1989
- LatticeECP/EC and Later set_option Tcl Command, on page 1990

LatticeECP/EC and Later Device Mapping Options

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project->Implementation Options). You can set the following:

- Fanout Guide, on page 1980
- Disable I/O Insertion, on page 1980
- Retiming, on page 1984
- Pipelining, on page 1984
- GSR Resource, on page 1980
- Compile Point Remapping in Lattice Designs, on page 1982

LatticeECP/EC and Later Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with the technologies.

Copyright © 2013 Synopsys, Inc.

LatticeECP/EC and Later set_option Tcl Command

You can use the set_option Tcl command to specify the same device mapping options that are available through the Implementation Options dialog box (Project->Implementation Options).

This section provides information on specific options for the technologies. For a complete list of options, see *set_option*, on page 1679.The set_option command supports the following options for the LatticeECP3/ECP2S/ECP2MS/ECP2M/ECP2 and LatticeECP/EC devices.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Fanout Guide	Sets the fanout limit guideline for the open Project. The default fanout limit is 100. For information about setting fanout limits, see *Fanout Guide*, on page 1980.
Force GSR Usage	Enables (yes) or disables (no) forced usage of the global Set/Reset routing resources for the open Project. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is no. For additional information about GSR, see *GSR Resource*, on page 1980.

Copyright © 2013 Synopsys, Inc.

Option	Description
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Compile Point Remapping in Lattice Designs*, on page 1982, for details. The Synplify software does not support compile points.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for LatticeECP3/ECP2S/ECP2MS/ECP2M/ECP2 and LatticeECP/EC devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | **0** or **-disable_io_insertion 1** | **0**
-fanout_guide *value*
-force_gsr yes | **no** | **auto**
-maxfan *value*
-resolve_multiple_driver 1|**0**
-RWCheckOnRam 1|**0**
-update_models_cp 1|**0**

Copyright © 2013 Synopsys, Inc.

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
- **-pipe 1 | 0**
- **-resource_sharing 1|0**
- **-retiming 1 | 0**
- **-symbolic_fsm_compiler 1|0** or **-autosm 1|0**

Copyright © 2013 Synopsys, Inc.

LatticeSC/SCM and LatticeXP2/XP Devices

This section describes guidelines for LatticeSC/SCM and LatticeXP2/XP devices using the following options in the synthesis tool:

- LatticeSC/SCM and LatticeXP2/XP Device Mapping Options, on page 1993
- LatticeSC/SCM and LatticeXP2/XP Attributes, on page 1993
- LatticeSC/SCM and LatticeXP2/XP set_option Tcl Command, on page 1994

LatticeSC/SCM and LatticeXP2/XP Device Mapping Options

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project->Implementation Options). You can set the following:

- Fanout Guide, on page 1980
- Disable I/O Insertion, on page 1980
- Retiming, on page 1984
- Pipelining, on page 1984
- GSR Resource, on page 1980
- Compile Point Remapping in Lattice Designs, on page 1982

LatticeSC/SCM and LatticeXP2/XP Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with LatticeSC/SCM and LatticeXP2/XP devices.

Copyright © 2013 Synopsys, Inc.

LatticeSC/SCM and LatticeXP2/XP set_option Tcl Command

You can use the set_option Tcl command to specify the same device mapping options as are available through the Implementation Options dialog box (Project -> Implementation Options).

This section provides information on specific options for the architectures. For a complete list of options, see *set_option*, on page 1679.The set_option command supports the following options for LatticeSC/SCM and LatticeXP2/XP devices.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Fanout Guide	Sets the fanout limit guideline for the open Project. The default fanout limit is 100. For information about setting fanout limits, see *Fanout Guide*, on page 1980.
Force GSR Usage	Enables (yes) or disables (no) forced usage of the global Set/Reset routing resources for the open Project. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is no. For additional information about GSR, see *GSR Resource*, on page 1980.

Copyright © 2013 Synopsys, Inc.

Option	Description
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Compile Point Remapping in Lattice Designs*, on page 1982, for details. The Synplify software does not support compile points.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for LatticeSC/SCM and LatticeXP2/XP devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | **0** or **-disable_io_insertion 1** | **0**
-fanout_guide *value*
-force_gsr yes | **no** | **auto**
-maxfan *value*
-resolve_multiple_driver 1|**0**
-RWCheckOnRam 1|**0**
-update_models_cp 1|**0**

Copyright © 2013 Synopsys, Inc.

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
- **-pipe 1 | 0**
- **-resource_sharing 1|0**
- **-retiming 1 | 0**
- **-symbolic_fsm_compiler 1|0** or **-autosm 1|0**

Copyright © 2013 Synopsys, Inc.

MachXO and Platform Devices

This section describes guidelines for MachXO, MachXO2, and Platform Manager devices using the following options in the synthesis tool:

- MachXO and Platform Manager Device Mapping Options, on page 1997
- MachXO and Platform Manager Attributes, on page 1997
- MachXO and Platform Manager set_option Tcl Command, on page 1998

MachXO and Platform Manager Device Mapping Options

Device mapping options are selected from the Device tab in the Implementation Options dialog box (Project -> Implementation Options). You can set the following:

- Fanout Guide, on page 1980
- Disable I/O Insertion, on page 1980
- Retiming, on page 1984
- Pipelining, on page 1984
- GSR Resource, on page 1980
- Compile Point Remapping in Lattice Designs, on page 1982

MachXO and Platform Manager Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with MachXO, MachXO2, and Platform Manager devices.

Copyright © 2013 Synopsys, Inc.

MachXO and Platform Manager set_option Tcl Command

You can use the set_option Tcl command to specify the same device mapping options as are available through the Implementation Options dialog box (Project->Implementation Options).

This section provides information on specific options for the architectures. For a complete list of options, see *set_option*, on page 1679.

The set_option command supports the following options for MachXO, MachXO2, and Platform Manager devices.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Fanout Guide	Sets the fanout limit guideline for the open Project. The default fanout limit is 100. For information about setting fanout limits, see *Fanout Guide*, on page 1980.
Force GSR Usage	Enables (yes) or disables (no) forced usage of the global Set/Reset routing resources for the open Project. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is no. For additional information about GSR, see *GSR Resource*, on page 1980.

Copyright © 2013 Synopsys, Inc.

Option	Description
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Compile Point Remapping in Lattice Designs*, on page 1982, for details. The Synplify software does not support compile points.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for MachXO, MachXO2, and Platform Manager devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | 0 or **-disable_io_insertion 1 | 0**
-fanout_guide *value*
-force_gsr yes | no | auto
-maxfan *value*
-resolve_multiple_driver 1|0
-RWCheckOnRam 1|0
-update_models_cp 1|0

Copyright © 2013 Synopsys, Inc.

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
- **-pipe 1 | 0**
- **-resource_sharing 1|0**
- **-retiming 1 | 0**
- **-symbolic_fsm_compiler 1|0** or **-autosm 1|0**

Copyright © 2013 Synopsys, Inc.

ispGAL, ispGDX, and ispLSI Devices

This section describes guidelines for using the synthesis tool with ispGAL, ispGDX, ispGDX2, and ispLSI1K, 2K, 5K and 8K devices. The topics include:

- ispGAL, ispGDX, and ispLSI Device Mapping Options, on page 2001
- ispGAL, ispGDX, and ispLSI Attributes, on page 2001
- ispGAL, ispGDX, and ispLSI set_option Command, on page 2001

ispGAL, ispGDX, and ispLSI Device Mapping Options

You set device selection options for the ispGAL, ispGDX, and ispLSI technologies from the Device tab in the Implementation Options dialog box (Project -> Implementation Options). You can set the technology, part, speed grade, and package. You can also set:

- Disable I/O Insertion, on page 1980

ispGAL, ispGDX, and ispLSI Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with ispGAL, ispGDX, and ispLSI technologies.

ispGAL, ispGDX, and ispLSI set_option Command

You use the set_option Tcl command to set the target technology, part, speed grade, and package for the design. This command lets you specify the same device mapping options as are available in the Implementation Options dialog box (Project -> Implementation Options).

This section provides information on specific options for the ispGAL, ispGDX, and ispLSI architectures. For a complete list of options for this command refer to *set_option*, on page 1679.

Copyright © 2013 Synopsys, Inc.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is false (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for ispGAL, ispGDX, or ispLSI devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | **0** or **-disable_io_insertion 1** | **0**
-resolve_multiple_driver 1|0
-RWCheckOnRam 1|0

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
- **-resource_sharing 1|0**
- **-symbolic_fsm_compiler 1|0** or **-autosm 1|0**

Copyright © 2013 Synopsys, Inc.
2003

ispXPGA Devices

This section describes guidelines for using the synthesis tool with devices from the ispXPGA technology. The topics include:

- ispXPGA Device Mapping Options, on page 2004
- ispXPGA Attributes, on page 2004
- ispXPGA set_option Tcl Command, on page 2004
- ispXPGA LUT Function Support, on page 2006

ispXPGA Device Mapping Options

You set device mapping options for the ispXPGA technology from the Device tab in the Implementation Options dialog box (Project ->Implementation Options). You can set the following:

- Fanout Guide, on page 1980
- Disable I/O Insertion, on page 1980

ispXPGA Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with the ispXPGA technology.

ispXPGA set_option Tcl Command

You use the set_option Tcl command to set options like the target technology, device architecture, and synthesis styles. This command lets you specify the same device mapping options as are available in the Implementation Options dialog box.

This section provides information on specific options for the Lattice ispXPGA architecture. For a complete list of options for this command refer to set_option, on page 1679.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is false (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Fanout Guide	Sets the fanout limit guideline for the open Project. The default fanout limit is 500. For information about setting fanout limits, see *ispXPGA Attributes*, on page 2004.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for ispXPGA devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-disable_io_insertion 1 |0 -block 1 | 0
-fanout_guide *value*
-maxfan *value*
-resolve_multiple_driver 1|0
-RWCheckOnRam 1|0

Copyright © 2013 Synopsys, Inc.

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-resource_sharing 1|0
-symbolic_fsm_compiler 1|0 or **-autosm 1|0**

ispXPGA LUT Function Support

For ispXPGA devices, the synthesis tool supports user-instantiated LUTs. When a hexadecimal init value is provided, it is converted to a LUT function.

Copyright © 2013 Synopsys, Inc.

ispXPLD5000MX and ispLSI5000VE Devices

This section describes guidelines for using the synthesis tool with ispXPLD5000MX and ispLSI5000VE devices. The topics include:

- ispXPLD5000MX and ispLSI5000VE Device Mapping Options, on page 2007
- ispXPLD5000MX and ispLSI5000VE Attributes, on page 2007
- ispXPLD5000MX and ispLSI5000VE set_option Tcl Command, on page 2008
- ispXPLD Selective Carry-Chain Inferencing, on page 2010

ispXPLD5000MX and ispLSI5000VE Device Mapping Options

You set device mapping options for the ispXPLD5000MX and ispLSI5000VE_UPS technologies from the Device tab in the Implementation Options dialog box (Project -> Implementation Options).

- Disable I/O Insertion, on page 1980
- Map Logic to Macrocells, on page 1981
- Maximum Cell Fanin, on page 1981
- Maximum Terms/Macrocell, on page 1981
- Percentage of Design to Optimize for Timing, on page 1982

ispXPLD5000MX and ispLSI5000VE Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with the ispXPLD5000MX and ispLSI5000VE technologies.

Copyright © 2013 Synopsys, Inc.

ispXPLD5000MX and ispLSI5000VE set_option Tcl Command

You use the set_option Tcl command to set options like the target technology, device architecture, and synthesis styles. This command lets you specify the same device mapping options as are available in the Implementation Options dialog box (Project->Implementation Options).

This section provides information on specific options for the ispXPLD5000MX and ispLSI5000VE_UPS technologies. For a complete list of options for this command refer to *set_option*, on page 1679.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Map Logic	Turns on (1) or off (0) direct mapping to the macrocells cells (instead of primitive gates) during synthesis for the open project. The default value is 0, which can sometimes give a smaller result. If you set this option to true, the following three additional Tcl command options (described below) are available: -areadelay, -fanin_limit, -maxterms.
Maximum Cell Fanin	Sets the maximum fanin during synthesis for the open project. You can try to improve routability by reducing the fanin limit. The default value is 20. This option is available only if set_option -map_logic is set to 1.
Maximum Terms/Macrocell	Sets the maximum number of terms per macrocell during synthesis for the open project. The default value is 16. This option is available only if set_option -map_logic is set to 1.

Copyright © 2013 Synopsys, Inc.

Option	Description
Percent of design to optimize for timing	Sets the percentage of nets you want optimized during synthesis for the open project. This option is available only if set_option -map_logic is set to 1. The default value is 0.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for ispXPLD5000MX or ispLSI5000VE_UPS devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-areadelay *percent_value*
-fanin_limit *value*
-map_logic 1 | **0**
-maxfanin *value*
-maxterms *value*
-max_terms_per_macrocell *value*
-resolve_multiple_driver 1|0
-RWCheckOnRam 1|0

Copyright © 2013 Synopsys, Inc.
2009

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-resource_sharing 1|0
-symbolic_fsm_compiler 1|0 or **-autosm 1|0**

ispXPLD Selective Carry-Chain Inferencing

For ispXPLD devices, all counters are implemented as carry chains when greater than four bits wide. To override this option, use the syn_use_carry_chain attribute.

To force the synthesis tool to enable carry chains, the syntax is:

syn_use_carry_chain=1

To force the synthesis tool to disable carry chains, the syntax is:

syn_use_carry_chain=0

If neither of these values exist, the default behavior (enable carry chains) is used. You apply this attribute to the registers of the counter or the adder.

See *syn_use_carry_chain (Lattice)*, on page 1552 for more information.

ispMACH and MACH Devices

This section describes guidelines for using the synthesis tool with Lattice ispMACH and MACH devices. The topics include:

- ispMACH and MACH Device Mapping Options, on page 2011
- ispMACH and MACH Attributes, on page 2011
- ispMACH and MACH set_option Tcl Command, on page 2011

ispMACH and MACH Device Mapping Options

Select device mapping options for the ispMACH technology from the Device tab in the Implementation Options dialog box (select Project -> Implementation Options from the Project view to display the dialog box). You can set the following options:

- Disable I/O Insertion, on page 1980
- Map Logic to Macrocells, on page 1981
- Maximum Cell Fanin, on page 1981
- Maximum Terms/Macrocell, on page 1981
- Percentage of Design to Optimize for Timing, on page 1982

ispMACH and MACH Attributes

See *Lattice Attribute and Directive Summary*, on page 2021 for a summary of the synthesis attributes and directives you can use with ispMACH and MACH technologies.

ispMACH and MACH set_option Tcl Command

You use the set_option command to set synthesis options such as the target technology, device architecture, and synthesis styles. Through the set_option command you specify the same device mapping options as are available through the dialog box displayed in the Project view with Project -> Implementation Options.

Copyright © 2013 Synopsys, Inc.

This section provides information on specific options for the ispMACH and MACH architectures. For a complete list of options for this command, refer to *set_option*, on page 1679.

Option	Description
Disable I/O Insertion	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.
Map Logic	Turns on (1) or off (0) direct mapping to the macrocells cells (instead of primitive gates) during synthesis for the open project. The default value is 0, which can sometimes give a smaller result. If you set this option to true, the following three additional Tcl command options (described below) are available: -areadelay, -fanin_limit, -maxterms.
Maximum Cell Fanin	Sets the maximum fanin during synthesis for the open project. You can try to improve routability by reducing the fanin limit. The default value is 20. This option is available only if set_option -map_logic is set to 1.
Maximum Terms/Macrocell	Sets the maximum number of terms per macrocell during synthesis for the open project. The default value is 16. This option is available only if set_option -map_logic is set to 1.
Percent of design to optimize for timing	Sets the percentage of nets you want optimized during synthesis for the open project. This option is available only if set_option -map_logic is set to 1. The default value is 0.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

Copyright © 2013 Synopsys, Inc.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see *set_option Command for Lattice Architectures*, on page 2017.

set_option Device Options

This is the syntax for setting device options for ispMACH4000B/C/V/ZC/ZE or ispMACH5000B/VG devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-areadelay *percent_value*
-fanin_limit *value*
-map_logic 1 | **0**
-maxfanin *value*
-maxterms *value*
-max_terms_per_macrocell *value*
-resolve_multiple_driver 1|0
-RWCheckOnRam 1|0

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-resource_sharing 1|0
-symbolic_fsm_compiler 1|0 or **-autosm 1|0**

Copyright © 2013 Synopsys, Inc.
2013

Lattice iCE65 and iCE40 Devices

This section provides guidelines for using the synthesis tool with devices for the Lattice iCE65 and iCE40[1] technologies. The topics include:

- iCE65 and iCE40 Device Mapping Options, on page 2014
- iCE65 and iCE40 set_option Tcl Command, on page 2015

iCE65 and iCE40 Device Mapping Options

Device mapping options are synthesis algorithms to optimize your design. To specify these options, select Project->Implementation Options and set the options on the Device tab. You can set other options on other tabs of this dialog box. For descriptions of optimization options and constraints, see *Options Panel*, on page 197 and *Constraints Panel*, on page 200.

The following table lists device mapping options (Device tab) for the Lattice iCE65 and ICE40 technologies.

Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views, and use them to create collections using the Tcl Find command.
Clock Conversion	Performs gated- and generated-clock optimizations when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details.
Disable I/O Insertion	When enabled, it prevents automatic I/O insertion during synthesis. By default, it is disabled. See *Disable I/O Insertion Globally or Port by Port*, on page 1977 for details.
Fanout Guide	Sets a guideline for the fanout limit. The default is 100. For tips on setting and using fanout limits, see Setting Fanout Limits, on page 444 and *Controlling Buffering and Replication*, on page 446in the User Guide.

1. SiliconBlue has merged with Lattice, but complete migration of its iCE65/ iCE40 devices is still in progess. Independently run synthesis with the iCEcube2 place-and-route tool.

Copyright © 2013 Synopsys, Inc.

Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See Compile Point Synthesis Basics, on page 1098, for details.

iCE65 and iCE40 set_option Tcl Command

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles. The set_option Tcl command lets you specify the same implementation options as you do through the dialog box displayed in the Project view with Project->Implementation Options.

set_option Command Options

You can use the set_options command to specify the Tcl equivalents of the options on the Device and Options tabs of the Implementation Options dialog box. For descriptions of these Tcl options, see*set_option Command for Lattice Architectures*, on page 2017.

Copyright © 2013 Synopsys, Inc.
2015

set_option Device Options

This is the syntax for setting device options for Lattice iCE65 and iCE40 devices.

set_option -technology *keyword* **-part** *partName*
-speed_grade *value* **-package** *packageName*
-block 1 | **0** or **-disable_io_insertion 1** | **0**
-fanout_guide *value*
-maxfan *value*
-resolve_multiple_driver 1|0
-run_prop_extract 1|0
-RWCheckOnRam 1|0
-update_models_cp 1|0

set_option Synthesis Options

The following set_option arguments are the equivalents of the options on the Options tab of the Implementation Options dialog box:

set_option
-fix_gated_and_generated_clocks 1|0
-pipe 1 | **0**
-resource_sharing 1|0
-retiming 1 | **0**
-symbolic_fsm_compiler 1|0 or **-autosm 1|0**

set_option Command for Lattice Architectures

The set_option Tcl command offers an alternative way to set options that are available on the Device and Options tabs on the Implementation Options dialog box. You can save options set with this command in a file and re-execute them. For a complete list of all available set_option arguments, see *set_option*, on page 1679, or enter help set_option in a Tcl Script window

The following table describes the subset of options available for different Lattice technologies. All options are not available for all technologies.

OPTION	DESCRIPTION
Target Technology Options	
-technology *keyword*	Specifies the target technology. See *Technology Keywords*, on page 2019 for a list of valid keywords.
-part *partName*	Specifies a part for the implementation. Refer to Project->Implementation Options->Device or to the Altera documentation for available choices.
-speed_grade *value*	Sets the speed grade for the implementation. Refer to the Implementation Options dialog box for available choices.
-package *packageName*	Specifies the package. Refer to Project-> Implementation Options->Device or to the Altera documentation for available choices.
Optimization Options	
-areadelay *percent_value*	Sets the percentage of nets you want optimized during synthesis for the open project. This option is available only if set_option -map_logic is set to 1. The default value is 0.
-block 1 | 0 **-disable_io_insertion 1 | 0**	Prevents (1) or allows (0) I/O pad insertion during synthesis for the open Project. The default value is 0 (enable I/0 pad insertion). For additional information about disabling I/O pads, see *Disable I/O Insertion*, on page 1980.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-fanout_guide *value* **-maxfan** *value*	Sets the fanout limit guideline for the open Project. The default fanout limit is 100. For information about setting fanout limits, see *Fanout Guide*, on page 1980.
-force_gsr no | yes | auto	Enables (yes) or disables (no) forced usage of the global Set/Reset routing resources for the open Project. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. The default value is no. For additional information about GSR, see *GSR Resource*, on page 1980.
-fix_gated_and_ generated_clocks 1|0	Performs gated- and generated-clock optimizations when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details.
-map_logic 1 | 0	Turns on (1) or off (0) direct mapping to the macrocells cells (instead of primitive gates) during synthesis for the open project. The default value is 0, which can sometimes give a smaller result. If you set this option to true, the following three additional Tcl command options (described below) are available: -areadelay, -fanin_limit, -maxterms.
-maxfanin *value*	Sets the maximum fanin during synthesis for the open project. You can try to improve routability by reducing the fanin limit. The default value is 20. This option is available only if set_option -map_logic is set to 1.
-maxterms *value* **-max_terms_per_macrocell** *value*	Sets the maximum number of terms per macrocell during synthesis for the open project. The default value is 16. This option is available only if set_option -map_logic is set to 1.
-pipe 1 | 0	Pipelines registers located outside of a multiplier module back into the module to create pipeline stages. Default is 1 for ON. See *Pipelining*, on page 426 in the *User Guide* for a procedure. This feature is not available in the Synplify tool.
-resource_sharing 1|0	Enables or disables resource sharing. See *Sharing Resources*, on page 448 in the *User Guide* for information about using it.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-resolve_multiple_driver 1 | 0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
-retiming 1 | 0	When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). Enabling -retiming also enables -pipe. See *Retiming*, on page 430 in the *User Guide* for a procedure. This feature is not available in the Synplify tool.
-run_prop_extract 1 | 0	Determines whether the tool extracts properties for annotation. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
-update_models_cp 1 | 0	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Compile Point Remapping in Lattice Designs*, on page 1982, for details. The Synplify software does not support compile points.

Technology Keywords

The following table lists the -technology keyword for all Lattice technologies:

Family	Valid -technology Keywords
iCE65 iCE40	SBTiCE65, SBTiCE40
ispXPGA	ISPXPGA
ispGAL ispGDX ispLSI	ISPGAL, ISPGDX, ISPGDX2, PLSI1K, PLSI2K, PLSI5K, PLSI8K

Copyright © 2013 Synopsys, Inc.

Family	Valid -technology Keywords
ispXPLD5000MX ispLSI5000VE_UPS	ISPXPLD5000MX, ISPLSI5000VE_UPS
LatticeECP/EC	LATTICE-ECP3, LATTICE-ECP2S, LATTICE-ECP2MS, LATTICE-ECP2M, LATTICE-ECP2, LATTICE-ECP, LATTICE-EC
LatticeSC LatticeSCM	LATTICE-SC, LATTICE-SCM
LatticeXP	LATTICE-XP2, LATTICE-XP,
MACH ispMACH	MACH, ISPMACH4000B, ISPMACH4000C, ISPMACH4000ZC, ISPMACH4000ZE, ISPMACH5000B, ISPMACH5000VG
MACHXO/ Platform Manager	MACHXO, MACHXO2, Platform_Manager
ORCA	ORCAFPSC, ORCA2C, ORCA3C, ORCA4E

Copyright © 2013 Synopsys, Inc.

Lattice Attribute and Directive Summary

The following table summarizes the synthesis and Lattice-specific attributes and directives available with the Lattice Technology. Complete descriptions and examples can be found in Chapter 16, *Attribute and Directive Syntax*.

Attribute/Directive	Description
black_box_pad_pin	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
full_case	Specifies that a Verilog case statement has covered all possible cases.
loc	Specifies pin, register, and bus locations for Lattice I/Os.
loop_limit	Specifies a loop iteration limit for for loops.
orca_padtype	Specifies the pad type for Lattice ORCA I/Os.
orca_props	Specifies Latticed ORCA I/O properties including CLKMODE for forward annotation.
parallel_case	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
pragma translate_off/pragma translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
syn_allow_retiming	Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.
syn_black_box	Defines a black box for synthesis.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_direct_enable	Identifies which signal to use as the enable input to an enable flip-flop, when multiple candidates are possible. (LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, ORCA, and Platform Manager families)
syn_direct_reset	Controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the reset when the design has a conditional reset on multiple candidates.
syn_direct_set	Controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the set when the design has a conditional set on multiple candidates.
syn_encoding	Specifies the encoding style for state machines.
syn_enum_encoding	Specifies the encoding style for enumerated types (VHDL only).
syn_enum_encoding	Specifies the encoding style for enumerated types (VHDL only).
syn_force_pads	Enables/disables IO insertion on a port level or global level.
syn_force_seq_prim	Indicates that the fix gated clocks algorithm can be applied to the associated primitive.
syn_gatedclk_clock_en	Specifies the name of the enable pin within a black box to support fix gated clocks.
syn_gatedclk_clock_en_polarity	Indicates the polarity of the clock enable port on a black box, so that the software can apply the algorithm to fix gated clocks.
syn_global_buffers	Determines the number of global buffers available.
syn_hier	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_insert_buffer (Lattice)	Inserts a clock buffer according to the vendor-specified value.
syn_insert_pad	Inserts I/O buffers to either ports or nets.
syn_isclock	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_keep	Prevents the internal signal from being removed during synthesis and optimization.
syn_looplimit	Specifies a loop iteration limit for while loops.
syn_maxfan	Sets a fanout limit for an individual input port or register output. (LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, ORCA, and Platform Manager families)
syn_multstyle	Determines how multipliers are implemented. (LatticeECP, LatticeSC, LatticeXP, MachXO, and Platform Manager families)
syn_netlist_hierarchy	Determines if the EDIF output netlist is flat or hierarchical. (LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, ORCA, and Platform Manager families)
syn_noarrayports	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals. (LatticeECP/EC, LatticeXP, LatticeSC/SCM, MachXO, ORCA, and Platform Manager families)
syn_noclockbuf	Disables automatic clock buffer insertion.
syn_noclockpad	Infers SB_GB and SB_IO instead of SB_GB_IO on clock nets.
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.
syn_pipeline	Specifies that registers be moved into multipliers and ROMs in order to improve frequency.
syn_preserve	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_ramstyle	Determines the way in which RAMs are implemented for the LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Plaform Manager families.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
syn_reduce_controlset_size	Controls the minimum size of the unique control-set on which control-set optimizations can occur.
syn_replicate	Disables replication.
syn_romstyle (Lattice)	Determines the way in which ROMs are implemented for the LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, and Platform Manager families.
syn_romstyle Attribute (Lattice iCE65/iCE40)	Determines how ROM architectures are implemented.
syn_sharing	Specifies resource sharing of operators.
syn_srlstyle	Determines how sequential shift components are implemented.
syn_state_machine	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n>	Defines timing clock to output delay through the black box. The *n* indicates a value between 1 and 10.
syn_tpd<n>	Specifies timing propagation for combinational delay through the black box. The *n* indicates a value between 1 and 10.
syn_tristate	Specifies that a black-box pin is a tristate pin.
syn_tsu<n>	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.
syn_use_carry_chain (Lattice)	Enables/disables carry chain implementation for counter or adder.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_use_carry_chain (Lattice)	Use this attribute to turn on or off carry chain implementation for arithmetic and combinational operators.
syn_useenables	For CPLD technologies, generates register instances with clock enable pins for selective clock-enable inference.
syn_useioff (Lattice)	Packs flip-flops in the I/O ring to improve input/output path timing in LatticeECP/EC, LatticeSC/SCM, LatticeXP, MachXO, ORCA, and Platform Manager families.
syn_useioff (Lattice iCE65/iCE40)	Controls register packing in I/Os.
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

APPENDIX D

Designing with Microsemi

The following topics describe how to design and synthesize with the Microsemi technology:

- Basic Support for Microsemi Designs, on page 2028
- Microsemi Components, on page 2031
- Output Files and Forward-annotation for Microsemi, on page 2042
- Optimizations for Microsemi Designs, on page 2044
- Integration with Microsemi Tools and Flows, on page 2053
- Microsemi Device Mapping Options, on page 2054
- Microsemi Tcl set_option Command Options, on page 2056
- Microsemi Attribute and Directive Summary, on page 2059

Copyright © 2013 Synopsys, Inc.

Basic Support for Microsemi Designs

This section describes the use of the synthesis tool with Microsemi devices. It describes

- Microsemi Features, on page 2029
- Microsemi Features, on page 2029
- Synthesis Constraints and Attributes for Microsemi, on page 2029

Microsemi Device-specific Support

The synthesis tool creates technology-specific netlists for a number of Microsemi families of FPGAs. New devices are added on an ongoing basis. For the most current list of supported devices, check the Device panel of the Implementation Options dialog box (see *Device Panel*, on page 195).

The following technologies are supported:

Mixed-Signal FPGAs

- SmartFusion2 and SmartFusion
- Fusion

Low-Power FPGAs

- IGLOO Series (IGLOO, IGLOO+, and IGLOOE)
- ProASIC3 Series (ProASIC3, ProASIC3E, and ProASIC3L)

Antifuse FPGAs

- Axcelerator
- eX

Legacy FPGAs

- ACT1, ACT2/1200L, and ACT3
- ProASIC^{PLUS}
- 3200DX, 40MX, 42MX, 54SX, and 54SXA

Copyright © 2013 Synopsys, Inc.

After synthesis, the synthesis tool generates EDIF netlists as well as a constraint file that is forward annotated as input into the Microsemi Libero tool.

Microsemi Features

The synthesis tool contains the following Microsemi-specific features:

- Direct mapping to Microsemi c-modules and s-modules
- Timing-driven mapping, replication, and buffering
- Inference of counters, adders, and subtractors; module generation
- Automatic use of clock buffers for clocks and reset signals
- Automatic I/O insertion. See *I/O Insertion*, on page 2047 for more information.

Synthesis Constraints and Attributes for Microsemi

The synthesis tools let you specify timing constraints, general HDL attributes, and Microsemi-specific attributes to improve your design. You can manage the attributes and constraints in the SCOPE interface. Microsemi has vendor-specific I/O standard constraints it supports for synthesis. For a list of supported I/O standards, see *Microsemi I/O Standards*, on page 2029.

Microsemi I/O Standards

The following table lists the supported I/O standards for the Fusion, IGLOO, ProASIC3, ProASIC, ProASIC^{PLUS}, and Axcelerator technology families. Some I/O standards have associated modifiers you can set, such as slew, termination, drive, power, and Schmitt, which allow the software to infer the correct buffer types.

Copyright © 2013 Synopsys, Inc.

ProASIC3L, ProASIC3E, IGLOOe, Fusion	IGLOO, IGLOO+, ProASIC3	ProASIC, ProASIC^{PLUS}	Axcelerator
GTL25	LVCMOS_12	LVCMOS_25	GTL+25
GTL+25	LVCMOS_15	LVCMOS_33	GTL+33
GTL33	LVCMOS_18	LVTTL	HSTL_Class_I
GTL+33	LVCMOS_33	PCI33	HSTL_Class_II
HSTL_Class_I	LVCMOS_5		LVCMOS_15
HSTL_Class_II	LVDS		LVCMOS_18
LVCMOS_12	LVPECL		LVCMOS_5
LVCMOS_15	LVTTL		LVDS
LVCMOS_18	PCI		LVPECL
LVCMOS_33	PCIX		LVTTL
LVCMOS_5			PCI
LVDS			PCIX
LVPECL			SSTL_2_Class_I
LVTTL			SSTL_2_Class_II
PCI			SSTL_3_Class_I
PCIX			SSTL_3_Class_II
SSTL_2_Class_I			
SSTL_2_Class_II			
SSTL_3_Class_I			
SSTL_3_Class_II			

See Also:

- *Industry I/O Standards*, on page 496 for a list of industry I/O standards.
- *Microsemi Attribute and Directive Summary*, on page 2059 for a list of Microsemi attributes and directives.

Microsemi Components

The following topics describe how the synthesis tools handle various Microsemi components, and show you how to work with or manipulate them during synthesis to get the results you need:

- Macros and Black Boxes in Microsemi Designs, on page 2031
- DSP Block Inference, on page 2033
- Microsemi RAM Implementations, on page 2036
- Instantiating RAMs with SYNCORE, on page 2041

Macros and Black Boxes in Microsemi Designs

You can instantiate Smartgen[1] macros or other Microsemi macros like gates, counters, flip-flops, or I/Os by using the supplied Microsemi macro libraries to pre-define the Microsemi macro black boxes. For certain technologies, the following macros are also supported:

- SIMBUF Macro
- MATH18X18 Block
- Microsemi Fusion Analog Blocks
- SmartFusion Macros
- SmartFusion2 MACC Block

For general information on instantiating black boxes, see *Instantiating Black Boxes in VHDL*, on page 961, and *Instantiating Black Boxes in Verilog*, on page 749. For specific procedures about instantiating macros and black boxes and using Microsemi black boxes, see the following sections in the *User Guide*:

- *Defining Black Boxes for Synthesis*, on page 330
- *Using Predefined Microsemi Black Boxes*, on page 789
- *Using Smartgen Macros*, on page 790

1. Smartgen macros now replace the ACTgen macros. ACTgen macros were available in the previous Designer 6.x place-and-route tool.

Copyright © 2013 Synopsys, Inc.

SIMBUF Macro

The synthesis software supports instantiation of the SIMBUF macro. The SIMBUF macro provides the flexibility to probe signals without using physical locations, such as possible from the Identify tool. The Resource Summary will report the number of SIMBUF instantiations in the IO Tile section of the log file.

SIMBUF macros are supported for ProASIC3, ProASIC3E, ProASIC3L, IGLOO, IGLOOe, IGLOO+, SmartFusion, Fusion, and Axcelerator devices.

MATH18X18 Block

The synthesis software supports instantiation of the MATH18X18 block. The MATH18X18 block is useful for mapping arithmetic functions for RTAXSDSP devices.

Microsemi Fusion Analog Blocks

Microsemi Fusion has several analog blocks built into the ProASIC3/3E device. The synthesis tool treats them as black boxes. The following is a list of the available analog blocks.

- AB
- NVM
- XTLOSC
- RCOSC
- CLKSRC
- NGMUX
- VRPSM
- INBUF_A
- INBUF_DA
- OUTBUF_A
- CLKDIVDLY
- CLKDIVDLY1

Copyright © 2013 Synopsys, Inc.

SmartFusion Macros

The synthesis software supports the following SmartFusion macros:

- FAB_CCC
- FAB_CCC_DYN

SmartFusion2 MACC Block

SmartFusion2 devices support bit-signed 18x18 multiply-accumulate blocks. This architecture provides dedicated components called SmartFusion2 MACC blocks, for which DSP-related operations can be performed like multiplication followed by addition, multiplication followed by subtraction, and multiplication with accumulate. For more information, see *DSP Block Inference*, on page 2033.

DSP Block Inference

This feature allows the synthesis tools to infer DSP or MATH18x18 blocks for SmartFusion2, Axcelerator RTAX2000D and RTAX4000D devices only. The following structures are supported:

- Multipliers
- Mult-adds — Multiplier followed by an Adder
- Mult-subs — Multiplier followed by a Subtractor
- Wide multiplier inference

  A multiplier is treated as wide, if any of its inputs is larger than 18 bits signed or 17 bits unsigned. The multiplier can be configured with only one input that is wide, or else both inputs are wide. Depending on the number of wide inputs for signed or unsigned multipliers, the synthesis software uses the cascade feature to determine how many math blocks to use and the number of Shift functions it needs.
- MATH block inferencing across hierarchy

  This enhancement to MATH block inferencing allows packing input registers, output registers, and any adders or subtractors into different hierarchies. This helps to improve QoR by packing logic more efficiently into MATH blocks.

Copyright © 2013 Synopsys, Inc.

By default, the synthesis software maps the multiplier to DSP blocks if all inputs to the multiplier are more than 2-bits wide; otherwise, the multiplier is mapped to logic. You can override this default behavior using the syn_multstyle attribute. See syn_multstyle, on page 1404 for details.

The following conditions also apply:

- Signed and unsigned multiplier inferencing is supported.
- Registers at inputs and outputs of multiplier/multiplier-adder/multiplier-subtractor are packed into DSP blocks.
- Synthesis software fractures multipliers larger than 18X18 (signed) and 17X17 (unsigned) into smaller multipliers and packs them into DSP blocks.
- When multadd/multsub are fractured, the final adder/subtractor are packed into logic.

DSP Cascade Chain Inference

The MATH18x18 block cascade feature supports the implementation of multi-input Mult-Add/Sub for devices with MATH blocks. The software packs logic into MATH blocks efficiently using hard-wired cascade paths, which improves the QoR for the design.

Prerequisites include the following requirements:

- The input size for multipliers is *not* greater than 18x18 bits (signed) and 17x17 bits (unsigned).
- Signed multipliers have the proper sign-extension.
- All multiplier output bits feed the adder.
- Multiplier inputs and outputs can be registered or not.

Multiplier-Accumulators (MACC) Inference

The Multiplier-Accumulator structures use internal paths for adder feedback loops inside the MATH18x18 block instead of connecting it externally.

Prerequisites include the following requirements:

- The input size for multipliers is *not* greater than 18x18 bits (signed) and 17x17 bits (unsigned).
- Signed multipliers have the proper sign-extension.

Copyright © 2013 Synopsys, Inc.

- All multiplier output bits feed the adder.
- The output of the adder must be registered.
- The registered output of the adder feeds back to the adder for accumulation.
- Since the Microsemi MATH block contains one multiplier, only Multiplier-Accumulator structures with one multiplier can be packed inside the MATH block.

The other Multiplier-Accumulator structure supported is with Synchronous Loadable Register.

Prerequisites include the following requirements:

- All the requirements mentioned above apply for this structure as well.
- For the Loading Multiplier-Accumulator structure, new Load data should be passed to input C.
- The LoadEn signal should be registered.

DSP Limitations

Currently, DSP inferencing does not support the following functions:

- Overflow extraction
- Arithmetic right shift for operand C
- Dynamic Mult-AddSub

Note: For more information about Microsemi DSP math blocks along with a comprehensive set of examples, see the *Inferring Microsemi RTAX-DSP MATH Blocks* application note on SolvNet.

Copyright © 2013 Synopsys, Inc.

Microsemi RAM Implementations

Refer to the following topics for Microsemi RAM implementations:

- RAM Read Enable Extraction
- ProASIC3/3E/3L and IGLOO+/IGLOO/IGLOOe
- ProASIC (500K) and ProASICPLUS (PA)
- Axcelerator
- SmartFusion2

RAM Read Enable Extraction

RAM Read Enable extraction currently supports RAMs for output registers, with an enable. This feature is available for ProASIC3E and Axcelerator devices only.

The following example contains a RAM with read enable.

```
`timescale 100 ps/100 ps
/* Synchronous write and read RAM */

module test (dout, addr, din, we, clk, ren);

parameter data_width = 8;
parameter address_width = 4;
parameter ram_size = 16;

output [data_width-1:0] dout;
input [data_width-1:0] din;
input [address_width-1:0] addr;
input we, clk, ren;

reg [data_width-1:0] mem [ram_size-1:0];
reg [data_width-1:0] dout;

always @(posedge clk) begin
   if(we)
      mem[addr] <= din;
      if (ren)
      dout = mem[addr];
end

endmodule
```

Copyright © 2013 Synopsys, Inc.

ProASIC3/3E/3L and IGLOO+/IGLOO/IGLOOe

The synthesis software extracts single-port and dual-port versions of the following RAM configurations:

RAM4K9	Synchronous write, synchronous read, transparent output
RAM512X18	Synchronous write, synchronous read, registered output

The architecture of the inferred RAM for the ProASIC3/3E/3L or IGLOO+/IGLOO/IGLOOe, can be registers, block_ram, rw_check, or no_rw_check. You set these values in the SCOPE interface using the syn_ramstyle attribute.

The following is an example of the RAM4K9 configuration:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
port (q : out std_logic_vector(9 downto 0);
   d : in std_logic_vector(9 downto 0);
   addr : in std_logic_vector(9 downto 0);
   we : in std_logic;
   clk : in std_logic);
end ramtest;

architecture rtl of ramtest is
```


Copyright © 2013 Synopsys, Inc.

```
type mem_type is array (1023 downto 0) of std_logic_vector (9
downto 0);

signal mem : mem_type;
signal read_addr : std_logic_vector(9 downto 0);

begin

q <= mem(conv_integer(read_addr));

process (clk) begin
   if rising_edge(clk) then
      if (we = '1') then
         read_addr <= addr;
         mem(conv_integer(read_addr)) <= d;
      end if;
   end if;
end process;

end rtl;
```

The following is an example of the RAM512X18 configuration:

```
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;
signal read_addr : std_logic_vector(2 downto 0);
begin
q <= mem(conv_integer(read_addr));
process (clk) begin
   if rising_edge(clk) then
      if (we = '1') then
         read_addr <= addr;
         mem(conv_integer(read_addr)) <= d;
      end if;
   end if;
end process;
end rtl;
```

Copyright © 2013 Synopsys, Inc.

ProASIC (500K) and ProASIC^{PLUS} (PA)

The synthesis software extracts single-port and dual-port versions of the following RAM configurations:

SA	Synchronous write, asynchronous read
SST	Synchronous write, synchronous read, transparent output
SSR	Synchronous write, synchronous read, registered output

Axcelerator

The synthesis software extracts single-port and dual-port versions of the following RAM configurations. The memory blocks operate in synchronous mode for both read and write operations. There are two read modes and one write mode:

Read Nonpipelined	Synchronous – one clock edge
Read Pipelined	Synchronous – two clock edges
Write	Synchronous – one clock edge

The architecture of the inferred RAM for the Axcelerator device can be registers, block_ram rw_check, or no_rw_check. You set these values in the SCOPE interface using the syn_ramstyle attribute.

SmartFusion2

SmartFusion2 devices support two types of RAM macros: RAM1K18 and RAM64X18. The synthesis software extracts the RAM structure from the RTL and infers RAM1K18 or RAM64X18 based on the size of the RAM.

Copyright © 2013 Synopsys, Inc.

The default criteria for specifying the macro is described in the table below for the following RAM types.

True Dual-Port Synchronous Read Memory	The synthesis tool maps to RAM1K18, regardless of its memory size.
Simple Dual-Port or Single-Port Synchronous Memory	If the size of the memory is: • 4608 bits or greater, the synthesis tool maps to RAM1K18. • Greater than 12 bits and less than 4608 bits, the synthesis tool maps to RAM64X18. • Less than or equal to 12 bits, the synthesis tool maps to registers.
Simple Dual-Port or Single-Port Asynchronous Memory	When the size of the memory is 12 bits or greater, the synthesis tool maps to RAM64x18. Otherwise, it maps to registers.

You can override the default behavior by applying the syn_ramstyle attribute to control how the memory gets mapped. To map to

- RAM1K18 set syn_ramstyle = "lsram"
- RAM64X18 set syn_ramstyle = "uram"
- Registers set syn_ramstyle = "registers"

The value you set for this attribute always overrides the default behavior.

Instantiating RAMs with SYNCORE

The SYNCORE Memory Compiler is available under the IP Wizard to help you generate HDL code for your specific RAM implementation requirements. For information on using the SYNCORE Memory Compiler, see *Specifying RAMs with SYNCore*, on page 939 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

Output Files and Forward-annotation for Microsemi

After synthesis, the software generates a log file and output files for Microsemi. The following describe some of the reports with Microsemi-specific information, or files that forward-annotate information for the Microsemi P&R tools.

- Forward-annotating Constraints for Placement and Routing, on page 2042
- Synthesis Reports, on page 2043
- Point-to-Point Timing Analysis, on page 2044

Forward-annotating Constraints for Placement and Routing

For Microsemi Axcelerator, Fusion, IGLOO, ProASIC (500K), ProASIC^{PLUS} (PA), ProASIC3, and SmartFusion technology families, the synthesis tool forward-annotates timing constraints to placement and routing through an Microsemi constraint (*filename*_sdc.sdc) file. These timing constraints include clock period, max delay, multiple-cycle paths, input and output delay, and false paths. During synthesis, the Microsemi constraint file is generated using synthesis tool attributes and constraints.

By default, Microsemi constraint files are generated. You can disable this feature in the Project view. To do this, bring up the Implementation Options dialog box (Project -> Implementation Options), then, on the Implementation Results panel, disable Write Vendor Constraint File.

Forward-annotated Constraints

The constraint file generated for Microsemi's place-and-route tools has an _sdc.sdc file extension. Constraints files that properly specify either Synplify-style timing constraints or Synopsys SDC timing constraints can be forward annotated to support the Microsemi P&R tool.

Copyright © 2013 Synopsys, Inc.

Synthesis	Microsemi
define_clock	create_clock The create_clock constraint is allowed for all NGT families. No wildcards are accepted. The pin or port must exist in the design. The -name argument is not supported as that would define a virtual clock. However, for backward compatibility, a -name argument does not generate an error or warning when encountered in an .sdc file.
define_path_delay	set_max_delay The set_max_delay constraint is allowed for all NGT families, however, it will not drive ProASIC (500K) TDPR because of conversion issues to GCF (the whole path is needed in GCF, not the start and end point). Wildcards are accepted.
define_multicycle_path	set_multicycle_path You must specify at least one of the -from or -to arguments, however, it is best to specify both. Wildcards are accepted. Multicycle constraints with -from and/or -to arguments only are supported for Microsemi Axcelerator, ProASIC, ProASIC^{PLUS}, and ProASIC3/3E technologies. Multicycle constraints with a -through argument are not supported for any NGT family.
define_false_path	set_false_path Only false path constraints with a -through argument are supported for NGT families. However, it might not drive ProASIC (500K) TDPR because of conversion issues to GCF (some non supported objects). False path constraints with either -from and/or -to arguments are not supported for any NGT family. Wildcards are accepted.

Synthesis Reports

The synthesis tool generates a resource usage report, a timing report, and a net buffering report for the Microsemi designs that you synthesize.To view the synthesis reports, click View Log.

Copyright © 2013 Synopsys, Inc.

Point-to-Point Timing Analysis

For the Axcelerator technology, you can do point-to-point timing analysis using the ( ) icon or the Analysis->Timing Analyst command. For details about using the Timing Analyst to generate point-to-point views and timing reports, see *Generating Custom Timing Reports with STA*, on page 729 in the *User Guide*.

Optimizations for Microsemi Designs

The synthesis tools offer various optimizations for Microsemi designs. The following describe the optimizations in more detail:

- The syn_maxfan Attribute in Microsemi Designs, on page 2045
- Promote Global Buffer Threshold, on page 2045
- I/O Insertion, on page 2047
- Number of Critical Paths, on page 2047
- Retiming, on page 2047
- Update Compile Point Timing Data Option, on page 2048
- Operating Condition Device Option, on page 2049
- Radiation-tolerant Applications, on page 2051

Copyright © 2013 Synopsys, Inc.

The syn_maxfan Attribute in Microsemi Designs

The syn_maxfan attribute is used to control the maximum fanout of the design, or an instance, net, or port. The limit specified by this attribute is treated as a hard or soft limit depending on where it is specified. The following rules described the behavior:

- Global fanout limits are usually specified with the fanout guide options (Project->Implementation Options->Device), but you can also use the syn_maxfan attribute on a top-level module or view to set a global soft limit. This limit may not be honored if the limit degrades performance. To set a global hard limit, you must use the Hard Limit to Fanout option.
- A syn_maxfan attribute can be applied locally to a module or view. In this case, the limit specified is treated as a soft limit for the scope of the module. This limit overrides any global fanout limits for the scope of the module.
- When a syn_maxfan attribute is specified on an instance that is not of primitive type inferred by Synopsys FPGA compiler, the limit is considered a soft limit which is propagated down the hierarchy. This attribute overrides any global fanout limits.
- When a syn_maxfan attribute is specified on a port, net, or register (or any primitive instance), the limit is considered a hard limit. This attribute overrides any other global fanout limits. Note that the syn_maxfan attribute does not prevent the instance from being optimized away and that design rule violations resulting from buffering or replication are the responsibility of the user.

Promote Global Buffer Threshold

The Promote Global Buffer Threshold option is for the SmartFusion, Fusion, IGLOO, and ProASIC3 technology families only. This option is for both ports and nets.

The Tcl command equivalent is set_option -globalthreshold *value,* where the value refers to the minimum number of fanout loads. The default value is 1.

Copyright © 2013 Synopsys, Inc.

Only signals with fanout loads larger than the defined value are promoted to global signals. The synthesis tool assigns the available global buffers to drive these signals using the following priority:

1. Clock
2. Asynchronous set/reset signal
3. Enable, data

SmartFusion2 Global Buffer Promotion

The synthesis software inserts the global buffer (CLKINT) on clock, asynchronous set/reset, and data nets based on a threshold value. SmartFusion2 devices have specific threshold values that cannot be changed for the different types of nets in the design. Inserting global buffers on nets with fanout greater than the threshold can help reduce the route delay during place and route.

The threshold values for SmartFusion2 devices are the following:

Net	Global buffer inserted for threshold value > or =
Clock	2
Asynchronous Set/Reset	12
Data	50000

To override these default option settings you can:

- Use the syn_noclockbuf attribute on a net that you do not want a global buffer inserted, even though fanout is greater than the threshold.
- Use syn_insert_buffer="CLKINC" so that the tool inserts a global buffer on the particular net, which is less than the threshold value. You can only specify CLKINC as a valid value for SmartFusion2 devices.

Copyright © 2013 Synopsys, Inc.

I/O Insertion

The Synopsys FPGA synthesis tool inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist unless you disable I/O insertion. You can control I/O insertion with the Disable I/O Insertion option (Project->Implementation Options->Device).

If you do not want to automatically insert any I/O pads, check the Disable I/O Insertion box (Project->Implementation Options->Device). This is useful to see how much area your blocks of logic take up, before synthesizing an entire FPGA. If you disable automatic I/O insertion, you will not get any I/O pads in your design unless you manually instantiate them yourself.

If you disable I/O insertion, you can instantiate the Microsemi I/O pads you need directly. If you manually insert I/O pads, you only insert them for the pins that require them.

Number of Critical Paths

The Max number of critical paths in SDF option (Project->Implementation Options->Device) is only available for the SmartFusion, Fusion, IGLOO, ProASIC3, ProASIC (500K), and ProASIC^{PLUS} (PA) technology families. It lets you set the maximum number of critical paths in a forward-annotated constraint (`sdf`) file. The `sdf` file displays a prioritized list of the worst-case paths in a design. Microsemi Designer prioritizes routing to ensure that the worst-case paths are routed efficiently.

The default value for the number of critical paths that are forward-annotated is 4000. Various design characteristics affect this number, so experiment with a range of values to achieve the best circuit performance possible.

Retiming

Retiming is the process of automatically moving registers (register balancing) across combinational gates to improve timing, while ensuring identical logic behavior. Currently retiming is available for the SmartFusion, Fusion, IGLOO, Axcelerator, ProASIC3, ProASIC (500K), and ProASIC^{PLUS} (PA) technology families. This feature is not available for the Synplify tool.

Copyright © 2013 Synopsys, Inc.

You enable/disable global retiming with the Retiming device mapping option (Project view or Device panel). You can use the syn_allow_retiming attribute to enable or disable retiming for individual flip-flops. See *syn_allow_retiming*, on page 1216 and the *Synplify User Guide* for more information.

Update Compile Point Timing Data Option

The Update Compile Point Timing Data option is not available with the Synplify tool. In SmartFusion, Fusion, IGLOO, ProASIC3, ProASIC (500K), and ProASIC^{PLUS} (PA) design families, the Synopsys FPGA compile-point synthesis flow lets you break down a design into smaller synthesis units, called *compile points*, making incremental synthesis possible. See *Synthesizing Compile Points*, on page 1109 in the *User Guide*.

The Update Compile Point Timing Data option controls whether or not changes to a locked compile point force remapping of its parents, taking into account the new timing model of the child.

Note: To simplify this description, the term *child* is used here to refer to a compile point that is contained inside another; the term *parent* is used to refer to the compile point that contains the child. These terms are thus not used here in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points.

Disabled

When the Update Compile Point Timing Data option is *disabled* (the default), only (locked) compile points that have changed are remapped, and their remapping does *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used, instead, to map and optimize its parents.

An exceptional case occurs when the option is disabled and the *interface* of a locked compile point is changed. Such a change requires that the immediate parent of the compile point be changed accordingly, so both are remapped. In this exceptional case, however, the *updated* timing model (not the old model) of the child is used when remapping this parent.

Copyright © 2013 Synopsys, Inc.

Enabled

When the Update Compile Point Timing Data option is *enabled*, locked compile-point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account (because the option was disabled).

The timing model of a compile point is updated when either of the following is true:

- The compile point is remapped, and the Update Compile Point Timing Data option is enabled.
- The interface of the compile point is changed.

Operating Condition Device Option

You can specify an operating condition for certain Microsemi technologies:

- ProASIC (500K)
- ProASIC^{PLUS} (PA), ProASIC3/3E/3L
- IGLOO+/IGLOO/IGLOOe
- SmartFusion, FUSION
- Axcelerator

Different operating conditions cause differences in device performance. The operating condition affects the following:

- optimization, if you have timing constraints
- timing analysis
- timing reports

To set an operating condition, select the value for Operating Conditions from the menu on the Device tab of the Implementation Options dialog box.

Copyright © 2013 Synopsys, Inc.

To set an operating condition in a project or Tcl file, use the command:

set_option -opcond *value*

where *value* can be specified like the following typical operating conditions:

Default	Typical timing
MIL-WC	Worst-case Military timing
MIL-TC	Typical-case Military timing
MIL-BC	Best-case Military timing
Automotive-WC	Worst-case Automotive timing

For Example

The Microsemi operating condition can contain any of the following specifications:

- MIL—military
- COM—commercial

- IND—Industrial
- TGrade1
- TGrade2

as well as, include one of the following designations:

- WC—worst case
- BC—best case
- TC—typical case

For specific operating condition values for your required technology, see the Device tab on the Implementation Options dialog box.

Note that even when a particular operating condition is valid for a family, it may not be applicable to every part/package/speed-grade combination in that family. Consult Microsemi's documentation or software for information on valid combinations and more information on the meaning of each operating condition.

Radiation-tolerant Applications

You can specify the radiation-resistant design technique to use on an object for a design with the syn_radhardlevel attribute. This attribute can be applied to a module/architecture or a register output signal (inferred register in VHDL), and is used in conjunction with the Microsemi macro files supplied with the software.

Values for syn_radhardlevel are as follows:

Value	Description
none	Standard design techniques are used.
cc	Combinational cells with feedback are used to implement storage rather than flip-flop or latch primitives.
tmr	Triple module redundancy or triple voting is used to implement registers. Each register is implemented by three flip-flops or latches that “vote” to determine the state of the register.
tmr_cc	Triple module redundancy is used where each voting register is composed of combinational cells with feedback rather than flip-flop or latch primitives

Copyright © 2013 Synopsys, Inc.

For details, see:

- *Working with Radhard Designs*, on page 790
- *syn_radhardlevel*, on page 1462

Integration with Microsemi Tools and Flows

The following describe how the synthesis tools support various tools and flows for Microsemi designs:

- Compile Point Synthesis, on page 2053
- Microsemi Place-and-Route Tools, on page 2054

Compile Point Synthesis

Compile-point synthesis is available for the Synplify Pro tool and only for the Microsemi SmartFusion, Fusion, IGLOO, Axcelerator, ProASIC3, ProASIC (500K), and ProASIC^{PLUS} (PA) technology families. When you want to isolate portions of a design in order to stabilize results and/or improve runtime performance during placement and routing, the Synopsys FPGA The compile-point synthesis flow lets you achieve incremental design and synthesis without having to write and maintain sets of complex, error-prone scripts to direct synthesis and keep track of design dependencies. See *Synthesizing Compile Points*, on page 1109 for a description, and *Working with Compile Points*, on page 1087 in the *User Guide* for a step-by-step explanation of the compile-point synthesis flow.

In device technologies that can take advantage of compile points, you break down your design into smaller synthesis units or *compile points*, in order to make incremental synthesis possible. A compile point is a module that is treated as a block for incremental mapping: When your design is resynthesized, compile points that have already been synthesized are not resynthesized, unless you have changed:

- the HDL source code in such a way that the design logic is changed,
- the constraints applied to the compile points, or
- the device mapping options used in the design.

(For details on the conditions that necessitate resynthesis of a compile point, see *Compile Point Basics*, on page 1088, and *Update Compile Point Timing Data Option*, on page 2048.)

Copyright © 2013 Synopsys, Inc.

Microsemi Place-and-Route Tools

You can run place and route automatically after synthesis. For details on how to set options, see *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide*.

For details about the place-and-route tools, refer to the Microsemi documentation.

Microsemi Device Mapping Options

You select device mapping options for Microsemi technologies, select Project -> Implementation Options->Device and set the options.

Option	For details, see...
Conservative Register Optimization	See the *Microsemi Tcl set_option Command Options*, on page 2056 for more information about the preserve_registers option.
Disable I/O Insertion	*I/O Insertion*, on page 2047.
Fanout Guide	*Setting Fanout Limits*, on page 444 of the *User Guide* and *The syn_maxfan Attribute in Microsemi Designs*, on page 2045.
Hard Limit to Fanout	*Setting Fanout Limits*, on page 444 of the *User Guide* and *The syn_maxfan Attribute in Microsemi Designs*, on page 2045.
Max number of critical paths in SDF (certain technologies)	*Number of Critical Paths*, on page 2047.
Operating Conditions (certain technologies)	*Operating Condition Device Option*, on page 2049
Promote Global Buffer Threshold	*Controlling Buffering and Replication*, on page 446 of the *User Guide* and *Promote Global Buffer Threshold*, on page 2045.

Copyright © 2013 Synopsys, Inc.

Option	For details, see...
Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Retiming (certain technologies; not available for the Synplify tool)	*Retiming*, on page 430 of the *User Guide and Retiming*, on page 2047.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Update Compile point Timing Data (certain technologies; not available for the Synplify tool)	*Update Compile Point Timing Data Option*, on page 2048

Copyright © 2013 Synopsys, Inc.

Microsemi Tcl set_option Command Options

You can use the set_option Tcl command to specify the same device mapping options as are available through the Implementation Options dialog box displayed in the Project view with Project -> Implementation Options (see *Configure Revision Control Tools Command*, on page 184).

This section describes the Microsemi-specific set_option Tcl command options. These include the target technology, device architecture, and synthesis styles.

The table below provides information on specific options for Microsemi architectures. For a complete list of options for this command, refer to *set_option*, on page 1679. You cannot specify a package (-package option) for some Microsemi technologies in the synthesis tool environment. You must use the Microsemi back-end tool for this.

Option	Description
-technology *keyword*	Sets the target technology for the implementation. Keyword must be one of the following Microsemi architecture names: 3200DX, 40MX, 42MX, 500K, 54SX, 54SXA, ACT1, ACT2, ACT3, AXCELERATOR, EX, FUSION, IGLOO, IGLOOE, IGLOO+, PA, ProASIC3, ProASIC3E, ProASIC3L, SmartFusion, and SmartFusion2. For the 1200XL architecture, use ACT2.
-part *partName*	Specifies a part for the implementation. Refer to the Implementation Options dialog box for available choices.
-package *packageName*	Specifies the package. Refer to Project-> Implementation Options->Device for available choices. The Package option is available only for the Axcelerator, Fusion, IGLOO, IGLOOe, IGLOO+, ProASIC3, ProASIC3E, ProASIC3L, and SmartFusion devices.
-speed_grade *value*	Sets the speed grade for the implementation. Refer to the Implementation Options dialog box for available choices. This option is not supported by the ProASIC (500K) technology.

Copyright © 2013 Synopsys, Inc.

Option	Description
-disable_io_insertion 1 | 0	Prevents (1) or allows (0) insertion of I/O pads during synthesis. The default value is false (enable I/0 pad insertion). For additional information about disabling I/O pads, see *I/O Insertion*, on page 2047.
-fanout_guide *value*	Sets the fanout limit guideline for the current project. If you want to set a hard limit, you must also set the -maxfan_hard option to true. For more information about fanout limits, see *The syn_maxfan Attribute in Microsemi Designs*, on page 2045.
-globalthreshold *value*	Sets the minimum fanout load value. This option applies only to the SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, and ProASIC3/3E/3L technologies. For more information, see *Promote Global Buffer Threshold*, on page 2045.
-maxfan_hard 1	Specifies that the specified -fanout_guide value is a hard fanout limit that the synthesis tool must not exceed. To set a guideline limit, see the -fanout_guide option. For more information about fanout limits, see *The syn_maxfan Attribute in Microsemi Designs*, on page 2045.
-opcond *value*	Sets the operating condition for device performance in the areas of optimization, timing analysis, and timing reports. This option applies only to the SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, ProASIC (500K), ProASIC^{PLUS} (PA), and ProASIC3/3E/3L technologies. Values are Default, MIL-WC, IND-WC, COM-WC, and Automotive-WC. See *Operating Condition Device Option*, on page 2049 for more information.
-preserve_registers 1|0	When enabled, the software uses less restrictive register optimizations during synthesis if area is not as great a concern for your device. The default for this option is disabled (0).
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

Copyright © 2013 Synopsys, Inc.

Option	Description
-report_path *value*	Sets the maximum number of critical paths in a forward-annotated SDF constraint file. This option applies only to the SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, ProASIC (500K), ProASIC^{PLUS} (PA), and ProASIC3/3E/3L technologies. For information about setting critical paths, see *Number of Critical Paths*, on page 2047.
-retiming 1 | 0	When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). This option is only available for the Axcelerator, SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, ProASIC (500K), ProASIC^{PLUS} (PA), and ProASIC3/3E/3L technologies and is not available for the Synplify tool. For additional information about retiming, see *Retiming*, on page 2047
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
-update_models_cp 1 | 0	Determines whether (1) or not (0) changes to a locked compile point force remapping of its parents, taking into account the new timing model of the child. This option is only available for the SmartFusion, Fusion, IGLOO+/IGLOO/IGLOOe, ProASIC (500K), ProASIC^{PLUS} (PA), and ProASIC3/3E/3L technologies and is not available for the Synplify tool. See *Update Compile Point Timing Data Option*, on page 2048, for details.

Copyright © 2013 Synopsys, Inc.

Microsemi Attribute and Directive Summary

The following table summarizes the synthesis and Microsemi-specific attributes and directives available with the Microsemi technology. Complete descriptions and examples are in Chapter 16, *Attribute and Directive Syntax*.

Attribute/Directive	Description
alsloc	Forward-annotates the relative placements of macros and IP blocks to Microsemi Designer. This attribute does not apply to ProASIC (500K) and ProASIC^{PLUS} (PA) designs.
alspin	Assigns scalar or bus ports to Microsemi I/O pin numbers. This attribute does not apply to ProASIC (500K) and ProASIC^{PLUS} (PA) designs.
alspreserve	Specifies that a net be preserved, and prevents it from being removed during place-and-route optimization. This attribute does not apply to ProASIC (500K) and ProASIC^{PLUS} (PA) designs.
black_box_pad_pin (D)	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins (D)	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
full_case (D)	Specifies that a Verilog case statement has covered all possible cases.
loop_limit (D)	Specifies a loop iteration limit for for loops.
parallel_case (D)	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
syn_allow_retiming	Specifies whether registers can be moved during retiming.
syn_black_box (D)	Defines a black box for synthesis.

(D) indicates directives; all others are attributes.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_direct_enable	Assigns clock enable nets to dedicated flip-flop enable pins. It can also be used as a compiler directive that marks flip-flops with clock enables for inference.
syn_encoding	Specifies the encoding style for state machines.
syn_enum_encoding (D)	Specifies the encoding style for enumerated types (VHDL only).
syn_global_buffers	Sets the number of global buffers to use in a ProASIC3/3E/3L.
syn_hier	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.
syn_insert_buffer (Microsemi)	*Synplify and Synplify Pro tools.* Inserts a clock buffer according to the specified value.
syn_isclock (D)	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_keep (D)	Prevents the internal signal from being removed during synthesis and optimization.
syn_maxfan	Overrides the default fanout guide for an individual input port, net, or register output.
syn_multstyle	Determines how multipliers are implemented for Microsemi devices.
syn_netlist_hierarchy	Determines whether the EDIF output netlist is flat or hierarchical.
syn_noarrayports	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.
syn_noclockbuf	Turns off the automatic insertion of clock buffers.
syn_noprune (D)	Controls the automatic removal of instances that have outputs that are not driven.
syn_pad_type	Specifies an I/O buffer standard.

(D) indicates directives; all others are attributes.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_preserve (D)	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
syn_preserve_sr_priority	Forces set/reset flip-flops to honor the coded priority for the set or reset. ACT 1 and 40MX architectures only.
syn_probe	Adds probe points for testing and debugging.
syn_radhardlevel	Specifies the radiation-resistant design technique to apply to a module, architecture, or register.
syn_ramstyle	Specifies the implementation to use for an inferred RAM. You apply syn_ramstyle globally, to a module, or to a RAM instance.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
syn_replicate	Controls replication.
syn_resources	Specifies resources used in black boxes.
syn_sharing (D)	Specifies resource sharing of operators.
syn_state_machine (D)	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n> (D)	Defines timing clock to output delay through the black box. The *n* indicates a value between 1 and 10.
syn_tpd<n> (D)	Specifies timing propagation for combinational delay through the black box. The *n* indicates a value between 1 and 10.
syn_tristate (D)	Specifies that a black-box pin is a tristate pin.
syn_tsu<n> (D)	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.
syn_useenables	Generates clock enable pins for registers.
translate_off/translate_on (D)	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

(D) indicates directives; all others are attributes.

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

APPENDIX E

Designing with Xilinx

This chapter contains guidelines for synthesizing Xilinx-based designs. It discusses the following topics:

- Basic Support for Xilinx Designs, on page 2064
- Input for Xilinx Designs, on page 2071
- Xilinx Components, on page 2072
- Xilinx RAMs and ROMs, on page 2084
- Xilinx I/O Support, on page 2096
- Xilinx Optimizations, on page 2099
- Output Files and Forward-annotation for Xilinx, on page 2107
- Integration with Xilinx Tools and Flows, on page 2118
- Virtex and Spartan-II and Later Technologies, on page 2121
- Xilinx CPLDs, on page 2131
- Xilinx Attribute and Directive Summary, on page 2136

Copyright © 2013 Synopsys, Inc.

Basic Support for Xilinx Designs

The following topics briefly describe flows, constraints, and attributes supported by the synthesis tools for Xilinx designs:

- Xilinx Device Support, on page 2064
- Xilinx Stacked Silicon Interconnect (SSI) Technology, on page 2065
- Supported Synthesis Design Flows for Xilinx Designs, on page 2066
- Synthesis Constraints and Attributes for Xilinx Designs, on page 2066

For information on support in specific areas, see the other sections in this chapter.

Xilinx Device Support

The synthesis tool creates technology-specific netlists for various Xilinx FPGA and CPLD technologies. Synopsys is committed to supporting new Xilinx technologies as they become available. For a list of supported devices, check the Device panel of the Implementation Options dialog box. If a new technology is not in your current release, contact Synopsys to arrange an overlay.

You can use the Synplify Premier software with all Xilinx technologies in logic synthesis mode, but physical synthesis is not available for older Xilinx technologies. If you are using the Synplify Premier tool with an older technology and the Physical Synthesis switch does not appear on the Options tab of the Implementation Options dialog box, then physical synthesis for that technology is not supported.

See the following for device-specific information:

- Virtex and Spartan-II and Later Technologies, on page 2121
- Xilinx CPLDs, on page 2131

After synthesis, the tool generates XNF or EDIF netlists that are used as input to run the Xilinx ISE Design Suite. Also, the tool can generate a XDC netlist that is used as input to run the Xilinx Vivado place-and-route tool.

Copyright © 2013 Synopsys, Inc.

Xilinx Stacked Silicon Interconnect (SSI) Technology

Xilinx Stacked Silicon Interconnect (SSI) technology allows multiple die to be combined in a single chip providing high capacity and high-performance computing and processing for FPGAs. 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) and is connected with 49 Super Long Lines (SLL) per column between adjacent SLRs. Devices can contain 2, 3, or 4 SLRs.

Xilinx place-and-route (P&R) software treats the SLICE coordinate system for SSI as a monolithic device. However, there is a timing penalty for signals that cross SLR boundaries, so place and route tries to automatically partition logic to the SLR. If these results are not optimal, you can use the syn_assign_to_slr attribute to assign logic to a specific SLR for the device. For more information, see *syn_assign_to_slr*, on page 1231.

You can use this attribute with the logic synthesis flows targeting Virtex-7 SSI devices. The Design Planner supports SLR assignments without the need to create 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`). For details about using the Design Planner, see *Working with Xilinx SSI Devices*, on page 645.

After synthesis, SLR assignments are forward annotated to the XDC file for the Xilinx Vivado P&R tool. The P&R tool uses the create_pblock, add_cells_to_pblock, and resize_pblock constraints specified in this order. The SLR ranges are defined for each device separately.

For example, constraints are shown below for the Xilinx 2000T device.

```
create_pblock slr0
create_pblock slr1
create_pblock slr2
create_pblock slr3

add_cells_to_pblock slr0 [instance name]
add_cells_to_pblock slr2 [port name]

resize_pblock slr0 -add CLOCKREGION_X0Y0:CLOCKREGION_X1Y2
resize_pblock slr1 -add CLOCKREGION_X0Y3:CLOCKREGION_X1Y5
resize_pblock slr2 -add CLOCKREGION_X0Y6:CLOCKREGION_X1Y8
resize_pblock slr3 -add CLOCKREGION_X0Y9:CLOCKREGION_X1Y11
```


Copyright © 2013 Synopsys, Inc.

Supported Synthesis Design Flows for Xilinx Designs

The following summarizes the synthesis design flows supported for Xilinx designs. Not all flows are available for all technologies.

Logic synthesis	See *Logic Synthesis Design Flow*, on page 36
Logic synthesis with floorplan	(Synplify Premier with Design Planner). See *Design Plan-Based Logic Synthesis*, on page 42 in the *User Guide*.
Logic synthesis with fast synthesis	(Synplify Premier). See *Fast Synthesis*, on page 1061 in the *User Guide*.
Logic synthesis with enhanced optimizations	(Synplify Premier). See *Logic Synthesis with Enhanced Optimization*, on page 40 in the *User Guide*.
Design Plan-Based logic synthesis	(Synplify Premier with Design Planner). See *Design Plan-Based Logic Synthesis*, on page 42 in the *User Guide*.
Xilinx Physical Plus	(Synplify Premier). See *Xilinx Physical Plus*, on page 64 in the *User Guide*.

For other flows to integrate with Xilinx software, see *Integration with Xilinx Tools and Flows*, on page 2118 for information.

Synthesis Constraints and Attributes for Xilinx Designs

The synthesis tools let you specify timing constraints, general HDL attributes, and Xilinx-specific attributes to improve your design. You can manage the attributes and constraints in the SCOPE interface. Xilinx has vendor-specific I/O standard constraints it supports for synthesis. For a list of supported I/O standards, see *Xilinx I/O Standards*, on page 2067.

Copyright © 2013 Synopsys, Inc.

Xilinx I/O Standards

The following table shows the applicable I/O standards for the Virtex and Spartan families and the equivalent ISE I/O standards. For Virtex-6 and Spartan-6 devices, see *Virtex-6 and Spartan-6 I/O Standards*, on page 2068.

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan	Virtex-5	Virtex-II Virtex-IIP Virtex-4
AGP1X	AGP1	X	X	X
AGP2X	AGP2	X	X	X
BLVDS_25	BLVDS_25			X
CTT	CTT	X	X	X
DIFF_HSTL_15_Class_II	DIFF_HSTL15_II			X
DIFF_HSTL_18_Class_II	DIFF_HSTL18_II			X
DIFF_SSTL_18_Class_II	DIFF_SSTL18_II			X
DIFF_SSTL_2_Class_II	DIFF_SSTL2_II			X
GTL	GTL	X	X	X
GTL+	GTLP	X	X	X
HSTL_Class_I	HSTL_I	X	X	X
HSTL_Class_II	HSTL_II	X	X	X
HSTL_Class_III	HSTL_III	X	X	X
HSTL_Class_IV	HSTL_IV	X	X	X
HSTL_18_Class_I	HSTL18_I	X	X	X
HSTL_18_Class_II	HSTL18_II	X	X	X
HSTL_18_Class_III	HSTL18_III	X	X	X
HSTL_18_Class_IV	HSTL18_IV	X	X	X
HyperTransport	LDT_25			Virtex-4
LVCMOS_15	LVCMOS15	X	X	X
LVCMOS_18	LVCMOS18	X	X	X
LVCMOS_25	LVCMOS25	X	X	X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan	Virtex-5	Virtex-II Virtex-IIP Virtex-4
LVCMOS_33	LVCMOS33	X	X	X
LVDS	LVDS_25	X	X	X
LVDSEXT_25	LVDSEXT_25			X
LVPECL	LVPECL	X	X	X
LVTTL	LVTTL	X	X	X
PCI33	PCI33_3	X	X	X
PCI66	PCI66_3	X	X	X
PCI-X_133	PCIX	X	X	X
SSTL_18_Class_I	SSTL18_I	X	X	X
SSTL_18_Class_II	SSTL18_II	X	X	X
SSTL_2_Class_I	SSTL2_I	X	X	X
SSTL_2_Class_II	SSTL2_II	X	X	X
SSTL_3_Class_I	SSTL3_I	X	X	X
SSTL_3_Class_II	SSTL3_II	X	X	X
ULVDS_25	ULVDS_25			X

Virtex-6 and Spartan-6 I/O Standards

The following table shows the applicable I/O standards for the Virtex-6 and Spartan-6 families and the equivalent ISE I/O standards.

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan-6	Virtex-6
12C	12C	X	
BLVDS	BLVDS_25	X	
DIFF_HSTL_Class_I	DIFF_HSTL_I	X	X
DIFF_HSTL_Class_II	DIFF_HSTL_II	X	X
DIFF_HSTL_18_Class_II	DIFF_HSTL_II_18	X	X

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan-6	Virtex-6
DIFF_SSTL_18_Class_II	DIFF_SSTL18_II	X	X
DIFF_SSTL_2_Class_II	DIFF_SSTL2_II	X	X
DISPLAY_PORT	DISPLAY_PORT	X	
HSTL_Class_I	HSTL_I	X	X
HSTL_Class_I_12	HSTL_I_12	X	X
HSTL_Class_II	HSTL_II	X	X
HSTL_Class_III	HSTL_III	X	X
HSTL_Class_I_18	HSTL_I_18	X	X
HSTL_Class_II_18	HSTL_II_18	X	X
HSTL_Class_III_18	HSTL_III_18	X	X
HT_25	HT_25		X
LVCMOS_15	LVCMOS15	X	X
LVCMOS_18	LVCMOS18	X	X
LVCMOS_25	LVCMOS25	X	X
LVCMOS_15_JEDEC	LVCMOS15_JEDEC	X	
LVCMOS_18_JEDEC	LVCMOS18_JEDEC	X	
LVCMOS_25_JEDEC	LVCMOS25_JEDEC	X	
LVDS	LVDS_25	X	X
LVDS_33	LVDS_33	X	
LVDSEXT_25	LVDSEXT_25		X
LVPECL_25	LVPECL_25	X	
LVPECL_33	LVPECL_33	X	
MINI_LVDS_25	MINI_LVDS_25	X	
MINI_LVDS_33	MINI_LVDS_33	X	
MOBILE_DDR	MOBILE_DDR	X	
PCI33	PCI33_3	X	

Copyright © 2013 Synopsys, Inc.

Synopsys FPGA Synthesis I/O Standard	ISE I/O Standard	Spartan-6	Virtex-6
PCI66	PCI66_3	X	
PCI-X_133	PCIX	X	
RSDS_25	RSDS_25	X	
SDIO	SDIO	X	
SMBUS	SMBUS	X	
SSTL_15	SSTL15		X
SSTL_15_class_II	SSTL15_II	X	X
SSTL_18_Class_I	SSTL18_I	X	X
SSTL_18_Class_II	SSTL18_II	X	X
SSTL_2_Class_I	SSTL2_I	X	X
SSTL_2_Class_II	SSTL2_II	X	X
SSTL_3_Class_I	SSTL3_I	X	
SSTL_3_Class_II	SSTL3_II	X	
TMDS_33	TMDS_33	X	

See Also:

- *Industry I/O Standards*, on page 496 for a list of industry I/O standards.
- *Xilinx Attribute and Directive Summary*, on page 2136 for a list of Xilinx attributes and directives.

Input for Xilinx Designs

The following reference input sources for Xilinx designs:

IP Core Support

In a typical design flow, IP is instantiated as black boxes in the VHDL/Verilog source. EDIF and/or NGC files for these modules are supplied to the ISE Design Suite to complete the design. For information on Xilinx IP, see *Xilinx Cores*, on page 1046.

EDK Cores

For IP cores generated with Xilinx EDK, you can use the ise2syn utility to incorporate the cores into your design. For information about the utility and its use, see *Converting Xilinx Projects with ise2syn*, on page 1051 in the *User Guide* and *Xilinx ise2syn Utility*, on page 1017.

Initial Value Support for Xilinx

Initial values specified in the RTL for sequential elements and memories can be mapped to startup values on the FPGA. Xilinx devices support power on startup values for registers and memories. See *Initializing Xilinx RAM*, on page 398 and *Specifying Xilinx Register INIT Values*, on page 807 in the *User Guide* for step-by-step procedures.

The tool also supports initial values on asynchronous reset and no-change output registers that are mapped into RAMs.

The compiler supports both procedural assignments and calls to readmemb() and readmemh() in Verilog, from within the initial block. When loading data files with readmemb() and readmemh(), the memory is loaded from the lowest address to highest address if the start and finish address is not specified. If both start and finish addresses are specified, then loading begins at the start address and continues toward the finish address. In case start address is specified without the finish address, then loading starts at the specified start address and continues upward toward the highest address in the memory.

Copyright © 2013 Synopsys, Inc.

For example:

```
reg [7:0] mem [31:0]
$readmemh ("mem.ini", ram_bank1)
   /* Initialize RAM with contents from locations 0 thru 31*/;

$readmemh ("mem.ini", ram_bank1,0)
   /* Initialize RAM with contents from locations 0 thru 31*/;

$readmemh ("mem.ini", ram_bank1, 0, 31)
   /* Initialize RAM with contents from locations 0 thru 31*/;

$readmemh ("mem.ini", ram_bank2, 31, 0)
   /* Initialize RAM with contents from locations 31 thru 0*/;
```

Note that this behavior is based on Verilog 2005 standard.

Sequential elements can be mapped into registers/latches/SRLs, or select/block RAM. Initial values are supported only on sequential elements mapped to select/block RAM. If a read-first block RAM has an output register with an initial value, the tool preserves the INIT value and packs the register into the RAM.

Xilinx Components

The following topics describe how the synthesis tools handle various Xilinx components, and show you how to work with or manipulate them during synthesis to get the results you need:

- Macros and Black Boxes, on page 2073
- Xilinx Registers, on page 2076
- DDR Registers, on page 2076
- ROM Mapping, on page 2084
- Latch Mapping, on page 2080
- Global Set/Resets and Startup Blocks, on page 2080
- Relative Locations, on page 2081
- DSP Components in Virtex Designs, on page 2081
- Advanced LUT Combining, on page 2082

Copyright © 2013 Synopsys, Inc.

- Wide Adders in Virtex-5 and Virtex-6 Designs, on page 2082
- Control Sets in Spartan/Virtex Architectures, on page 2083

For information about I/Os, see *Xilinx I/O Support*, on page 2096

Macros and Black Boxes

You can create an instance of any externally defined macro, including a user-defined macro or Xilinx macro such as an I/O or flip-flop, by instantiating a black box in your Verilog or VHDL source code. These black boxes are empty Verilog module descriptions and VHDL component declarations.

The synthesis tool provides Xilinx macro libraries that you can use to instantiate components such as 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. The Verilog library is located at *install_dir*/lib/xilinx/unisim.v. It is automatically compiled when you choose a Xilinx target device.

To instantiate Xilinx macros such as gates, counters, flip-flops, or I/Os, use the appropriate library that defines the macros as black boxes. The macro libraries are located in the *install_dir*/lib/xilinx directory. See *Specifying Xilinx Macros*, on page 795 in the *User Guide*.

Unimacro Library Support

While most designs use generic code, the use of device-specific components is often required to achieve the desired, end-circuit implementation and results. Xilinx ISE provides a number of device-specific components that are part of the Xilinx unimacro library. To use these components in the synthesis tool, you must set the XILINX environment variable to point to the ISE installation directory.

Note: If the XILINX environment variable is not set or set incorrectly, the compiler errors out if a unimacro component is encountered in the design.

A list of the supported unimacro components can be found at:

- $XILINX/verilog/src/unimacro/ (Verilog) or

Copyright © 2013 Synopsys, Inc.

- $XILINX/vhdl/src/unimacro/ (VHDL)

Any component in the unimacro library can be directly instantiated. The following usage models are supported.

The following example uses Verilog to instantiate an ADDMACC_MACRO module from the unimacro library.

Example – Verilog Unimacro

```
module ADDMACC_MACRO_WRP ( PRODUCT, CARRYIN, CE, CLK,
   MULTIPLIER, LOAD, LOAD_DATA, PREADD1, PREADD2, RST );

// Visible parameters
parameter DEVICE = "VIRTEX6";
parameter LATENCY = 4;
parameter WIDTH_PREADD = 25;
parameter WIDTH_MULTIPLIER = 18;
parameter WIDTH_PRODUCT = 48;
output [WIDTH_PRODUCT-1:0] PRODUCT;
input CARRYIN;
input CE;
input CLK;
input [WIDTH_MULTIPLIER-1:0] MULTIPLIER;
input LOAD;
input [WIDTH_PRODUCT-1:0] LOAD_DATA;
input [WIDTH_PREADD-1:0] PREADD1;
input [WIDTH_PREADD-1:0] PREADD2;
input RST;

ADDMACC_MACRO
#(.DEVICE("SPARTAN6"))
U (.PRODUCT(PRODUCT),
   .CARRYIN(CARRYIN),
   .CE(CE),
   .CLK(CLK),
   .MULTIPLIER(MULTIPLIER),
   .LOAD(LOAD),
   .LOAD_DATA(LOAD_DATA),
   .PREADD1(PREADD1),
   .PREADD2(PREADD2),
   .RST(RST)
);

endmodule
```

Copyright © 2013 Synopsys, Inc.

The following example uses VHDL to instantiate an ADDMACC_MACRO entity from the unimacro library. When using VHDL, the library UNIMACRO must be called before instantiating any component from the unimacro library as noted in the library and use statements at beginning of the example.

Example – VHDL Unimacro

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
library UNIMACRO;
use UNIMACRO.vcomponents.all;

entity ADDMACC_MACRO_WRP is
generic (DEVICE : string := "VIRTEX6";
         LATENCY : integer := 4;
         WIDTH_PREADD : integer := 25;
         WIDTH_MULTIPLIER : integer := 18;
         WIDTH_PRODUCT : integer := 48 );

port (PRODUCT : out std_logic_vector(WIDTH_PRODUCT-1 downto 0);
      CARRYIN : in std_logic;
      CE : in std_logic;
      CLK : in std_logic;
      MULTIPLIER : in std_logic_vector
         (WIDTH_MULTIPLIER-1 downto 0);
      LOAD : in std_logic;
      LOAD_DATA : in std_logic_vector(WIDTH_PRODUCT-1 downto 0);
      PREADD1 : in std_logic_vector(WIDTH_PREADD-1 downto 0);
      PREADD2 : in std_logic_vector(WIDTH_PREADD-1 downto 0);
      RST : in std_logic );
end entity ADDMACC_MACRO_WRP;

architecture ADDMACC_MACRO_arch of ADDMACC_MACRO_WRP is
begin
INST : ADDMACC_MACRO
generic map ( DEVICE => "VIRTEX6" )
port map (PRODUCT => PRODUCT ,
          CARRYIN => CARRYIN ,
          CE => CE ,
          CLK => CLK ,
          MULTIPLIER => MULTIPLIER ,
          LOAD => LOAD ,
          LOAD_DATA => LOAD_DATA ,
          PREADD1 => PREADD1 ,
          PREADD2 => PREADD2 ,
          RST => RST );
end architecture ADDMACC_MACRO_arch;
```


Copyright © 2013 Synopsys, Inc.

Xilinx Registers

The compiler can infer additional register primitive types so that the mapper can implement a better solution when it targets Xilinx technologies:

- Registers with clock enables – The compiler identifies and extracts clock-enable logic for registers. The RTL view displays registers with enables as special primitives with an extra pin for the enable signal. The special primitives are dffe, dffre, dffse, dffrse, dffpatre, and dffpatrse. Set syn_direct_enable in the HDL source file (not in the SCOPE spreadsheet or a constraint file) to direct the compiler to infer the clock enables you want. For further syntax details see *syn_direct_enable*, on page 1248. By default, registers without clock enables will be inferred.
- Registers with synchronous sets/resets – The compiler identifies and extracts registers with synchronous sets and resets, and passes them to the mapper. The special primitives are sdffr, sdffs, sdffrs, sdffpat, sdffre, sdffse, and sdffpate. Set syn_direct_set and/or syn_direct_reset in the HDL source file (not in the SCOPE spreadsheet or a constraint file) to direct the compiler to infer the set/reset you want. For further syntax details see *syn_direct_set*, on page 1258 and *syn_direct_reset*, on page 1254. The compiler does this automatically.

DDR Registers

The Virtex-II and later technologies and the Spartan-3 and later technologies support input DDR registers. Output DDR registers are instantiated by using Xilinx FDDRRSE/FDDRCPE primitives or ODDR (Virtex-4/Virtex-5/Virtex-6). Output DDR registers (FDDRRSE/FDDRCPE/ODDR) are automatically inferred without having to instantiate them in the HDL. The following example shows the HDL coding style to infer the output DDR registers.

Example

```
module fddrrse(d1, d0, clk0, clk1, rst, set, ce, q);
input [0:0]d1;
input [0:0]d0;
input clk0, clk1;
input rst;
```

```
input set;
input ce;
output [0:0]q;
reg [0:0]q0;
reg [0:0]q1;

always @(posedge clk0)
begin
   if(rst)
      q0 = 1'b0;
   else if(set)
      q0 = 1'b1;
   else if(ce)
      q0 = d0;
end

always @(posedge clk1)
begin
   if(rst)
      q1 = 1'b0;
   else if(set)
      q1 = 1'b1;
   else if(ce)
      q1 = d1;
end
assign q = clk0 ? q0 : q1;
endmodule
```

The following schematic represents this Verilog code. The mapper checks that clk0 and clk1 are output from a DCM, and that the two clocks have 180 degrees of phase shift. The mapper then replaces the clocks with a DDR primitive. To infer the output DDR primitive, additional checks are made to ensure that clk0 and clk1 are 180 degrees out of phase.

Copyright © 2013 Synopsys, Inc.

Alternate coding styles:

1. Use positive and negative edges to clock the two registers which are then inferred as DDR flip-flops.

2. Use the negative edge of clk0 to trigger q1.

Dynamic Shift Register Lookup (SRL) Table

The synthesis tools infer dynamic and static sequential shift components in corresponding Virtex and Spartan architectures. If you do not want to automatically infer the shift registers, set the syn_srlstyle attribute to registers. See *Inferring Shift Registers*, on page 405 in the *User Guide* for details.

VHDL SRL Example

This is a VHDL example of a shift register with no resets. It has four 8-bit wide registers and a 2-bit wide read address. Registers shift when the write enable is 1.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity srltest is
   port (inData: std_logic_vector(7 downto 0);
         clk, en : in std_logic;
         outStage : in integer range 3 downto 0;
         outData: out std_logic_vector(7 downto 0) );
end srltest;

architecture rtl of srltest is
type dataAryType is array(3 downto 0) of
   std_logic_vector(7 downto 0);
signal regBank : dataAryType;
begin
outData <= regBank(outStage);
   process(clk)
   begin
      if (clk'event and clk = '1') then
         if (en='1') then
            regBank <= (regBank(2 downto 0) & inData);
         end if;
      end if;
   end process;
end rtl;
```

Verilog SRL Example

This is a Verilog example of a shift register with no resets. It has sixteen 4-bit wide registers and a 4-bit wide read address. Registers shift when the write enable is 1.

Copyright © 2013 Synopsys, Inc.

```
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
```

Latch Mapping

For architectures that do not support level-sensitive latches, the latches can be mapped to the latch components LD_1 (level-sensitive latch), LDC_1 (level-sensitive latch with clear), LDP_1 (level-sensitive latch with preset), and LDCP_1 (level-sensitive latch with preset and clear) so that you can supply implementations. Sample XNF files are included containing implementations you can use when you run the Xilinx ISE Design Suite. Alternatively, you can change the XNF files for these latches to create your own implementations before performing placement and routing.

The sample latch XNF files (with extension `xnf`) are located in the *install_dir*/lib/xilinx directory.

Global Set/Resets and Startup Blocks

The synthesis tools automatically infer the dedicated routing resource for global sets/resets (GSR). However, you can also access this resource using Xilinx startup blocks. See *Specifying Global Sets/Resets and Startup Blocks*, on page 797 in the *User Guide* for details.

The Force GSR Usage option (Implementation Options->Device) lets you control whether the synthesis tool uses the routing resource:

auto	Lets the synthesis tool determine the appropriate setting for to the design.
yes	Ensures that the tool uses the global set/reset (GSR) routing resources.
no	Does not use the GSR resources.

Relative Locations

Use these Xilinx-specific attributes to specify the relative position of components in your designs.

- xc_map
- xc_rloc
- xc_uset

See *Specifying RLOCs,* on page 817 in the *User Guide* for information on how to use these attributes to specify relative locations. You can also define RLOCs and RLOC_ORIGINs with the synthesis attribute, as described in *Specifying RLOCs and RLOC_ORIGINs with the synthesis Attribute,* on page 819 in the *User Guide*.

DSP Components in Virtex Designs

For the Virtex-4, Virtex-5, Virtex-6, Spartan-3A DSP, and Spartan-6 families, the Synopsys FPGA tool automatically infers DSP structures. Structures can be general or customized depending on the coding style.

DSPs are used for the most important structures in the design, so they are not automatically inferred for adders and counters. To control the inference of DSPs on operators, registers, and module/architectures, use the syn_dspstyle attribute. The syn_dspstyle attribute value is either logic, which ignores the DSP, or dsp48, which infers the DSP.

Copyright © 2013 Synopsys, Inc.

DSP Examples

The following examples show the inference of an adder to a DSP.

VHDL

```
attribute syn_dspstyle : string;
attribute syn_dspstyle of adder_sig : signal is "dsp48";
```

Verilog

```
wire [9:0] adder_sig /* synthesis syn_dspstyle = "dsp48" */;
```

The following examples show the inference of a multiplier to logic.

VHDL

```
attribute syn_dspstyle : string;
attribute syn_dspstyle of mult_sig : signal is "logic";
```

Verilog

```
wire [9:0] mult_sig /* synthesis syn_dspstyle = "logic" */;
```

Advanced LUT Combining

The Enable Advanced LUT Combining (Implementation Options->Device) option is used to prepare the netlist for advanced LUT combining, so that the Xilinx ISE P&R tool can pack into the LUT6_2 depending on available resources. This option is available for Virtex-5 and Spartan-6 or newer devices and supported only in Xilinx ISE 10.1 (sp1) or later versions. You can also enable this mode using the equivalent command: set_option -enable_prepacking 1.

Wide Adders in Virtex-5 and Virtex-6 Designs

Virtex-5 and Virtex-6 support a wide adder/subtractor structure by cascading DSPs using the CARRYCASCOUT pin. The structure supports three, two, one, or zero pipeline registers with different synchronous control signals. An adder/subtractor with three pipelined registers gives maximum performance. Both the adders and subtractors support two or three signed/unsigned inputs (with carry/borrow).

Copyright © 2013 Synopsys, Inc.

The synthesis tools infer a wide adder or subtractor and map it to a DSP if you set the syn_dspstyle to dsp48 (*syn_dspstyle*, on page 1264). See *Inferring Wide Adders*, on page 798 in the *User Guide* for the complete procedure.

Control Sets in Spartan/Virtex Architectures

The Spartan-6 and Virtex-5 and later architectures optimize area by having all registers that appear in the same slice share the clock, clock enable, and synchronous set/reset signals. Each unique combination of these signals is called a control set.

The Xilinx ISE Design Suite cannot place two flip-flops with different control sets in the same slice. This restriction can result in placement issues when there are a large number of unique control sets and there are more than two registers in a slice as found in the larger Xilinx technologies (e.g., the Spartan-6 and Virtex-6 architectures include eight registers in each slice).

The synthesis software controls slice placement by moving control signals to the data inputs when the control set is unique. This redefinition of the control set reduces the number of unique control sets which improves the placement of registers in the design. An attribute (syn_reduce_controlset_size) is available to set the minimum size of the control set on which control-set optimizations can occur. For details on this attribute, see *syn_reduce_controlset_size (Xilinx)*, on page 1480.

Copyright © 2013 Synopsys, Inc.

Xilinx RAMs and ROMs

- ROM Mapping, on page 2084
- Xilinx RAMs, on page 2084

ROM Mapping

You can map ROM to block RAM with the syn_romstyle attribute. See *Mapping Xilinx ROM to Block RAM*, on page 392 of the *User Guide* for details.

Xilinx RAMs

Depending on the technology, you can infer distributed or block RAMs. You can also disable automatic RAM inference and/or specify the RAM implementation with the syn_ramstyle attribute.

Asymmetric Block RAM Support

Supports Xilinx Virtex-6 and later devices

The synthesis tools provide support for RAM with asymmetric ports. Asymmetric ports have different data width and address depth; these ports are not interchangeable. The synthesis tool can infer a single block RAM primitive for the simple dual-port RAM.

Support is limited to specific coding styles of simple dual-port RAM as shown below.

Verilog Example: Coding Style 1

The following Verilog example implements an asymmetric port RAM where:

- Port A is 256x8-bit read-only
- Port B is 64x32-bit write-only
- Write first mode has no control signals on address register with different clocks

Copyright © 2013 Synopsys, Inc.

```
module v_asymmetric_ram (clkA, clkB, weB, addrA, addrB, doA, diB);

parameter WIDTHA      = 8;
parameter SIZEA       = 256;
parameter ADDRWIDTHA  = 8;
parameter WIDTHB      = 32;
parameter SIZEB       = 64;
parameter ADDRWIDTHB  = 6;

input clkA;
input clkB;
input weB;
input     [ADDRWIDTHA-1:0]  addrA;
input     [ADDRWIDTHB-1:0]  addrB;
output    [WIDTHA-1:0]      doA;
input     [WIDTHB-1:0]      diB;
reg       [ADDRWIDTHA-1:0]  addrA_reg;

`define max(a,b) {(a) > (b) ? (a) : (b)}
`define min(a,b) {(a) < (b) ? (a) : (b)}

function integer log2;
input integer value;
reg [31:0] shifted;
integer res;
   begin
      if (value < 2)
         log2 = value;
      else
      begin
      shifted = value-1;
         for (res=0; shifted>0; res=res+1)
         shifted = shifted>>1;
         log2 = res;
      end
   end
endfunction

localparam maxSIZE   = `max(SIZEA, SIZEB);
localparam maxWIDTH  = `max(WIDTHA, WIDTHB);
localparam minWIDTH  = `min(WIDTHA, WIDTHB);
localparam RATIO     = maxWIDTH / minWIDTH;
localparam log2RATIO = log2(RATIO);
reg     [minWIDTH-1:0]  RAM [0:maxSIZE-1];
reg     [WIDTHB-1:0]  readB;
genvar i;
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge clkA)
begin
   addrA_reg <= addrA;
end
assign doA = RAM[addrA_reg];

generate for (i = 0; i < RATIO; i = i+1)
   begin: ramread
      localparam [log2RATIO-1:0] lsbaddr = i;
      always @(posedge clkB)
      begin
      if (weB)
         RAM[{addrB, lsbaddr}] <= diB[(i+1)
         *minWIDTH-1:i*minWIDTH];
      end
   end
endgenerate

endmodule
```

VHDL Example: Coding Style 1

The following VHDL example implements an asymmetric port RAM where:

- Port A is 256x8-bit read-only
- Port B is 64x32-bit write-only
- Write_first mode has no control signals on address register with different clocks

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity asymmetric_ram is
generic (
   WIDTHA      : integer := 8;
   SIZEA       : integer := 256;
   ADDRWIDTHA  : integer := 8;
   WIDTHB      : integer := 32;
   SIZEB       : integer := 64;
   ADDRWIDTHB  : integer := 6 );

   port (clkA   : in  std_logic;
         clkB   : in  std_logic;
```

```
         weB    : in  std_logic;
         addrA  : in  std_logic_vector(ADDRWIDTHA-1 downto 0);
         addrB  : in  std_logic_vector(ADDRWIDTHB-1 downto 0);
         diB    : in  std_logic_vector(WIDTHB-1 downto 0);
         doA    : out std_logic_vector(WIDTHA-1 downto 0) );
end asymmetric_ram;

architecture behavioral of asymmetric_ram is
function max(L, R: INTEGER) return INTEGER is
   begin
      if L > R then
         return L;
      else
         return R;
      end if;
   end;

function min(L, R: INTEGER) return INTEGER is
   begin
      if L < R then
         return L;
      else
         return R;
      end if;
   end;

function log2 (val: INTEGER) return natural is
      variable res : natural;
   begin
      for i in 0 to 31 loop
         if (val <= (2**i)) then
            res := i;
            exit;
         end if;
      end loop;
   return res;
end function Log2;

constant minWIDTH : integer := min(WIDTHA,WIDTHB);
constant maxWIDTH : integer := max(WIDTHA,WIDTHB);
constant maxSIZE  : integer := max(SIZEA,SIZEB);
constant RATIO : integer := maxWIDTH / minWIDTH;
type ramType is array (0 to maxSIZE-1) of
std_logic_vector(minWIDTH-1 downto 0);
shared variable ram : ramType := (others => (others => '0'));
signal addrA_reg  : std_logic_vector(ADDRWIDTHA-1 downto 0);
```


Copyright © 2013 Synopsys, Inc.

```
begin
   process (clkA)
   begin
      if rising_edge(clkA) then
         addrA_reg <= addrA;
      end if;
   end process;
doA <= ram(conv_integer(addrA_reg));

   process (clkB)
   begin
      if rising_edge(clkB) then
         if weB = '1' then
            for i in 0 to RATIO-1 loop
               ram(conv_integer(
               addrB & conv_std_logic_vector(i,log2(RATIO))))
             := diB((i+1)*minWIDTH-1 downto i*minWIDTH);
            end loop;
         end if;
      end if;
   end process;

end behavioral;
```

The following example shows the RTL View for Coding Style 1:

The following example shows the Technology View for Coding Style 1:

Verilog Example: Coding Style 2

The following Verilog example implements an asymmetric port RAM where:

- Port A is 256x32-bit write-only
- Port B is 512x16-bit read-only
- Read_first mode has enable on the read port with different clocks and enables

```
module asymmetric_ram (clkA, clkB, enA, enB, weA, addrA,
   addrB, diA, doB );
parameter WIDTHA = 32;
parameter SIZEA = 256;
parameter ADDRWIDTHA = 8;
parameter WIDTHB= 16;
parameter SIZEB = 512;
parameter ADDRWIDTHB = 9;

input clkA, clkB, enA, enB, weA;
```


Copyright © 2013 Synopsys, Inc.

```
input [ADDRWIDTHA-1:0] addrA;
input [ADDRWIDTHB-1:0] addrB;
input [WIDTHA-1:0]    diA ;
output reg [WIDTHB-1:0] doB;
reg [WIDTHA-1:0] mux;
reg [WIDTHA-1:0] RAM [SIZEA-1:0];

always @(posedge clkA)
begin
   if(enA & weA)
      RAM[addrA] <= diA;
end

always @(posedge clkB)
   begin
   mux = RAM[addrB[ADDRWIDTHB-1:1]];
   if(enB)
      if (addrB[0])
      begin
         doB <= mux[WIDTHA-1:WIDTHB];
      end
      else
      begin
         doB <= mux[WIDTHB-1:0];
      end
   end

endmodule
```

VHDL Example: Coding Style 2

The following VHDL example implements an asymmetric port RAM where:

- Port A is 256x32-bit write-only
- Port B is 512x16-bit read-only
- Read_first mode has enable on the read port with different clocks and enables

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity asymmetric_ram is
   generic (
      WIDTHA      : integer := 32;
```

```
      SIZEA       : integer := 256;
      ADDRWIDTHA  : integer := 8;
      WIDTHB      : integer := 16;
      SIZEB       : integer := 512;
      ADDRWIDTHB  : integer := 9
   );

   port (
      clkA   : in  std_logic;
      clkB   : in  std_logic;
      rst    : in  std_logic;
      weA    : in  std_logic;
      enA    : in  std_logic;
      enB    : in  std_logic;
      addrA  : in  std_logic_vector(ADDRWIDTHA-1 downto 0);
      addrB  : in  std_logic_vector(ADDRWIDTHB-1 downto 0);
      diA    : in std_logic_vector(WIDTHA-1 downto 0);
      doB    : out std_logic_vector(WIDTHB-1 downto 0)
   );
end asymmetric_ram;

architecture behavioral of asymmetric_ram is
type ramType is array (0 to SIZEA-1)
   of std_logic_vector (WIDTHA-1 downto 0);
SHARED VARIABLE ram : ramType;

begin
process (clkA)
   begin
   if rising_edge(clkA) then
      if enA = '1' then
         if weA = '1' then
            ram(conv_integer(addrA)) := diA;
         end if;
      end if;
   end if;
end process;

process (clkB)
variable mux  : std_logic_vector(WIDTHA-1 downto 0);
   begin
   if rising_edge(clkB) then
      if enB = '1' then
         if addrB(0) = '0' then
            mux := ram(conv_integer
               (addrB(ADDRWIDTHB-1 downto 1)));
```


Copyright © 2013 Synopsys, Inc.

```
            doB <= mux (WIDTHB-1 downto 0);
         else
            mux := ram(conv_integer(
               addrB(ADDRWIDTHB-1 downto 1)));
            doB <= mux(WIDTHA-1 downto WIDTHB);
         end if;
      end if;
   end if;
end process;

end behavioral;
```

The following example shows the RTL View for Coding Style 2:

The following example shows the Technology View for Coding Style 2:

Limitations

Limitations include the following:

- Support is limited to simple dual-port RAM type coding style.
- Memory accessibility for Read and Write ports must match exactly.
- Ratio between Read and Write data widths are a power of two.
- Ratio between port depths are a power of two.
- Supports read widths that are less than write widths.

  If the read width is greater than the write width, then the compiler infers one n ram for each read operation forcing multiple RAMB primitives to be inferred.
- Initial values for RAM are not supported.
- Control signals and initial values on Read address register are not supported.

Copyright © 2013 Synopsys, Inc.

Distributed RAM Implementations

The synthesis tool can infer synchronous RAMs from your HDL source code, and generate single- or dual-port RAMs using the distributed RAM resources in the CLBs. See *Verilog Guidelines for RAM Inference*, on page 709 (Verilog), and *VHDL Guidelines for RAM Inference*, on page 940 (VHDL) for syntax details, and *Implementing Xilinx RAMs Automatically*, on page 379 and *Specifying RAM Implementation Styles*, on page 375 of the *User Guide*.

Mapping of Byte Enables to Block RAM

The synthesis tool can map byte enable pins to block RAM if they are coded in the prescribed style. For details, see *Verilog Guidelines for Byte-Enable Block RAMs*, on page 741 (Verilog) or *VHDL Guidelines for Byte-Enable Block RAMs*, on page 954 (VHDL).

Mapping of Byte-Write Enable to Block RAM

The synthesis tool can map byte-write enable for RAMs with mutually exclusive writes to block RAMs. For details, see *Verilog Guidelines for Byte-Wide Write Enable RAMs*, on page 742 (Verilog) or *VHDL Guidelines for Byte-Wide Write Enable RAMs*, on page 955 (VHDL).

Single-port and Dual-port Block RAM Implementations

Block RAMs support multiple clocks and use enables and resets to control RAM read and write operations. By default, the tool infers block SelectRAM, but you can force block RAM inference by specifying the syn_ramstyle attribute with a value of block_ram or no_rw_check.

- Single-port block RAMs

  In single-port block RAMs, the enable signal has the highest priority. See *Verilog Coding for Xilinx Single-port and Dual-port Block RAMs*, on page 738 for more information and examples.

- Dual-port Block RAMs

  Except for additional address collision recovery logic, a dual-port RAM is essentially two independent single-port RAMs that share a common memory. Dual-port RAMs have only one write port. See *Verilog Coding for Xilinx Single-port and Dual-port Block RAMs*, on page 738 for more information and examples.

Copyright © 2013 Synopsys, Inc.

Multi-Port RAM Implementations

The compiler extracts a multi-port RAM called nram, which has one read port and multiple write ports. Each write port has its own write clock, write enable, data in, and write address. For syntax and coding information, see *Multi-Port RAM Extraction*, on page 711 (Verilog), and *Multi-port RAM Extraction*, on page 946 (VHDL). For usage, see *Basic Guidelines for Coding RAMs*, on page 370, *Specifying RAM Implementation Styles*, on page 375 and *Implementing Xilinx RAMs Automatically*, on page 379 in the *User Guide*.

The nram is mapped to true dual-port block RAMs in the following cases:

- An inferred RAM with two writes and one read. The read shares an address with only one of the write ports.
- Two inferred RAMs with exactly the same write addresses, clocks, and enables, but with different read address can be paired together and mapped.

Parity Bus

When using Virtex-II block RAM, the parity bus is used to infer RAMs of specific data bus widths. For example, instead of implementing a memory with a 9-bit data bus as two block RAMs (one 8-bit and one 1-bit), the software uses a single block RAM with an 8-bit DI/DO data bus, and implements the ninth bit using the DIP/DOP parity bus.

To utilize the parity bus, treat the RAM as if it is of width 9/18/36 when calculating and selecting the block RAM aspect ratio. The software creates a separate memory array for the parity bus. When it assigns bits, the software assigns the parity bus to the MSBs of the data. For example, if d[8:0] is implemented in RAMB16_S9, d[8] is assigned to DIP[0], and d[7:0] are assigned to DI[7:0].

Disabling RAM Inference

If your RAM resources are limited, designate additional instances of inferred RAMs as flip-flops and logic using the syn_ramstyle attribute with a value of registers. This attribute is specified for the signal (VHDL) or port (Verilog) that defines the RAM, or the label of the RAM component instance. For syntax information, see *syn_ramstyle*, on page 1467.

Copyright © 2013 Synopsys, Inc.

Xilinx I/O Support

The following topics describe how the synthesis tools handle Xilinx I/O components, and show you how to work with them during synthesis to get the results you need:

- Xilinx I/O Macros, on page 2096
- Xilinx I/O Pads, on page 2096
- Xilinx I/O Insertion, on page 2096
- IOB Packing, on page 2097
- Xilinx Differential I/O Buffers, on page 2097
- Xilinx BUFGMUX, on page 2097
- Xilinx I/Os and Pin Locations, on page 2097
- Xilinx Regional Clock Buffers, on page 2098

Xilinx I/O Macros

You can create an instance of a Xilinx I/O macro by instantiating a black box in your source code. These black boxes are empty Verilog module or VHDL component declarations. To instantiate the Xilinx macros, use the appropriate library that defines the macros as black boxes. The macro libraries are located in the *install_dir*/lib/xilinx directory. See *Specifying Xilinx Macros*, on page 795 in the *User Guide*.

Xilinx I/O Pads

For all Virtex technologies and all Spartan-II and later technologies, you can use the Xilinx macro libraries to instantiate I/O pads (Xilinx Select I/O™ resources) using different I/O standards. See *Specifying Xilinx Macros*, on page 795 in the *User Guide*.

Xilinx I/O Insertion

I/O pads are automatically inserted into the design. If you manually instantiate I/Os, I/Os are inserted only for the pins that need them.

Copyright © 2013 Synopsys, Inc.

If you do not want to insert *any* I/Os in the design, select the Disable I/O Insertion check box in the Implementation Options dialog box. This is useful for bottom-up compiles, because you can check the area your logic blocks use before you synthesize the whole design. If you disable I/O insertion, you will not get any I/Os in the design unless you manually instantiate them. For more information about dependencies when using this option, see *syn_insert_pad*, on page 1376.

IOB Packing

When a register drives an output, or an input drives a register, you can place the register in a CLB or in an IOB. Depending on the design requirements, you may want to pack the registers in the IOB instead of the CLB. You can use the syn_useioff attribute to control register packing. See *Packing Registers for Xilinx I/Os*, on page 804 in the *User Guide*.

Xilinx Differential I/O Buffers

By default, the synthesis tool does not automatically infer differential I/O buffers. You can direct the tool to infer IBUFDS, IBUFGDS, OBUFDS, OBUFTDS, and IOBUFDS by setting the syn_diff_io attribute to 1 or true. See *Working with Xilinx Buffers*, on page 815 in the *User Guide*.

Xilinx BUFGMUX

By default, the Xilinx mapper does not infer BUFGMUX/BUFGMUX_1 from the HDL. You can control inference by setting syn_insert_buffer on the mux instance. See *Working with Xilinx Buffers*, on page 815 in the *User Guide*.

Xilinx I/Os and Pin Locations

By default the synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs), and the Xilinx ISE Design Suite chooses the I/O locations. However, you can manually instantiate I/Os and specify pin locations with the syn_loc attribute. See *Inserting Xilinx I/Os and Specifying Pin Locations*, on page 809 in the *User Guide*.

Copyright © 2013 Synopsys, Inc.

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 based on the Virtex architectures and 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 syn_insert_buffer attribute. For more information, see *Working with Xilinx Regional Clock Buffers*, on page 816.

Xilinx Optimizations

The synthesis tools offer various optimiztions for Xilinx designs. The following describe the optimizations in more detail:

- Fanout Limits in Xilinx Designs, on page 2099
- Pipelining in Xilinx Designs, on page 2101
- Retiming in Xilinx Designs, on page 2101
- Xilinx Gated and Generated Clocks, on page 2102
- Sequential Optimizations in Xilinx Designs, on page 2102
- Verification Mode in Xilinx Designs, on page 2103
- Compile Point Remapping in Xilinx Designs, on page 2104
- Clock Skew Estimation for Xilinx Designs, on page 2105
- Power Optimization for Xilinx Designs, on page 2105

Fanout Limits in Xilinx Designs

Large fanouts can cause large delays and routability problems. The synthesis tool maintains reasonable fanouts automatically, and also allows you to specify maximum fanouts. There are two ways to specify fanout limits: the Fanout Guide option (*Fanout Guide Option*, on page 2100), and the syn_maxfan attribute (*The syn_maxfan Attribute for Xilinx Designs*, on page 2100).

The software first reduces fanout by replicating the driver of the high fanout net and splitting the net into segments. This replication can affect the number of register bits and LUTs in your design. If replication is not possible, the signals are buffered. Buffering increases intrinsic delay and consumes more resources, and is therefore not used until a slightly higher fanout limit has been reached than is specified. See *Controlling Buffering and Replication*, on page 446 in the *User Guide* for more details. For extremely large clock fanout nets, the software inserts global buffer (BUFG) cells in most cases. For large non-clock nets, it may be necessary to insert global buffers manually. Global buffers reduce the delay on large fanout nets and can free up routing resources for other signals. View the net buffering report for details (see *Xilinx Net Buffering Report*, on page 2108).

Copyright © 2013 Synopsys, Inc.

For tips on setting and using fanout limits, see *Setting Fanout Limits*, on page 444 and *Controlling Buffering and Replication*, on page 446 in the *User Guide*.

Fanout Guide Option

Select Project->Implementation Options->Device. Specify an integer value for Fanout Guide. The minimum value you can enter is 8. The default limit depends on the target device selected.

During technology mapping, the synthesis tool tries to keep the fanout to less than the specified fanout guide. However, the fanout guide is only a *guideline*, not a hard limit. This means that the synthesis tool takes it into account, but does not always respect it absolutely. If the limit imposes constraints that interfere with optimization, the software does not respect the limit.

The syn_maxfan Attribute for Xilinx Designs

The syn_maxfan attribute controls the maximum fanout of an instance, net, or port. See *syn_macro*, on page 1392 for details of the syntax. The limit specified by this attribute is treated as a hard or soft limit, depending on where it has been specified. A set of rules that treat the fanout limit as soft or as hard is described below:

- The maximum fanout is specified globally either by setting it as an implementation option (Fanout Guide) or by specifying the syn_maxfan attribute on a top-level module or view. A globally specified maximum fanout limit is considered as a soft limit (or guide) and may not be honored if the limit degrades performance.
- A syn_maxfan attribute can be applied on a module or view. In this case, the limit specified is treated as a soft limit for the scope of the module. This limit overrides the global fanout guide limit for the scope of the module.
- When a syn_maxfan attribute is specified on an instance that is not of primitive type as inferred by the Synopsys FPGA compiler, the limit is considered a soft limit, which is propagated down the hierarchy.
- When a syn_maxfan attribute is specified on a port, net, or register (or any primitive instance), the limit is considered a hard limit. Note that the syn_maxfan attribute does not prevent the instance from being optimized away. Implementing a hard fanout limit gives you the ability to specify implementations that will violate place and route design rules. The tool

Copyright © 2013 Synopsys, Inc.

checks for design rule violations and if detected, the fanout limit is not honored and a warning is generated.

Note that it may not be possible to detect all possible design rule violations that result from allowing a hard fanout limit. In these cases, you must take responsibility and adjust the fanout value (or remove the syn_maxfan attribute) to produce a valid netlist. Note also that the syn_maxfan attribute is treated as a guide on clock nets and asynchronous control nets (for example, asynchronous reset).

Pipelining in Xilinx Designs

Pipelining multipliers and ROMs allows your design to operate at a faster frequency. The synthesis tool moves registers that follow a multiplier or ROM into the multiplier or ROM, provided that you have registers in your RTL code. Pipelining is only available for the Synplify Pro and Synplify Premier products.

You can pipeline ROMs in all Xilinx FPGA technologies if the ROM is bigger than 512 words. You can also pipeline multipliers in all Virtex technologies and in the Spartan-II and later technologies.

- To invoke pipelining and apply it globally, enable the Pipelining option in the Project view.
- To apply this attribute locally, add the syn_pipeline attribute to the registers driven by the multipliers or ROMs. See *syn_pipeline*, on page 1442 for the syntax.

See *Pipelining*, on page 426 in the *User Guide* for a step-by-step procedure on using pipelining.

Retiming in Xilinx Designs

Retiming is a powerful technique for improving the timing performance of sequential circuits. The retiming process moves storage devices (flip-flops) across computational elements with no memory (gates/LUTs) to improve the performance of the circuit. You can use retiming with all Virtex technologies and with Spartan-II and later technologies. Pipelining is automatically enabled when retiming is enabled.

Retiming is only available for the Synplify Pro and Synplify Premier products.

Copyright © 2013 Synopsys, Inc.

Set global retiming either from the Project view check box or from the Device tab under the Impl Options button. Use the syn_allow_retiming attribute to enable or disable retiming for individual flip-flops. See *syn_allow_retiming*, on page 1216 for syntax details, or *Retiming*, on page 430 in the *User Guide* for a procedure.

Xilinx Gated and Generated Clocks

The Clock Conversion option on the GCC (Synplify Pro) or GCC & Prototyping Tools tab (Synplify Premier), when enabled, attempts to remove the gated clocks from synchronous logic and to convert generated clocks to the original clock with an enable. Gated- and generated-clock conversion is is not available for the CPLD technologies

Gated Clocks

Gated clocks are converted by removing the enable logic from the clock path. Extracting the enable logic allows clock constraints to be applied globally and eliminates broken paths. Converting gated clocks also makes use of the dedicated clock networks to simplify ASIC prototyping. For information on using the Clock Conversion option with gated clocks, see *Working with Gated Clocks*, on page 466 in the *User Guide*.

Generated Clocks

Generated clocks are converted to a circuit with an enable to improve performance by reducing clock skew. For more information on using the Clock Conversion option with generated clocks, see *Optimizing Generated Clocks*, on page 499 in the *User Guide*.

Sequential Optimizations in Xilinx Designs

The Disable Sequential Optimization option determines when sequential optimizations are performed. Unchecking the checkbox or setting the option to 0 (the defaults) enables sequential optimizations to be performed. Note that unused registers are always removed from the design regardless of this option setting. Disabling sequential optimizations (checking the Disable Sequential Optimizations

Copyright © 2013 Synopsys, Inc.

checkbox) can increase delay times and area requirements, and effectively disables the FSM Compiler. The Tcl equivalent for this device option is set_option -no_sequential_opt 1 | 0.

Verification Mode in Xilinx Designs

Verification mode determines whether optimizations are disabled so that the output is compatible with verification tools. This option is only supported in the Synplify Pro and Synplify Premier tools. You control verification mode with the Verification Mode option in the Device tab of the Implementation Options dialog box. Or you can use the equivalent Tcl command: set_option -verification_mode 1|0.

Note that although they are similar, Verification Mode differs from the Disable Sequential Optimizations option. The Disable Sequential Optimizations option only disables sequential optimizations, but Verification Mode disables other FPGA optimizations as well.

The following describes the effects of setting the Verification Mode option:

Enabled	Ensures that synthesis output files are compatible with the verification tool by disabling various sequential optimizations that cannot be easily verified. It disables boundary optimizations and sequential optimizations, like the inference of resettable SRLs. When you enable Verification Mode, you must also disable the Retiming and Pipelining options and remove any assigned syn_pipeline and syn_allow_retiming attributes. Enabling this mode lets you run Formality, which generates a verification `svf` file. This file allows you to formally verify the equivalence of your input RTL description with the gate-level netlist. Enable this option only when this verification is needed. For more information about using Formality, see Chapter 24, *Verifying Results with Formality* of the *User Guide*.
Disabled	Produces synthesis output that performs the standard default optimizations when verification mode is turned off.

Copyright © 2013 Synopsys, Inc.

Compile Point Remapping in Xilinx Designs

Compile points are smaller synthesis units that allow you to break up a larger design into smaller units and perform incremental synthesis. See *Synthesizing Compile Points*, on page 1109 and *Compile Point Basics*, on page 1088 for information about the flow and compile points. Compile points are supported for all Virtex technologies and Spartan-II or later technologies. Compile points are only supported by the Synplify Pro and Synplify Premier products.

The Update Compile Point Timing Data option determines if changes to a compile point force the remapping of its parent, taking into account the new timing model of the child. The term *child* refers to a compile point that is contained inside another; the term *parent* refers to the compile point that contains the child. These terms are thus not used here in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points.

The following table describes the settings for the Update Compile Point Timing Data option:

Disabled	(Default). Only compile points that have changed are remapped, and their remapping does *not* take into account changes in the timing models of any of their children. The old (pre-change) timing model of a child is used to map and optimize its parents. If the option is disabled and the *interface* of a locked compile point is changed, the immediate parent of the compile point must be changed accordingly. In this case, both are remapped, and the *updated* timing model of the child is used when remapping this parent.
Enabled	Compile point changes are taken into account by updating the timing model of the compile point and resynthesizing all of its parents (at all levels), using the updated model. This includes any compile point changes that took place prior to enabling this option, and which have not yet been taken into account because the option was disabled. The timing model of a compile point is updated when either of the following is true: • The compile point is remapped, and the Update Compile Point Timing Data option is enabled. • The interface of the compile point is changed.

Copyright © 2013 Synopsys, Inc.

Clock Skew Estimation for Xilinx Designs

In Xilinx FPGAs, most flip-flops are clocked by a signal routed through one of the large clock trees. The FPGA synthesis tools mimic the Xilinx clock skew calculations by representing these large clock trees as buffers (BUFGs or BUFRs) driving a single net. However, for on-chip variation (OCV) timing, the Xilinx tool might treat these nets as having three levels and remove the common delay (CRPR) up to the common node, the point on the net where the source and destination clocks diverge.

The synthesis tools also calculate skews from the common node to get more realistic and less pessimistic results. The tools match the Xilinx clock skew estimates for the clock paths listed below.

- Clock pad -> BUFG -> Flip-flop
- Clock Pad -> BUFR -> Flip-flop
- Clock Pad -> DCM with internal BUFG feedback -> Flip-flop
- Clock Pad -> PLL with internal BUFG feedback -> Flip-flop

The synthesis tools have some limitations to clock skew calculation. The tools do not support the following:

- Paths that involve combinational logic
- Paths that involve sequential logic, like divide-by-2 using a flip-flop
- Cascading BUFGs
- Cascading BUFRs
- Cascading DCMs
- Cascading PLLs
- Paths with DSP are approximated, using the flip-flop closest to the DSP
- Path with RAM are approximated, using the flip-flop closest to the RAM

Power Optimization for Xilinx Designs

Synplify Premier only. The Power Optimization switch allows you to reduce power for designs that contain Xilinx block RAM and DSP48 components.

To use this option, select the Implementation Options->Options panel and enable Power Optimization.

Copyright © 2013 Synopsys, Inc.

When Power Optimization is enabled, you can improve power consumption for Xilinx designs with block RAMs or DSP48 components.

- Block RAMs
  When memory components of a design are mapped to Xilinx block RAMs, the synthesis software utilizes smaller block RAM components to optimize for power.

  Power optimization is performed globally on the design, however, you can apply syn_ramstyle="block_ram" on specific memory to selectively avoid power optimization for these components.

- DSP48 Components
  When arithmetic components of a design, such as adders or multipliers are mapped to Xilinx DPS48, the synthesis software configures the DSP48 to optimize for power. Optimizations occur for input port bits of the DSP48 and the configuration of its internal registers.

Copyright © 2013 Synopsys, Inc.

Output Files and Forward-annotation for Xilinx

After synthesis, the software generates a log file and output files for Xilinx. The following describe some of the reports with Xilinx-specific information, or files that forward-annotate information to the Xilinx ISE Design Suite.

- Xilinx Resource Usage Reports, on page 2107
- Xilinx Net Buffering Report, on page 2108
- Virtex-5 and Virtex-6 Detailed Clock Path Report, on page 2109
- Custom Timing Reports for Xilinx, on page 2109
- Forward-annotation of Xilinx Constraints, on page 2110
- Forward-annotation of Port Names in EDIF, on page 2117

Xilinx Resource Usage Reports

The log file for Xilinx designs includes various reports: a resource usage report, a timing report, and a net buffering report. To view these synthesis reports, click View Log in the Project view. The Xilinx resource usage reports include this information:

- Cell usage (for example, carry chains, flip-flops)
- Total registers
- Registers packed in I/Os
- Memory cell usage
- Number of I/Os
- Global buffer usage
- LUTs
- Percent of utilization

The number of CLBs reported is within a few percentage points of the number reported by Xilinx after placement and routing.

Copyright © 2013 Synopsys, Inc.

Example: Xilinx Spartan-3 Resource Usage Report

```
Resource Usage Report for a_cnt_demo

Mapping to part: xc3s50tq144-4
Cell     usage:
FD       29 uses
GND      2 uses
MUXCY_L  14 uses
XORCY    16 uses
LUT1     21 uses
LUT2     4 uses
LUT3     2 uses
LUT4     17 uses

I/O ports: 9
I/O primitives: 8
OBUF           8 uses
BUFGP          1 use

I/O Register bits: 0
Register bits not including I/Os: 29 (1%)

Global Clock Buffers: 1 of 8 (12%)

Total load per clock:
   a_cnt_demo|osc: 19
   repeated_unit|reference_cnt_inferred_clock: 3

Mapping Summary:
Total LUTs: 44 (2%)
```

Xilinx Net Buffering Report

Click View Log to display the net buffering report in the log file. The report shows how many nets were buffered or had their sources replicated, and the number of segments created for the net. For an explanation of buffering and replication, see *Controlling Buffering and Replication*, on page 446 in the *User Guide*.

Virtex-5 and Virtex-6 Detailed Clock Path Report

For Synplify Premier Virtex-5 and Virtex-6 designs, the log file includes a Detailed Clock Path section for each Detailed Data Path section. There is a Detailed Clock Path section for every clock that is not an ideal clock.

In the following example, Derived Clock shows the clock name derived from clk_r, which is adcm|clk1_derived_clock.

```
Instance / Net                                      Pin       Pin                Arrival     No. of
Name                  Type                          Name      Dir     Delay      Time        Fan Out(s)
-----------------------------------------------------------------------------------------------------------
Start Clock :         CLK
------------
CLK                   Port                          CLK       In      -          0.000       -
CLK                   Net                           -         -       0.000      -           1
CLK_ibuf              IBUF                          I         In      -          0.000       -
CLK_ibuf              IBUF                          O         Out     1.012      1.012       -
CLK_c                 Net                           -         -       0.432      -           1
RX_DCM                DCM                           CLKIN     In      -          1.444       -
---------------
Derived Clock :       adcm|clk1_1_derived_clock
---------------
RX_DCM                DCM                           CLKDV     Out     -1.444     0.000       -
clk1_1                Net                           -         -       0.000      -           1
b3                    BUFG                          I         In      -          0.000       -
b3                    BUFG                          O         Out     0.000      0.000       -
clk1                  Net                           -         -       0.000      -           0
in2_reg[0]            FD                            C         In      -          0.000       -
===========================================================================================================

====================================
Detailed Report for Clock: adcm|clk1_1_derived_clock
====================================

Starting Points with Worst Slack
********************************

                Starting                                                    Arrival
Instance        Reference                     Type     Pin     Net          Time        Slack
                Clock
-----------------------------------------------------------------------------------------------------
in2_reg[0]      adcm|clk1_1_derived_clock     FD       Q       in2_reg[0]   0.450       0.242
in2_reg[1]      adcm|clk1_1_derived_clock     FD       Q       in2_reg[1]   0.450       0.242
in2_reg[2]      adcm|clk1_1_derived_clock     FD       Q       in2_reg[2]   0.450       0.242
```

Custom Timing Reports for Xilinx

In addition to the default timing report, you can generate custom timing reports for some technologies. The default timing report is part of the log file (*project_name*.srr), located in the results directory. This report provides a clock summary, I/O timing summary, and detailed timing information about paths you specify. The custom timing report lets you run targeted timing analysis. For example, you can specify start and end points of paths, and generate a report for only those sections of the design.

Copyright © 2013 Synopsys, Inc.

You must have a Synplify Premier or Synplify Pro license to generate custom-timing reports. This option is only available for certain device technologies.

You generate a custom timing report with the Analysis->Timing Analyst command. See *Generating Custom Timing Reports with STA*, on page 729 in the *User Guide* for a procedure.

Forward-annotation of Xilinx Constraints

By default, the synthesis tool forward-annotates timing constraints to the P&R tool through a Xilinx (`ucf`) file, and placement constraints through a Xilinx `ncf` file. If you do not want to forward-annotate the timing constraints and generate a `ucf` file, select Project -> Implementation Options, go to the Implementation Results panel, and disable Write Vendor Constraint File.

For details of the Xilinx constraints that are forward-annotated, see *Xilinx Constraint Files*, on page 2110.

Xilinx Constraint Files

The synthesis tools generate two files for Xilinx forward-annotation:

`synplicity.ucf`	Timing constraint file
`.ncf`	Physical constraint file

The following synthesis constraints are forward-annotated to Xilinx in the .ucf file:

define_clock define_clock_delay	The way in which the clocks are defined determines whether they are forward-annotated as Xilinx OFFSET or TIMESPEC constraints. See *Forward-annotation of Port Names in EDIF*, on page 2117 for more information.
define_input_delay define_output_delay	The synthesis tool automatically forward-annotates define_input_delay and define_output_delay constraints to the Xilinx .ucf file as OFFSET constraints. (See *Forward-annotation of I/O Delays in Xilinx Designs*, on page 2113.) If you do not want to forward-annotate these constraints, set the syn_forward_io_constraints to 0. Set this globally on the top level VHDL entity/architecture or Verilog module, or use the Attributes panel of the SCOPE spreadsheet. The synthesis tool might automatically relax I/O constraints if the period constraint is too tight. You can override this behavior with the xc_use_timespec_for_io attribute. See *Forward-annotation of Relaxed I/O Constraints for Xilinx*, on page 2117 for details. By default, the synthesis tool does not forward-annotate any input and output delay constraints for ports without a specific define_input_delay or define_output_delay constraint. You can override this behavior by checking the Use clock period for unconstrained IO option on the Constraints panel of the Implementation Options dialog box. Note that if input/output constraints are defined, they are always forward-annotated, regardless of the option setting; this option only affects ports without a specific constraint.
define_false_path	The synthesis tool automatically forward-annotates define_false_path constraint to the Xililnx .ucf file.

Copyright © 2013 Synopsys, Inc.

define_multicycle_path	See *Forward-annotation of Xilinx Multicycle Paths*, on page 2112.
syn_reference_clock	The syn_reference_clock attribute, specifies a clock frequency other than that implied by the signal on the clock pin of the register. The synthesis tool automatically forward-annotates these constraints. For details about this attribute, see *syn_reference_clock*, on page 1490.
define_path_delay	When you specify define_path_delay, the synthesis tool adds the input or output delays to the path delay. The tool forward-annotates a constraint that includes the I/O delay with the path delay. You might see discrepancies with the Xilinx P&R tool, which ignores the I/O delays and only reports the path delay.

The synthesis tool also forward-annotates relationships between different clocks. See the following sections for further details on specific aspects of forward-annotation:

- Forward-annotation of Xilinx Multicycle Paths, on page 2112
- Forward-annotation of I/O Delays in Xilinx Designs, on page 2113
- Forward-annotation of Port Names in EDIF, on page 2117
- Forward-annotation of Xilinx Clock Groups, on page 2115
- Forward-annotation of Xilinx Clock Components, on page 2116
- Forward-annotation of Relaxed I/O Constraints for Xilinx, on page 2117
- Forward-annotation of Compile-point Constraints, on page 2117
- Forward-annotation of Port Names in EDIF, on page 2117

Forward-annotation of Xilinx Multicycle Paths

The tool forward-annotates multicycle paths for registers that are packed in DSPs and RAMs to the UCF file. The -to constraints are forward-annotated as -to/-thru constraints.

Copyright © 2013 Synopsys, Inc.

Forward-annotation of I/O Delays in Xilinx Designs

I/O delays are very important for correctly calculating the timing of a design. In the synthesis tools, these delays are specified with the define_input_delay and define_output_delay constraints. The synthesis tool forward-annotates delay constraints as Xilinx OFFSET constraints in the UCF file, using the following rules:

The synthesis constraints differ from the Xilinx constraints:

- The synthesis constraints specify the external input/output to/from the top-level input/output ports. The input and output delays are the delays outside the FPGA with reference to the clock edge on which external logic is triggered.
- The Xilinx OFFSET delay constraint is a basic timing constraint that specifies the timing relationship between an external clock and its associated data-in or data-out pin. The Xilinx OFFSET constraint can be used in four combinations to declare internal and external FPGA delays:
  - IN-BEFORE
  - IN-AFTER (same as the synthesis input_delay constraint)
  - OUT-AFTER
  - OUT-BEFORE (same as the synthesis output_delay constraint)

This figure illustrates the differences between the synthesis delay and Xilinx OFFSET constraints:

Copyright © 2013 Synopsys, Inc.

To correctly handle these differences when using an input UCF file for synthesis, you must translate the input UCF using the Project->Convert Vendor Constraints command or the command line ucf2sdc utility before you synthesize the design.

The synthesis tool forward-annotates the delays as OFFSET IN-BEFORE and OFFSET OUT-AFTER constraints, so the synthesis delay constraint values have to be adjusted to account for the difference. The delay constraints are forward-annotated in the .ucf file.

For IN-BEFORE or OUT-AFTER constraints that are triggered on the positive edge, the tool calculates the values by subtracting the synthesis delay from the clock period. It forward-annotates the correct values, which are equivalent to, but do not match the synthesis delay values because of the calculation differences.

This example illustrates the differences in the SDC and UCF values. It has a clock period of 12 ns and is triggered on the rising clock edge.

SDC Constraints

```
define_input_delay -default 5.00 -route 0.00 -ref {clk:r}
define_output_delay -default 5.00 -route 0.00 -ref {clk:r}
```

Forward-annotated UCF Constraints

```
OFFSET = IN : 7.000 : BEFORE clk;
OFFSET = OUT : 7.000 : AFTER clk;
```

Forward-annotation of Xilinx Clocks

A clock defined on an instance differs from one defined on the net driving the instance. A clock definition on an instance also differs from a clock definition on a driving port (if the instance is driven directly by the port). The reason for the difference in behavior is because of the following Xilinx definitions:

- Clocks on primary inputs are translated into OFFSET constraints in the .ucf file.
- Other internal clocks are translated into TIMESPEC constraints in the .ucf file.

Copyright © 2013 Synopsys, Inc.

The following example shows these differences using the following circuit:

The following syntax defines the clock with respect to the port. With this syntax, the tool generates an OFFSET constraint is used in the .ucf file. This constraint can take into account a clock path delay.

```
define_clock -name {p:CLK_in} -period 20.000
   -clockgroup default_clkgroup_0
```

Contrast this with the following syntax, where the clock is defined on the output of the instance (with respect to the clock buffer). With this syntax, the tool generates TIMESPEC constraints (from/to) in the .ucf file. These constraints assume that the clock is ideal.

```
define_clock -name {i:U1} -period 20.000
   -clockgroup default_clkgroup_0
```

The resulting slack numbers for the two examples can differ for paths that start/end at I/O ports and end/start at sequential elements. The slack numbers are more pessimistic for a clock defined on a port.

Forward-annotation of Xilinx Clock Groups

If two clocks are in the same clock group, the synthesis tool writes out the UCF for Xilinx forward-annotation so that one clock is a fraction of the other. In this example, clk1 is derived as a fraction of clk2, which signals the place-and-route tool that the two clocks are part of the same clock group.

Copyright © 2013 Synopsys, Inc.

```
NET "clk2" TNM_NET = "clk2";
TIMESPEC "TS_clk2" = PERIOD "clk2" 10.000 ns HIGH 50.00%;
NET "clk1" TNM_NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" "TS_clk2" * 2.000000 HIGH 50.00%;
```

In the following example, the clocks are declared independently, so the place and route tool considers the clocks separately for timing calculation:

```
NET "clk2" TNM_NET = "clk2";
TIMESPEC "TS_clk2" = PERIOD "clk2" 10.000 ns HIGH 50.00%;
NET "clk1" TNM_NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" 20.000 ns HIGH 50.00%;
```

Forward-annotation of Xilinx Clock Components

The synthesis tool can take advantage of the Frequency Synthesis and Phase Shifting features of the following clock components for Xilinx devices:

- Digital Clock Manager (DCM)
- Delay Locked Loop (DLL)
- Mixed Mode Clock Manager (MMCM)

If you are using clock components for on-chip clock generation, you need only define the clock at the primary inputs. The synthesis tool propagates clocks through any number of clock components and automatically generates clocks at the outputs of the components, as needed, taking into account any phase shift or frequency change.

You can specify the phase shift and frequency multiplication parameters using Xilinx standard properties like duty_cycle_correction and clkdv_divide, as well as properties such as clkfx_multiply and clkfx_divide for DCMs, if needed.

The tool also takes into account the fact that these clocks are related (synchronized) to each other, and puts them in the same clock group. However only the clock at the input of a clock component is forward-annotated in the .ucf file, because the back-end tools understand these components and performs clock propagation across them.

Copyright © 2013 Synopsys, Inc.

Forward-annotation of Relaxed I/O Constraints for Xilinx

The synthesis tool constrains input-to-register, register-to-register and register-to-output paths with the frequency constraint. However, if the period constraint is too tight for the input-to-register or register-to-output paths, the synthesis tool tries to relax the constraints to these paths.

If you do not want to relax the I/O constraints, enable the xc_use_timespec_for_io attribute. When enabled, I/O constraints are forward-annotated using the Xilinx TIMESPEC FROM ... TO command, and there is no relaxation of the constraints. For more information on this attribute, see *xc_use_timespec_for_io*, on page 1628.

Forward-annotation of Compile-point Constraints

Constraints applied to instances inside a compile point are forward-annotated. Constraints applied to the interface (ports and bit ports) of a compile point are not forward-annotated. (See *Synthesizing Compile Points*, on page 1109, for information on compile points.).

The Synplify software does not support compile points.

Forward-annotation of Port Names in EDIF

You can control how port names are written in the EDIF output files generated for your designs. This is especially useful for instantiated black-box components (usually IP components). By default, the character case of port names is preserved as it appears in the HDL source code.

You use two attributes for this: one for scalar (individual) port names and the other for bus port names. Specify the attributes on individual components in your HDL source code, or globally by using constraint files and the SCOPE spreadsheet. See *syn_edif_scalar_format*, on page 1286 and *syn_edif_bit_format*, on page 1280 for syntax details.

Copyright © 2013 Synopsys, Inc.

Scalar ports	Use syn_edif_scalar_format to determines if port names are all uppercase or all lowercase (for example, AbCd becomes ABCD).
Bus Ports	Use the syn_edif_bit_format attribute to determine the following: • If names are all uppercase or all lowercase (AbCd becomes ABCD) • If additional characters are appended to the end of names (qout becomes qout_IP) • If the bus range is appended to the end of names (qout becomes qout[31:1]) • The format for a range delimiter when a range is appended (qout<31:0> or qout[31:0])

Integration with Xilinx Tools and Flows

The following describe how the synthesis tools support various tools and flows for Xilinx designs:

- Xilinx ISE Design Suite, on page 2118
- Xilinx Xflow and Xtclsh, on page 2119
- Xilinx Incremental Flow, on page 2119
- Compile Point Remapping in Xilinx Designs, on page 2104
- ISE to Synthesis Flow, on page 2120

Xilinx ISE Design Suite

When you launch the Xilinx ISE Design Suite directly from the synthesis tool, the tool automatically creates a corresponding Xilinx project. Alternatively you can set the ISE Design Suite to run automatically after synthesis. See *Running P&R Automatically after Synthesis*, on page 842 in the *User Guide* for details.

For details about the ISE Design Suite, consult the Xilinx documentation.

Xilinx Xflow and Xtclsh

In earlier releases, the synthesis tool generated an `opt` file that contained the P&R options, and then ran ISE by calling the xflow executable. The new default is for the synthesis tools to generate a Tcl options file (`tcl`) and call the xtclsh executable to run the ISE Design Suite. See *Specifying Xilinx Place-and-Route Options in a Tcl File*, on page 311 in the *User Guide* for information about using a Tcl file.

If you are using the new Tcl flow and the xtclsh executable, note that this flow names files based on the top-level name, whereas xflow named files based on the EDIF. If the top level of your design has a different name than the `edf` generated by the synthesis tools, the difference could affect scripts that are looking for specific files.

The following table lists the differences between the output files generated by the two flows when your synthesis EDIF file is called test.edf and your top-level design is called design:

xflow	test.mrp, test.twr, test.par, ... (Based on the EDIF name)
xtclsh	design.mrp, design.twr, design.par, ... (Based on the top-level design name)

If you want to use the old xflow method, enable the Use Xilinx xflow option in the Device tab of the Implementation Options dialog box. See *Specifying Xilinx Place-and-Route Options in an opt File*, on page 314 in the *User Guide*.

Xilinx Incremental Flow

Xilinx allows you to make incremental updates to your design. The synthesis tools support this capability with the following methodologies:

- Incremental flow for small changes
  See the incremental P&R *Xilinx SmartGuide Flow*, on page 870 in the *User Guide* for a procedure.
- Compile point synthesis with the Xilinx Partition flow using ISE 12.1
  See *Xilinx Partition Flow*, on page 873 in the *User Guide* for a procedure.
- Partition flow for major constraint changes using earlier versions of ISE
  See *Xilinx Partition Flow for Versions Before ISE 12.1*, on page 878 in the *User Guide* for a procedure.

Copyright © 2013 Synopsys, Inc.

Compile-point Synthesis Flow

In technologies that allow it, you can break down your design into smaller synthesis units or *compile points*, so that you can incrementally synthesize your design. A compile point is a module that is treated as a block for incremental mapping. They help to isolate portions of a design so you can stabilize results or improve place-and-route runtimes. When your design is resynthesized, compile points that have already been synthesized are not resynthesized, unless you have changed the HDL source code in such a way that the design logic, constraints, or design mapping options are changed.

See *Synthesizing Compile Points*, on page 1109 of the *User Guide* and *Xilinx Partition Flow*, on page 873 in the *User Guide* for details.

You can also use compile-point synthesis in conjunction with the Xilinx Incremental flow to design, optimize, and lock down a design, one section at a time.

ISE to Synthesis Flow

You can use the ise2syn utility to convert Xilinx projects into synthesis projects. See *Converting Xilinx Projects with ise2syn*, on page 1051 in the *User Guide* and *Xilinx ise2syn Utility*, on page 1017 for details.

Copyright © 2013 Synopsys, Inc.

Virtex and Spartan-II and Later Technologies

This section provides guidelines for using the synthesis tool with devices from the Virtex family and specific Spartan technologies. See the following for details:

- Virtex and Spartan-II and Later Device Mapping Options, on page 2122
- Virtex and Spartan-II and Later set_option Tcl Command, on page 2124
- Virtex and Spartan-II and Later project Tcl Command, on page 2130
- Virtex Components, on page 2130

Copyright © 2013 Synopsys, Inc.

Virtex and Spartan-II and Later Device Mapping Options

Device mapping options are synthesis algorithms to optimize your design. To set these options, select Project->Implementation Options and set the options on the Device tab. You can set other options on other tabs of this dialog box. For descriptions of optimization options and constraints, see *Options Panel*, on page 197 and *Constraints Panel*, on page 200.

The following table lists device mapping options (Device tab) for the Virtex and Spartan-II and latesr technologies. Some options might only be available for certain devices.

Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views and Design Planner, and use them to create collections using the Tcl Find command. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
Clock Conversion	Performs gated and generated clock optimization when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details. The Clock Conversion option is only available with the *Synplify Pro* and *Synplify Premier* products.
Disable I/O Insertion	When enabled, it prevents automatic I/O insertion during synthesis. By default, it is disabled. See *Xilinx I/O Insertion*, on page 2096 for details.
Disable Sequential Optimizations	Enables or disables sequential optimizations for the design. See *Sequential Optimizations in Xilinx Designs*, on page 2102.
Enable Advanced LUT Combining	Prepares the netlist for advanced LUT combining in Virtex-5 and Spartan-6 or newer designs, so that Xilinx ISE P&R can pack into the LUT6_2 depending on available resources. The default is on. See *Advanced LUT Combining*, on page 2082 for additional details.
Enhanced Optimization	In Synplify Premier logic synthesis mode, determines whether the tool runs additional optimizations. See *Logic Synthesis with Enhanced Optimization*, on page 40 in the *User Guide* for more information.

Copyright © 2013 Synopsys, Inc.

Fanout Guide	Sets a guideline for the fanout limit. The default value depends on the target device selected. See the Device tab of the Implementation Options panel for fanout limits. See *Fanout Limits in Xilinx Designs*, on page 2099 for background information. For tips on setting and using fanout limits, see *Setting Fanout Limits*, on page 444 and *Controlling Buffering and Replication*, on page 446 in the *User Guide*.
Read Write Check on RAM	Lets the synthesis tool insert bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations. The synthesis software globally inserts bypass logic around the RAM that read and write to the same address simultaneously. Disable this option, when you cannot simultaneously read and write to the same RAM location and want to minimize overhead logic. For details about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Update Compile Point Timing Data	Determines whether changes to a locked compile point force its parents to be remapped and updated with the new timing model of the child. See *Compile Point Remapping in Xilinx Designs*, on page 2104, for details. Compile points are only supported by the *Synplify Pro* and *Synplify Premier* products.

Copyright © 2013 Synopsys, Inc.

Use Xilinx Xflow	Determines whether the Xilinx xflow or xtclsh flow is used for P&R. For usage information, see *Specifying Xilinx Place-and-Route Options in a Tcl File*, on page 311 and *Specifying Xilinx Place-and-Route Options in an opt File*, on page 314 in the *User Guide*.
Use Xilinx XPartition Flow	Determines whether to use compile points with the Xilinx Partition flow, which compares partitions or modules in the previous and current implementations. Partitions that have not changed are preserved from the previous implementation. This feature saves a significant amount of time in that place and route for the entire design does not need to be rerun. For details, see *Xilinx Partition Flow*, on page 873 in the *User Guide*.
Verification Mode	Ensures that your synthesis output files are compatible with the verification tools. See *Verification Mode in Xilinx Designs*, on page 2103 for details. This feature is only available for the *Synplify Pro* and *Synplify Premier* products.

You can also use the corresponding set_option Tcl command to enable/disable these options. See *Virtex and Spartan-II and Later set_option Tcl Command, on page 2124* for more information.

Virtex and Spartan-II and Later set_option Tcl Command

Specifies the implementation options for a design. These are the options you set for synthesis such as the target technology, device architecture and synthesis styles. The set_option Tcl command lets you specify the same implementation options as you do through the dialog box displayed in the Project view with Project->Implementation Options.

This section provides information on some of the specific options for the Virtex family and the Spartan-II and later technologies.

The following table lists the options for these families. See *set_option*, on page 1679 or enter help set_option in the Tcl Script window for a full list of syntax and options.

OPTION	DESCRIPTION
Target Technology Options	
-technology *keyword*	Sets the target technology for the implementation. See *Virtex and Spartan Technologies*, on page 2129 for the list of technology keywords and their corresponding Xilinx libraries.
-part *partName*	Specifies a part for the implementation. Refer to the Implementation Options dialog box for available choices (*Implementation Options Command*, on page 194).
-speed_grade *value*	Sets the speed grade for the implementation. Refer to the Implementation Options dialog box for available choices (*Implementation Options Command*, on page 194).
-package *packageName*	Specifies the package for the implementation. Refer to the Implementation Options dialog box for available choices (*Implementation Options Command*, on page 194).
Optimization Options	
-automatic_compile_point 1|0	Enables/disables the automatic compile point option, which can analyze a design and identify modules that can automatically be defined as compile points and mapped in parallel using Multiprocessing. For more information, see *The Automatic Compile Point Flow*, on page 1109 in the *User Guide*.
-block 1 | 0	Controls I/O insertion during synthesis. The default is 0, where I/Os are inserted automatically. To disable I/O insertion, set it to 1. See *Xilinx I/O Insertion*, on page 2096 for a description.
-do_synplify_place 1|0	*Synplify Premier tools; Obsolete Command* Since the Synplify global placer is run by default with the Physical Synthesis and Physical Plus flows, this option is no longer needed. To run the Xilinx global placer, use the syn_use_xilinx_placement attribute.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-enable_prepacking 1|0	Prepares the netlist for advanced LUT combining in Virtex-5 and Spartan-6 or newer designs, so that Xilinx ISE can pack into LUT6_2s depending on available resources. The default is on. See *Advanced LUT Combining*, on page 2082 for additional details.
-enhanced_optimization 1|0	*Synplify Premier tools* Enables/disables additional optimizations during logic synthesis to get better QoR. This is the default. When disabled, the tool runs standard logic synthesis without the additional optimizations. See *Logic Synthesis with Enhanced Optimization*, on page 40 of the *User Guide* for details.
-fast_synthesis 1|0	*Synplify Premier tools* Enables/disables some mapper optimizations to improve logic synthesis runtimes. See *Fast Synthesis*, on page 1061 of the *User Guide* for details.
-fix_gated_and_ generated_clocks 1|0	Performs gated- and generated-clock optimizations when enabled. See *Working with Gated Clocks*, on page 466 and *Optimizing Generated Clocks*, on page 499 in the *User Guide* for details.
-hamming3 1|0	*Synplify Premier tools* When enabled, a Hamming 3 encoding is used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers.
-maxfan *value*	Sets the fanout limit. The default value depends on the target device selected. See the Device tab of the Implementation Options panel for fanout limits. See *Fanout Limits in Xilinx Designs*, on page 2099 for details.
-no_sequential_opt 1|0	Enables or disables the sequential optimizations for the design. Note that unused registers are always removed from the design regardless of this option setting. The default value is false or 0; therefore, sequential optimizations are performed. Values can be 1 or true, 0 or false. Disabling sequential optimizations can increase delay times and area requirements, and effectively disables the FSM Compiler.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-par_use_xflow 1 | 0	Determines whether the old Xilinx xflow is used. This option is turned off (0) by default. Set it to 1 to use xflow. See *Xilinx Xflow and Xtclsh*, on page 2119 for details.
-par_use_xpartition 1|0	Determines whether to use compile points with the Xilinx XPartition flow, which compares partitions or modules in the previous and current implementations. Partitions that have not changed are preserved from the previous implementation. This feature saves a significant amount of time in that place and route for the entire design does not need to be rerun. The default is 0.
-physical_synthesis 1|0	*Synplify Premier tools* Enables/disables physical synthesis, which includes global placement, placement-aware optimizations, and graph-based placement.
-pipe 1 | 0	Determines whether you run designs at a faster frequency by moving registers after the multiplier into the multiplier. The default is 0. See *Pipelining*, on page 426 of the *User Guide* for information on how to use it. The Synplify software does not support pipelining.
-power 1|0	*Synplify Premier tools* The software allows you to reduce power for designs that contain Xilinx block RAM and DSP48 components. For more information, see *Power Optimization for Xilinx Designs*, on page 2105.
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
-resource_sharing 1|0	Enables/disables resource sharing. See *Sharing Resources*, on page 448 in the *User Guide* for information about using it.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-retiming 1 | 0	Determines whether you use retiming to optimize your design. When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled). You must also enable -pipe. See *Retiming*, on page 430 of the *User Guide* for information on how to use it. The Synplify software does not support retiming.
-run_physical_plus 1|0	*Synplify Premier tools* Runs the Physical Plus modular flow. After logic synthesis with Enhanced Optimization has been run, you can incrementally run physical synthesis to implement netlist and timing changes using the Process view GUI. See *Xilinx Physical Plus*, on page 64 in the *User Guide* for more information.
-run_prop_extract 1 | 0	Determines whether the tool extracts properties for annotation. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
-safe_case 1|0	Implements state machines with user-defined recovery logic. For counters and other sequential logic, the tool prevents optimizations based on reachability analysis.
-symbolic_fsm_compiler 1 | 0 **-autosm 1 | 0**	Enables/disables the FSM compiler. Controls the use of FSM synthesis for state machines. The default is false (FSM Compiler disabled). Value can be 1 or true, 0 or false. See *Running the FSM Compiler*, on page 455 in the *User Guide* for details of its use.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-update_models_cp 1 | 0	Determines whether changes to a locked compile point force the parents to be remapped to use the new timing model of the child. See *Compile Point Remapping in Xilinx Designs*, on page 2104, for details. • 1 Forces remapping of parents • 0 Does not remap parents The Synplify software does not support compile points.
-use_fsm_explorer 1|0	Enables/disables the FSM Explorer. See *Running the FSM Explorer*, on page 459 in the *User Guide* for information about using it. The Synplify tool does not support FSM Explorer.
-verification_mode 1 | 0	Determines whether synthesis output files are compatible with the verification tools. To use this feature, you must have a Synplify Pro or Synplify Premier license. • 1 Ensures that synthesis output files are compatible with the verification tools. • 0 When disabled, produces synthesis output that performs the standard default optimizations.

Virtex and Spartan Technologies

The following table lists the supported Xilinx technologies and corresponding parameter values for the set_option -technology Tcl command.

Technology	Value	Technology	Value
Spartan-II	spartan2	Virtex-6	virtex6
Spartan-IIE	spartan2e	Virtex-7	virtex7
Spartan-3	spartan3	Kintex-7	kintex7
Spartan-3E	spartan3e	Artix-7	artix7
Spartan-IIE Automotive	aspartan2e	QPro Virtex Hi-Rel	qprovirtexh
Spartan-3 Automotive	aspartan3	QPro Virtex Rad-Hard	qprovirtexr

Copyright © 2013 Synopsys, Inc.

Technology	Value	Technology	Value
Spartan-6	spartan6	QPro Virtex-II Military	qprovirtex2
Virtex-II	virtex2	QPro Virtex-II Rad Tolerant	qprorvirtex2
Virtex-II Pro	virtex2p	QPro Virtex-E Military	qprovirtex-em
Virtex-4	virtex4	QPro Virtex Hi-Rel	qprovirtexh
Virtex-5	virtex5		

Virtex and Spartan-II and Later project Tcl Command

You can use the project Tcl command to specify the same result file formats and filenames as are available from the Implementation Results panel of the Implementation Options dialog box.

This table lists the options for the project Tcl command for the Virtex and Spartan-II and later technologies. See project, on page 1662 for complete syntax and options for this command, or type `help project` in the Tcl Script window:

Option	Description
-result_file *value*	Specifies the name of the synthesized netlist for the implementation.
-result_format edif	Specifies the synthesis result file format for the implementation. The format must be edif (lower-case letters).

Virtex Components

The synthesis tools support some Virtex-specific components. See the following for details:

- DSP Components in Virtex Designs, on page 2081
- Wide Adders in Virtex-5 and Virtex-6 Designs, on page 2082
- Advanced LUT Combining, on page 2082

Copyright © 2013 Synopsys, Inc.

Xilinx CPLDs

This section provides guidelines for using the synthesis tool with Xilinx CPLD devices. The Synplify Premier software does not support the Xilinx CPLD technologies. For descriptions of Xilinx CPLD support, see the following:

- Xilinx CPLD Device Mapping Options, on page 2131
- Xilinx CPLD set_option Tcl Command, on page 2132
- Xilinx CPLD Technologies, on page 2134
- Xilinx CPLD project Command, on page 2135

The applicable CPLD technologies include CoolRunner-II, CoolRunner-II Automotive, CoolRunner XPLA3, XC9500, XC9500XL, and XC9500XV.

Xilinx CPLD Device Mapping Options

Device mapping options are synthesis algorithms that you use to optimize your design. You can set the device mapping options for the design in the UI, (Implementation Options->Device) or through the equivalent Tcl command. You can set other options on other tabs of this dialog box. For descriptions of optimization options and constraints, see *Options Panel*, on page 197 and *Constraints Panel*, on page 200.

This table alphabetically lists the device mapping options for CPLD devices:

Annotated Properties for Analyst	Annotates the design with properties after the design is compiled. You can view these properties in the schematic views and Design Planner, and use them to create collections using the Tcl Find command. See *Annotating Timing Information in the Schematic Views*, on page 723 in the *User Guide* for more information.
Disable I/O Insertion	When enabled, it prevents automatic I/O insertion during synthesis. By default, it is disabled. See *Xilinx I/O Insertion*, on page 2096 for details.
Disable Sequential Optimizations	Enables or disables sequential optimizations for the design. See *Sequential Optimizations in Xilinx Designs*, on page 2102.

Copyright © 2013 Synopsys, Inc.

Fanout Guide	Sets a guideline for the fanout limit. The default value depends on the target device selected. See the Device tab of the Implementation Options panel for fanout limits. See *Fanout Limits in Xilinx Designs*, on page 2099 for background information. For tips on setting and using fanout limits, see *Setting Fanout Limits*, on page 444 and *Controlling Buffering and Replication*, on page 446 in the *User Guide*.
Read Write Check on RAM	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
Resolve Mixed Drivers	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.
Use Xilinx Xflow	Determines whether the Xilinx xflow or xtclsh flow is used for P&R. For usage information, see *Specifying Xilinx Place-and-Route Options in a Tcl File*, on page 311 and *Specifying Xilinx Place-and-Route Options in an opt File*, on page 314 in the *User Guide*.
Verification Mode	Ensures that your synthesis output files are compatible with the verification tools. See *Verification Mode in Xilinx Designs*, on page 2103 for details. To use this feature, you must have the Synplify Pro or Synplify Premier software.

You can also use the set_option Tcl command to enable/disable these options. See *Xilinx CPLD set_option Tcl Command* below for more information.

Xilinx CPLD set_option Tcl Command

Specifies the implementation options for a design from the command line. You set these options for the target technology, device architecture and synthesis styles. This table lists the set_option options for Xilinx CPLD technologies (GUI equivalents are in the Device and Options tabs of the Implementation Options dialog box:)

OPTION	DESCRIPTION
Target Technology Options	
-technology *keyword*	Sets the target technology for the implementation. See *Xilinx CPLD Technologies*, on page 2134 for the list of technology keywords and their corresponding Xilinx libraries.
-part *partName*	Specifies a part for the implementation. Refer to the Implementation Options dialog box for available choices – see *Implementation Options Command*, on page 194.
-speed_grade *value*	Sets the speed grade for the implementation. Refer to the Implementation Options dialog box for available choices – see *Implementation Options Command*, on page 194.
-package *packageName*	Specifies the package for the implementation. Refer to the Implementation Options dialog box for available choices – see *Implementation Options Command*, on page 194.
Optimization Options	
-block 1 | 0	Prevents I/O insertion during synthesis. The default is false, (I/O insertion mode). To disable I/O insertion, set this option to 1.
-maxfan *value*	Sets the fanout limit. The default value depends on the target device selected. See the Device tab of the Implementation Options panel for fanout limits.
-no_sequential_opt 1|0	Enables (0) or disables (1) sequential optimizations for a design. By default, sequential optimizations are performed. When disabled, delay time and area size can increase, and the FSM Compiler is effectively disabled. See *Sequential Optimizations in Xilinx Designs*, on page 2102 for details.
-par_use_xflow 1|0	When enabled, you can use the old xflow function to run the Xilinx ISE Design Suite. This option is turned off (0) by default.
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver.

Copyright © 2013 Synopsys, Inc.

OPTION	DESCRIPTION
-resource_sharing 1|0	Enables/disables resource sharing. See *Sharing Resources*, on page 448 in the *User Guide* for information about using it.
-RWCheckOnRam 1 | 0	If read or write conflicts exist for the RAM, enable this option to insert bypass logic around the RAM to prevent simulation mismatch. Disabling this option does not generate bypass logic. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle*, on page 1467.
-symbolic_fsm_compiler 1 | 0 **-autosm 1 | 0**	Enables/disables the FSM compiler. Controls the use of FSM synthesis for state machines. The default is false (FSM Compiler disabled). Value can be 1 or true, 0 or false. See *Running the FSM Compiler*, on page 455 in the *User Guide* for details of its use.
-use_fsm_explorer 1|0	Enables/disables the FSM Explorer. See *Running the FSM Explorer*, on page 459 in the *User Guide* for information about using it. Available only for the Synplify Pro and Synplify Premier tools.
-verification_mode 1 | 0	When enabled (1), ensures that your synthesis output files are compatible with the verification tools. When disabled (0), produces synthesis output that performs the standard default optimizations. This option is only available for the Synplify Pro and Synplify Premier products.

Xilinx CPLD Technologies

The following table lists the supported Xilinx CPLD technologies and corresponding parameter values for set_option -technology.

Technology	Parameter	Technology	Parameter
CoolRunner-II	coolrunner2	XC9500	xc9500
CoolRunner-II Automotive	acoolrunner2	XC9500XL	xc9500-XL
CoolRunner XPLA3	coolrunner		

Copyright © 2013 Synopsys, Inc.

Xilinx CPLD project Command

You can use the project Tcl command to specify the same result file formats and filenames as are available through Implementation Results panel of the Implementation Options dialog box (see *Implementation Results Panel*, on page 203). This section provides information on specific options for the project command for the Xilinx CPLD technologies. See *project*, on page 1662 for a complete list of options or enter help project in the Tcl Script window.

The project command has the following option specific to these technologies:

Option	Description
-result_format edif |xnf	Specifies the synthesis result file format for the implementation. The format must be edif or xnf (lower-case letters).
-result_file *value*	Specifies the name of the synthesized netlist for the implementation.

Copyright © 2013 Synopsys, Inc.

Xilinx Attribute and Directive Summary

The following table summarizes the synthesis and Xilinx-specific attributes and directives available with the Xilinx Technology. Complete descriptions and examples can be found in Chapter 16, *Attribute and Directive Syntax*.

Attribute/Directive	Description
black_box_pad_pin	Specifies that a pin on a black box is an I/O pad. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
black_box_tri_pins	Specifies that a pin on a black box is a tristate pin. It is applied to a component, architecture, or module, with a value that specifies the set of pins on the module or entity.
CLOCK_DEDICATED_ROUTE	Determines if the software enforces placement rules for the specified architecture.
diff_term	Specifies Differential Termination for true differential input I/O standards.
full_case	Specifies that a Verilog case statement has covered all possible cases.
loop_limit	Specifies a loop iteration limit for for loops.
parallel_case	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
syn_allow_retiming	Specifies if registers can be moved during retiming. The Synplify tool does not support retiming.
syn_allowed_resources	Specifies allowed resources for compile points. The value assigned to a given compile point includes the resources used by its children (at all levels). The Synplify tool does not support compile points.
syn_assign_to_region	*Xilinx Virtex-5 and later devices in the Synplify Premier tools* Assigns logic to regions created on the device.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_assign_to_slr	Available for *Xilinx Virtex-7* Stacked Silicon Interconnect (SSI) technology that allows multiple dies to be combined in a single device. Each die is called a Super Logic Region (SLR) and is connected with 49 Super Long Lines (SLL) per column between each adjacent SLR.
syn_auto_insert_bufg	*Synplify Premier tools; Xilinx Virtex-6 devices* Automatically inserts the BUFG primitive on high fanout nets to help reduce congestion for a physical synthesis design.
syn_auto_partition	Allows automatic partitioning to occur on Stacked Silicon Interconnect (SSI) technology that combines multiple dies on a device. This partitioning helps to place logic optimally for highest die utilization and routibility of the design. You can set this attribute to 0 or 1 so that partitioning constraints can be forward annotated in the XDC file to guide Vivado place and route.
syn_black_box	Defines a black box for synthesis.
syn_clean_reset	*Xilinx Virtex-4, Virtex-5, and Virtex-6 devices* Changes asynchronous reset registers, which cannot go into DSP blocks, into synchronous reset logic.
syn_clock_priority	Establishes clock priority between clocks. The priority is forward-annotated in the UCF file.
syn_diff_io	Controls the inference of I/O buffers.
syn_direct_enable	Identifies which signal to use as the enable input to an enable flip-flop when multiple candidates are possible.
syn_direct_reset	Controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the reset when the design has a conditioned reset on multiple candidates.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_direct_set	Controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the set when the design has a conditioned set on multiple candidates.
syn_dspstyle	*Virtex-4, Virtex-5, and Virtex-6 designs* Determines if an operator, register, or module/architecture is placed in the DSP component.
syn_edif_bit_format	Controls the character formatting and style of bus signal names, port names, and vector ranges in the EDIF output file.
syn_edif_scalar_format	Controls the character formatting of scalar signal and port names in the EDIF output file.
syn_encoding	Specifies the encoding style for state machines.
syn_enum_encoding	Specifies the encoding style for enumerated types (VHDL only).
syn_expand_coreloc	Writes output for all RAM, DSP, and I/O pad locations to the coreloc.sdc file. By default, only I/O pad locations are written to the `coreloc.sdc` file.
syn_fast_auto	Speeds up signal transitions on output ports.
syn_forward_io_constraints	Enables forward annotation of I/O constraints to the Xilinx ISE Design Suite.
syn_fsm_correction	Specifies a Hamming 3 encoding used in conjunction with error correction logic to automatically correct single-bit errors in the FSM state registers.
syn_global_buffers	Sets the number of global buffers to use in a design.
syn_hier	Controls the handling of hierarchy boundaries of a module or component during optimization and mapping.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_highrel_ioconnector	Adds I/O connectors that interface to modules with distributed TMR or DWC, which allows you to access the signals within its boundaries ensuring their inputs and outputs are not a single-point of failure.
syn_isclock	Specifies that a black-box input port is a clock, even if the name does not indicate it is one.
syn_insert_buffer (Xilinx)	Inserts a clock buffer according to the Xilinx specified values.
syn_insert_pad	Inserts I/O buffers to either ports or nets.
syn_keep	Prevents the internal signal from being removed during synthesis and optimization.
syn_loc	Specifies pin locations for I/O pins and cores, and forward-annotates this information to the ISE Design Suite.
syn_macro	Prevents instantiated macros from being merged or otherwise optimized away.
syn_map_dffrs	Allows the synthesis software to handle registers with asynchronous set and reset for Virtex-6 and Spartan-6 devices.
syn_maxfan	Overrides the default fanout guide for an individual input port, net, or register output.
syn_multstyle	Determines implementation style for multipliers.
syn_netlist_hierarchy	Determines if the EDIF output netlist is flat or hierarchical.
syn_noarrayports	Prevents the ports in the EDIF output netlist from being grouped into arrays, and leaves them as individual signals.
syn_noclockbuf	Controls the automatic insertion of global clock buffers.
syn_no_compile_point	Ignores modules as compile points in the Automatic Compile Point flow.
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_packer_effort_level	Determines the effort level for packing during Synplify Premier global placement.
syn_pad_type	Specifies an I/O buffer standard for Virtex-II and later, and Spartan-3 and later technologies
syn_pipeline	Specifies that registers be moved into ROMs or multipliers to improve frequency. The Synplify tool does not support pipelining,
syn_placer_effort_level	Set this attribute to only place the FIFO, RAM, and DSP blocks in the design and forward-annotate them for P&R or optimize the design for routing at the expense of timing.
syn_preserve	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
syn_probe	Adds probe points for testing and debugging. The Synplify software does not support probe point insertion.
syn_radhardlevel	Enables the triple modular redundancy and voting logic in the design.
syn_ramstyle	Determines the way in which RAMs are implemented.
syn_reduce_controlset_size (Xilinx)	Controls the minimum size of the unique control-set on which control-set optimizations can occur.
syn_reference_clock	Specifies a clock frequency other than that implied by the signal on the clock pin of the register.
syn_replicate	Disables replication.
syn_romstyle Attribute (Xilinx)	Determines how ROM architectures are implemented.
syn_rw_conflict_logic	Allows synthesis to NOT automatically infer a block RAM when unnecessary, and ensures that the tool does NOT insert bypass logic around the RAM to prevent a simulation mismatch.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_safe_case	Enables/disables the safe case option. When enabled, the high reliability safe case option turns off sequential optimizations for counters, FSM, and sequential logic to increase the circuit's reliability.
syn_sharing	Specifies resource sharing of operators.
syn_slow	Increases transition time of a specific output port or ports.
syn_srlstyle	Determines how to implement the sequential shift (seqShift) components. This attribute applies to Virtex technologies.
syn_state_machine	Determines if the FSM Compiler extracts a structure as a state machine.
syn_tco<n>	Defines timing clock to output delay through the black box. The *n* indicates a value between 1 and 10.
syn_tpd<n>	Specifies timing propagation for combinational delay through the black box. The *n* indicates a value between 1 and 10.
syn_tristate	Specifies that a black-box pin is a tristate pin.
syn_tristatetomux	Converts tristate drivers that drive nets below a certain limit to multiplexers.
syn_tsu<n>	Specifies the timing setup delay for input pins, relative to the clock. The *n* indicates a value between 1 and 10.
syn_use_xilinx_placement	Specifies that Xilinx global placement is used for the Physical Plus and Physical Synthesis flows.
syn_useenables	Prevents generation of registers with clock enable pins.
syn_useioff (Xilinx)	Packs flip-flops in the I/Os to improve input/output path timing.
syn_user_instance	Prevents user-instantiated instances from being merged or otherwise optimized away.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
syn_vote_loops	Specifies whether or not to add any synchronous voter to a sequential feedback path for a distributed TMR module.
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
xc_area_group	Assigns a compile point to a Xilinx area group. The Synplify tool does not support compile points.
xc_clockbuftype	Specifies that a clock port use the Clock Delay Locked Loop primitive, CLKDLL.
xc_fast	Speeds up the transition time of the output driver. The Synplify Premier software does not support this attribute.
xc_fast_auto	Controls the instantiation of fast output buffers in Virtex designs.
xc_global_buffers	Controls the number of global buffers to use in a design.
xc_isgsr	Specifies that a black-box port is connected to an internal STARTUP block, and prevents the synthesis tool from inferring a STARTUP block. This attribute applies to the XC4000 technologies. The Synplify Premier software does not support this attribute.
xc_padtype	Specifies an I/O buffer standard in Virtex designs.
xc_pullup/xc_pulldown	Specifies that a port is a pullup or pull-down port.
xc_rloc	Specifies the relative locations of all instances with the same xc_uset attribute value.
xc_slow	*CPLD devices* Specifies a slow transition time for the driver of an output port. The Synplify Premier software does not use this attribute.

Copyright © 2013 Synopsys, Inc.

Attribute/Directive	Description
xc_use_keep_hierarchy	Preserves the hierarchy of the marked module for diagnostics and timing simulation.
xc_use_rpms	Specifies how to handle relationally placed macros (RPMs) during global placement and physical synthesis.
xc_use_timespec_for_io	Uses the TIMESPEC command to write out I/O constraints. The Synplify tool does not support this attribute.

Copyright © 2013 Synopsys, Inc.

APPENDIX F

Verilog Example Code

This appendix contains the code samples that are referenced by the corresponding chapter.

Copyright © 2013 Synopsys, Inc.

Example - Constant function

```
module ram
// Verilog 2001 ANSI parameter declaration syntax
  #(parameter depth= 129,
  parameter width=16 )
// Verilog 2001 ANSI port declaration syntax
(input clk, we,
  // Calculate addr width using Verilog 2001 constant function
  input [clogb2(depth)-1:0] addr,
  input [width-1:0] di,
  output reg [width-1:0] do );
function integer clogb2;
input [31:0] value;
     for (clogb2=0; value>0; clogb2=clogb2+1)
     value = value>>1;
  endfunction
reg [width-1:0] mem[depth-1:0];
always @(posedge clk) begin
  if (we)
     begin
        mem[addr]<= di;
        do<= di;
     end
     else
        do<= mem[addr];
     end
endmodule
```

Back

Example – RAM Inferencing

```
module ram_test(q, a, d, we, clk);
output reg [7:0] q;
input [7:0] d;
input [7:0] a;
input clk, we;
reg [7:0] mem [255:0] ;
always @(posedge clk) begin
   if(we)
      mem[a] <= d;
   end
always @ (posedge clk)
q = mem[a];
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - RAM with registered output

```
module ram_test(q, a, d, we, clk);
output [7:0] q;
input [7:0] d;
input [6:0] a;
input clk, we;
reg [7:0] q;
reg [7:0] mem [127:0];
always @(posedge clk) begin
   q <= mem [a];
   if(we)
      mem[a] <= d;
   end
endmodule
```

Back

Example - RAM registered read address

```
module ram_test(q, a, d, we, clk);
output [7:0] q;
input [7:0] d;
input [6:0] a;
input clk, we;
reg [6:0] read_add;
reg [7:0] mem [127:0];
always @(posedge clk) begin
   if(we)
      mem[a] <= d;
      read_add <= a;
   end

assign q = mem[read_add];
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Asynch FSM with continuous assignment

```
module async1 (out, g, d);
output out;
input g, d;
assign out = g & d | !g & out | d &out;
endmodule
```

Back

Example - Asynch FSM with always block

```
module async2 (out, g, d);
output out;
input g, d;
reg out;
always @(g or d or out)
begin
  out = g & d | !g & out | d & out;
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - FSM coding style

```
module FSM1 (clk, rst, enable, data_in, data_out, state0, state1,
   state2);
input clk, rst, enable;
input [2:0] data_in;
output data_out, state0, state1, state2;
/* Defined state labels; this style preferred over `defines*/
parameter deflt=2'bxx;
parameter idle=2'b00;
parameter read=2'b01;
parameter write=2'b10;
reg data_out, state0, state1, state2;
reg [1:0] state, next_state;
/* always block with sequential logic*/
always @(posedge clk or negedge rst)
   if (!rst) state <= idle;
   else state <= next_state;
/* always block with combinational logic*/
always @(state or enable or data_in) begin
   /* Default values for FSM outputs*/
   state0 <= 1'b0;
   state1 <= 1'b0;
   state2 <= 1'b0;
   data_out <= 1'b0;
   case (state)
      idle : if (enable) begin
         state0 <= 1'b1;
         data_out <= data_in[0];
         next_state <= read;
      end
      else begin
         next_state <= idle;
      end
      read : if (enable) begin
         state1 <= 1'b1;
         data_out <= data_in[1];
         next_state <= write;
      end
      else begin
         next_state <= read;
      end
```

```
            write : if (enable) begin
               state2 <= 1'b1;
               data_out <= data_in[2];
               next_state <= idle;
            end
            else begin
               next_state <= write;
            end
            /* Default assignment for simulation */
            default : next_state <= deflt;
         endcase
      end
      endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Predictable Success

APPENDIX G

SystemVerilog Example Code

This appendix contains the code samples that are referenced by the corresponding chapter.

Copyright © 2013 Synopsys, Inc.

Example - Initializing Unpacked Array to Default Value

```
parameter WIDTH = 2;
typedef reg [WIDTH-1:0] [WIDTH-1:0] MyReg;
module top (
   input logic Clk,
   input logic Rst,
   input MyReg DinMyReg,
   output MyReg DoutMyReg );
MyReg RegMyReg;

always@(posedge Clk, posedge Rst) begin
   if(Rst) begin
      RegMyReg <= '{default:0};
      DoutMyReg <= '{default:0};
   end
   else begin
      RegMyReg <= DinMyReg;
      DoutMyReg <= RegMyReg;
   end
end
endmodule
```

Back

Example - Initializing Unpacked Array Under Reset Condition

```
parameter WIDTH = 2;
typedef reg [WIDTH-1:0] [WIDTH-1:0] MyReg;
module top (
   input logic Clk,
   input logic Rst,
   input MyReg DinMyReg,
   output MyReg DoutMyReg );
MyReg RegMyReg;

always@(posedge Clk, posedge Rst) begin
   if(Rst) begin
      RegMyReg <= `{2'd0, 2'd0};
      DoutMyReg <= `{2'd0, 2'd0};
   end
   else begin
      RegMyReg <= DinMyReg;
      DoutMyReg <= RegMyReg;
   end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Aggregate Assignment in Compilation Unit

```
// Start of compilation unit
parameter WIDTH = 2;
   typedef struct packed {
      int r;
      longint g;
      byte b;
      int rr;
      longint gg;
      byte bb;
   } MyStruct [WIDTH-1:0];
const MyStruct VarMyStruct = `{int:1,longint:10,byte:8'h0B} ;
const MyStruct ConstMyStruct =
   `{int:1,longint:$bits(VarMyStruct[0].r),byte:8'hAB} ;

module top (
   input logic Clk,
   input logic Rst,
   input MyStruct DinMyStruct,
   output MyStruct DoutMyStruct );
MyStruct StructMyStruct;

always@(posedge Clk, posedge Rst)
begin
   if(Rst) begin
      StructMyStruct <= VarMyStruct;
      DoutMyStruct <= ConstMyStruct;
   end
   else begin
      StructMyStruct <= DinMyStruct;
      DoutMyStruct <= StructMyStruct;
   end
end
endmodule
```

Back

Example - Aggregate Assignment in Package

```
package MyPkg;
parameter WIDTH = 2;
   typedef struct packed {
      int r;
      longint g;
      byte b;
      int rr;
      longint gg;
      byte bb;
   } MyStruct [WIDTH-1:0];
const MyStruct VarMyStruct = `{int:1,longint:10,byte:8'h0B} ;
const MyStruct ConstMyStruct =
   `{int:1,longint:$bits(VarMyStruct[0].r),byte:8'hAB} ;
endpackage : MyPkg
import MyPkg::*;

module top (
   input logic Clk,
   input logic Rst,
   input MyStruct DinMyStruct,
   output MyStruct DoutMyStruct );
MyStruct StructMyStruct;

always@(posedge Clk, posedge Rst) begin
   if(Rst) begin
      StructMyStruct <= VarMyStruct;
      DoutMyStruct <= ConstMyStruct;
   end
   else begin
      StructMyStruct <= DinMyStruct;
      DoutMyStruct <= StructMyStruct;
   end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Initializing Specific Data Type

```
parameter WIDTH = 2;
typedef struct packed {
   byte r;
   byte g;
   byte b; }
MyStruct [WIDTH-1:0];
module top (
   input logic Clk,
   input logic Rst,
   input MyStruct DinMyStruct,
   output MyStruct DoutMyStruct );
MyStruct StructMyStruct;

always@(posedge Clk, posedge Rst) begin
   if(Rst) begin
      StructMyStruct <= '{byte:0,byte:0};
      DoutMyStruct <= '{byte:0,byte:0};
   end
   else begin
      StructMyStruct <= DinMyStruct;
      DoutMyStruct <= StructMyStruct;
   end
end
endmodule
```

Back

Example - Including Block Name with end Keyword

```
module src (in1,in2,out1,out2);
input in1,in2;
output reg out1,out2;
reg a,b;
always@(in1,in2)
   begin : foo_in
      a = in1 & in2;
      b = in2 | in1;
   end : foo_in
always@(a,b)
   begin : foo_value
      out1 = a;
      out2 = b;
   end : foo_value
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - always_comb Block

```
module test01 (a, b, out1);
input a,b;
output out1;
reg out1;
always_comb
begin
   out1 = a & b;
end
endmodule
```

Back

Example - always_ff Block

```
module Test01 (a,b,clk,out1);
input a,b,clk;
output out1;
reg out1;
always_ff@(posedge clk)
   out1 <= a & b;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - always_latch Block

```
module Test01 (a,b,clk,out1);
input a,b,clk;
output out1;
reg out1;

always_latch
begin
   if(clk)
   begin
      out1 <= a & b;
   end
end
endmodule
```

Back

Example - Local Variable in Unnamed Block

```
module test(in1,out1);
input [2:0] in1;
output [2:0] out1;
integer i;
wire [2:0] count;
reg [2:0] temp;
assign count = in1;

always @ (count)
begin   // unnamed block
   integer i; //local variable
   for (i=0; i < 3; i = i+1)
      begin : foo
      temp = count + 1;
   end
end
assign out1 = temp;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Compilation Unit Access

```
//$unit_4_state begin
logic foo_logic = 1'b1;
//$unit_4_state ends
module test (
input logic data1,
input clk,
output logic out1,out1_local );
//local variables
logic foo_logic = 1'b0;
///////
always @(posedge clk)
begin
   out1 <= data1 ^ $unit::foo_logic; //Referencing
      //the compilation unit value.
   out1_local <= data1 ^ foo_logic; //Referencing the
      //local variable.
end
endmodule
```

Back

Example - Compilation Unit Constant Declaration

```
//$unit begin
const bit foo_bit = "11";
const byte foo_byte = 8'b00101011;
//$unit ends
module test (clk, data1, data2, out1, out2);
input clk;
input bit data1;
input byte data2;
output bit out1;
output byte out2;

always @(posedge clk)
begin
   out1 <= data1 | foo_bit;
   out2 <= data2 & foo_byte;
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Compilation Unit Function Declaration

```
parameter fact = 2;
function automatic [63:0] factorial;
input [31:0] n;
   if (n==1)
      return (1);
   else
      return (n * factorial(n-1));
endfunction

module src (input [1:0] a, input [1:0] b, output logic [2:0] out);
always_comb
begin
   out = a + b + factorial(fact);
end
endmodule
```

Back

Example - Compilation Unit Net Declarations

```
//$unit
wire foo = 1'b1;
//End of $unit
module test (
input data,
output dout );
assign dout = data * foo;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Compilation Unit Scope Resolution

```
//$unit begins
parameter width = 4;
//$unit ends
module test (data,clk,dout);
parameter width = 8; // local parameter
input logic [width-1:0] data;
input clk;
output logic [width-1:0] dout;

always @(posedge clk)
begin
   dout <= data;
end
endmodule
```

Back

Example - Compilation Unit Task Declaration

```
parameter FACT_OP = 2;
task automatic factorial(input integer operand,
   output [1:0] out1);
integer nFuncCall = 0;
begin
   if (operand == 0)
   begin
      out1 = 1;
   end
   else
   begin
      nFuncCall++;
      factorial((operand-1), out1);
      out1 = out1 * operand;
   end
end
endtask

module src (input [1:0] a, input [1:0] b, output logic [2:0] out);
logic [1:0] out_tmp;
always_comb
factorial(FACT_OP,out_tmp);
assign out = a + b + out_tmp;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Compilation Unit User-defined Datatype Declaration

```
//$unit begins
typedef struct packed {
int a;
int b;} my_struct;
//End of $unit
module test (p,q,r);
input my_struct p;
input my_struct q;
output int r;
assign r = p.a * q.b;
endmodule
```

Back

Example - Compilation Unit Variable Declaration

```
//$unit begins
logic foo_logic = 1'b1;
//$unit ends
module test (
input logic data1,
input clk,
output logic out1 );

always @(posedge clk)
begin
   out1 <= data1 ^ foo_logic;
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Multi-dimensional Packed Array with Whole Assignment

```
module test (
input [1:0] [1:0] sel,
input [1:0] [1:0] data1,
input [1:0] [1:0] data2,
input [1:0] [1:0] data3,
output reg [1:0] [1:0] out1,
output reg [1:0] [1:0] out2,
output reg [1:0] [1:0] out3 );

always @(sel,data1,data2)
   begin
      out1 = (sel[1]==11)? data1 : {11,11};
      out2 = (sel[1]==2'b11)? data2 : {11,10};
      out3 = data3;
   end
endmodule
```

Back

Example - Multi-dimensional Packed Array with Partial Assignment

```
module test (
input [7:0] datain,
input [1:0][2:0][3:0] datain2,
output [1:0][1:0][1:0] array_out,
output [23:0] array_out2,
output [3:0] array_out2_first_element,
   array_out2_second_element, array_out2_zero_element,
output [1:0] array_out2_first_element_partial_slice );
assign array_out = datain;
assign array_out2 = datain2;
assign array_out2_zero_element = datain2[1][0];
assign array_out2_first_element = datain2[1][1];
assign array_out2_second_element = datain2[1][2];
assign array_out2_first_element_partial_slice =
   datain2[0][0][3-:2];
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Multi-dimensional Packed Array with Arithmetic Ops

```
module test (
input signed [1:0][2:0] a, b,
output signed [1:0] [2:0] c, c_bar, c_mult, c_div, c_per,
output signed [1:0][2:0] d, d_bar, d_mult, d_div, d_per,
output signed e, e_bar, e_mult, e_div, e_per );
assign c = a + b;
assign d = a[1] + b[1];
assign e = a[1][2] + a[1][1] + a[1][0] + b[1][2] + b[1][1]
     + b[1][0];
assign c_bar = a - b;
assign d_bar = a[1] - b[1];
assign e_bar = a[1][2] - a[1][1] - a[1][0] - b[1][2]
     - b[1][1] - b[1][0];
assign c_mult = a * b;
assign d_mult = a[1] * b[1];
assign e_mult = a[1][2] * a[1][1] * a[1][0] * b[1][2] *
     b[1][1] * b[1][0];
assign c_div = a / b;
assign d_div = a[1] / b[1];
assign e_div = a[1][2] / b[1][1];
assign c_per = a % b;
assign d_per = a[1] % b[1];
assign e_per = a[1][2] % b[1][1];
endmodule
```

Back

Example - Packed/Unpacked Array with Partial Assignment

```
module test (
input [1:0] sel [1:0],
input [63:0] data [3:0],
input [63:0] data2 [3:0],
output reg [15:0] out1 [3:0],
output reg [15:0] out2 [3:0]);

always @(sel, data)
   begin
      out1 = (sel[1]==2'b00)? data[3][63-:16] :
         ((sel[1]==2'b01)? data[2][47-:16] :
         ((sel[0]==2'b10)? data[1][(63-32)-:16] :
            data[0][(63-48)-:16] ) );
      out2[3][15-:16] = data2[3][63-:16];
      out2[2][15-:16] = data2[3][47-:16];
      out2[1][15-:16] = data2[3][(63-32)-:16];
      out2[0][15-:8] = data2[3][(63-48)-:8];
   end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Multi-dimensional Array of Packed Structures Using Anonymous Type

```
module mda_str (
input struct packed {
logic [47:0] dest_addr;
logic [47:0] src_addr;
logic [7:0] type_len;
logic [63:0] data;
logic [2:0] crc;
} [1:0][3:0] str_pkt_in,
input sel1,
input [1:0] sel2,
output struct packed {
logic [47:0] dest_addr;
logic [47:0] src_addr;
logic [7:0] type_len;
logic [63:0] data;
logic [2:0] crc;
} str_pkt_out
);
always_comb
begin
str_pkt_out = str_pkt_in[sel1][sel2];
end
endmodule
```

Back

Example - Multi-dimensional Array of Packed and Unpacked Structures Using typedef

```
typedef struct {
byte r;
byte g;
byte b;
} [2:0] struct_im_t [0:1];
module mda_str (
input struct_im_t a,
input struct_im_t b,
output struct_im_t c,
input [7:0] alpha,
input [7:0] beta
);
typedef struct {
shortint r;
shortint g;
shortint b;
} [2:0] struct_im_r_t [0:1];
struct_im_r_t temp;
integer i,j;

always_comb
begin
   for(i=0;i<2;i=i+1)
   for(j=0;j<3;j=j+1)
   begin
      temp[i][j].r = a[i][j].r * alpha + b[i][j].r * beta;
      temp[i][j].g = a[i][j].g * alpha + b[i][j].g * beta;
      temp[i][j].b = a[i][j].b * alpha + b[i][j].b * beta;
      c[i][j].r = temp[i][j].r[15:8];
      c[i][j].g = temp[i][j].g[15:8];
      c[i][j].b = temp[i][j].b[15:8];
   end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Multi-dimensional Array of Unpacked Structures Using typedef

```
typedef struct {
byte r;
byte g;
byte b;
} struct_im_t [2:0][1:0];
module mda_str (
input struct_im_t a,
input struct_im_t b,
output struct_im_t c,
input [7:0] alpha,
input [7:0] beta
);
typedef struct {
shortint r;
shortint g;
shortint b;
} struct_im_r_t [2:0][1:0];
struct_im_r_t temp;
integer i,j;

always_comb
begin
   for(i=0;i<3;i=i+1)
   for(j=0;j<2;j=j+1)
   begin
      temp[i][j].r = a[i][j].r * alpha + b[i][j].r * beta;
      temp[i][j].g = a[i][j].g * alpha + b[i][j].g * beta;
      temp[i][j].b = a[i][j].b * alpha + b[i][j].b * beta;
      c[i][j].r = temp[i][j].r[15:8];
      c[i][j].g = temp[i][j].g[15:8];
      c[i][j].b = temp[i][j].b[15:8];
   end
end
endmodule
```

Back

Example - Multi-dimensional Array of Packed Structures Using typedef

```
typedef struct packed {
logic [47:0] dest_addr;
logic [47:0] src_addr;
logic [7:0] type_len;
logic [63:0] data;
logic [3:0] crc;
} [1:0][1:0] str_pkt_mp_t;
typedef struct packed {
logic [47:0] dest_addr;
logic [47:0] src_addr;
logic [7:0] type_len;
logic [63:0] data;
logic [3:0] crc;
} str_pkt_t;
module mda_str (
input str_pkt_mp_t pkt_mp_in,
input sel1,
input sel2,
output str_pkt_t pkt_out
);
always_comb
begin
   pkt_out = pkt_mp_in[sel1][sel2];
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Array Querying Function with Data Type as Input

```
//Data type
typedef bit [1:2][4:1]bit_dt[3:2][4:1];
module top
(
//Output
output byte q1_left,
output byte q1_low );
assign q1_left = $left(bit_dt);
assign q1_low = $low(bit_dt);
endmodule
```

Back

Example - Array Querying Function $dimensions and $unpacked_dimensions Used on a Mixed Array

```
module top
(
//Input
input bit [1:2][4:1]d1[3:2][4:1],
//Output
output byte q1_dimensions,
output byte q1_unpacked_dimensions );
assign q1_dimensions = $dimensions(d1);
assign q1_unpacked_dimensions = $unpacked_dimensions(d1);
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Array Querying Function $left and $right Used on Packed 2D-data Type

```
module top
(
//Input
input logic[1:0][3:1]d1,
//Output
output byte q1_left,
output byte q1_right,
output byte q1_leftdimension
);
assign q1_left = $left(d1);
assign q1_right = $right(d1);
assign q1_leftdimension =$left(d1,2); // Dimension expression
  // returns value of the second dimension[3:1]
endmodule
```

Back

Example - Array Querying Function $low and $high Used on Unacked 3D-data Type

```
module top
(
//Input
input logic d1[2:1][1:5][4:8],
//Output
output byte q1_low,
output byte q1_high,
output byte q1_lowdimension
);
assign q1_low = $low(d1);
assign q1_high = $high(d1);
assign q1_lowdimension = $low(d1,3); // Dimension expression
  // returns value for the third dimension (i.e.,[4:8])
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Array Querying Function $size and $increment Used on a Mixed Array

```
module top
(
//Input
input byte d1[4:1],
//Output
output byte q1_size,
output byte q1_increment );
assign q1_size = $size(d1);
assign q1_increment = $increment(d1);
endmodule
```

Back

Example - Instantiating an interface Construct

```
 //TECHPUBS The following example defines, accesses, and
instantiates an interface construct.
 interface intf(input a, input b); //define the interface
 logic a1, b1;
 assign a1 = a;
 assign b1 = b;
 modport write (input a1, input b1); //define the modport
 endinterface

 module leaf(intf.write foo, output logic q); //access the intf
interface
 assign q = foo.a1 + foo.b1;
 endmodule

 module top(input a, input b, output q);
 intf inst_intf (a,b); //instantiate the intf interface
 leaf leaf_inst (inst_intf.write,q);
 endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Type Casting of Aggregate Data Types

```
//Data type
typedef logic Logic_3D_dt [15:0][1:0][1:0];
typedef logic Logic_1D_dt [64:1];
module top (
//Inputs
input Logic_3D_dt Logic_3D,
//Outputs
output longint arith
);
//Constant delcaration
const Logic_1D_dt Logic_1DConst = ‘{default:1’b1};
//Arithmetic Operation
assign arith = longint’(Logic_3D) + longint’(Logic_1DConst);
endmodule
```

Back

Example - Bit-stream Casting

```
typedef struct {
bit start_bit = 0;
byte data_bits;
bit stop_bit = 1; }
uart_format_dt;
typedef logic tx_format_dt[9:0] ;
module top (
//Inputs
input byte data,
//Outputs
output tx_format_dt tx_data
);
uart_format_dt uart_data;
assign uart_data.data_bits = data;
assign tx_data = tx_format_dt'(uart_data);
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Sign Casting

```
module top (
//Inputs
input integer Integer,
input shortint Shortint,
//Outputs
output longint arith
);
//Arithmetic operation
assign arith = unsigned'(Integer) * unsigned'(Shortint);
endmodule
```

Back

Example - Size Casting

```
module top (
//Inputs
input longint Longint,
input byte Byte,
//Outputs
output shortint arith1
);
//Arithmetic operation
assign arith1 = 10'(Byte + Longint);
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Type Casting of Singular Data Types

```
typedef logic [31:0] unsigned_32bits;
typedef logic [15:0] unsigned_16bits;
module top (
//Inputs
input integer Integer,
input shortint Shortint,
//Outputs
output longint arith
);
//Arithmetic operation
assign arith = unsigned_32bits'(Integer) *
unsigned_16bits'(Shortint) ;
endmodule
```

Back

Example - Basic Packed Union (arithmetic operation)

```
typedef union packed
{
  logic [3:0][0:3]u1;
  shortint u2;
  bit signed [1:2][8:1]u3;
}union_dt; // Union data type

module top
  (input union_dt d1,
   input union_dt d2,
   output union_dt q1,
   output union_dt q2
  );
assign q1.u2 = d1.u1 + d2.u2;
assign q2.u1 = d1.u2 - d1.u1[2][1];
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Array of Packed Union

```
typedef int int_dt;
typedef union packed
{
  int_dt u1;
  bit [0:3][1:8]u2;
}union_dt;
module top
  (input union_dt [1:0] d1, //Array of union
   input union_dt [1:0] d2, //Array of union
   output union_dt q1,
   output union_dt q2
  );
assign q1.u1 = d1[1].u1 ^ d2[0].u1;
assign q2.u2 = ~(d1[0].u2 | d2[1].u1);
endmodule
```

Back

Example - Basic Packed Union (logical operation)

```
typedef int unsigned UnsignInt_dt;
typedef union packed
{
  int u1;
  UnsignInt_dt u2;
}union_dt; //Union data type

module top
  (input union_dt d1,
input union_dt d2,
output union_dt q1,
output union_dt q2
);
assign q1.u1 = d1.u1 ^ d2.u1;
assign q2.u2 = d1.u2 | d2.u1;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Nested Packed Union

```
typedef union packed
{
  byte u1;
  bit[1:0][4:1]u2;
  union packed
  {
     logic[8:1]nu1;
     byte unsigned nu2;
  }NstUnion; //Nested Union
}NstUnion_dt;

module top
  (input NstUnion_dt d1,
   input NstUnion_dt d2,
   output NstUnion_dt q1,
   output NstUnion_dt q2
  );
assign q1 = d1.NstUnion.nu1 & d2.u2[1];
assign q2.u1 = d2.NstUnion.nu2 |~ d1.u1;
endmodule
```

Back

Example - State-machine Design

```
module enum_type_check (clk, rst, same, statemachine1_is_five,
   statemachine2_is_six, statemachine1, statemachine2, both);
input clk, rst;
output reg same, statemachine1_is_five, statemachine2_is_six;
output int statemachine1, statemachine2, both;
enum {a[0:3] = 4} my,my1;

always@(posedge clk or posedge rst)
begin
if (rst)
   begin
      my <= a0;
   end
else
   case(my)
      a0 :begin
         my <= a1;
      end
      a1 :begin
         my <= a2;
      end
      a2 :begin
         my <= a3;
      end
      a3 :begin
         my <= a0;
      end
   endcase
end

always@(posedge clk or posedge rst)
begin
if (rst)
   begin
      my1 <= a0;
   end
else
   case(my1)
      a0 :begin
         my1 <= a3;
```


Copyright © 2013 Synopsys, Inc.

```
            end
            a1 :begin
               my1 <= a0;
            end
            a2 :begin
               my1 <= a1;
            end
            a3 :begin
               my1 <= a2;
            end
         endcase
      end

      always@(posedge clk)
      begin
      statemachine1 <= my;
      statemachine2 <= my1;
      both <= my + my1;
      if (my == my1)
         same <= 1'b1;
      else
         same <= 0;
      if (my == 5)
         statemachine1_is_five <= 1'b1;
      else
         statemachine1_is_five <= 1'b0;
      if (my1 == 6)
         statemachine2_is_six <= 1'b1;
      else
         statemachine2_is_six <= 1'b0;
      end
      endmodule
```

Back

Example – Type Parameter of Language-Defined Data Type

```
//Compilation Unit
module top
#(
  parameter type PTYPE = shortint,
  parameter type PTYPE1 = logic[3:2][4:1] //parameter is of
     //2D logic type
)
(
//Input Ports
  input PTYPE din1_def,
  input PTYPE1 din1_oride,
//Output Ports
  output PTYPE dout1_def,
  output PTYPE1 dout1_oride
);
sub u1_def //Default data type
(
  .din1(din1_def),
  .dout1(dout1_def)
);
sub #
(
  .PTYPE(PTYPE1) //Parameter type is override by 2D Logic
)
u2_oride
(
  .din1(din1_oride),
  .dout1(dout1_oride)
);
endmodule
//Sub Module
module sub
#(
  parameter type PTYPE = shortint //parameter is of shortint type
)
(
//Input Ports
  input PTYPE din1,
//Output Ports
  output PTYPE dout1
```


Copyright © 2013 Synopsys, Inc.

```
);
always_comb
begin
  dout1 = din1 ;
end
endmodule
```

Back

Example – Type Local Parameter

```
//Compilation Unit
module sub
#(
parameter type PTYPE1 = shortint, //Parameter is of shortint type
parameter type PTYPE2 = longint //Parameter is of longint type
)
(
//Input Ports
  input PTYPE1 din1,
//Output Ports
  output PTYPE2 dout1
);
//Localparam type definitation
localparam type SHORTINT_LPARAM = PTYPE1;
SHORTINT_LPARAM sig1;
assign sig1 = din1;
assign dout1 = din1 * sig1;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Type Parameter of User-Defined Data Type

```
//Compilation Unit
typedef logic [0:7]Logic_1DUnpack[2:1];
typedef struct {
  byte R;
  int B;
  logic[0:7]G;
} Struct_dt;
module top
#(
  parameter type PTYPE = Logic_1DUnpack,
  parameter type PTYPE1 = Struct_dt
)
(
//Input Ports
  input PTYPE1    din1_oride,
//Output Ports
  output PTYPE1    dout1_oride
);
sub #
(
  .PTYPE(PTYPE1) //Parameter type is override by a structure type
)
u2_oride
(
  .din1(din1_oride),
  .dout1(dout1_oride)
);
endmodule
//Sub Module
module sub
#(
  parameter type PTYPE = Logic_1DUnpack // Parameter 1D
     // logic Unpacked data type
)
(
//Input Ports
  input PTYPE din1,
//Output Ports
  output PTYPE dout1
);
```

```
always_comb
begin
  dout1.R = din1.R;
  dout1.B = din1.B ;
  dout1.G = din1.G ;
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Parameter of Type enum

```
typedef enum {s1,s2,s3=24,s4=15,s5} enum_dt;
module sub
#(parameter enum_dt ParamEnum = s4)
   (input clk,
    input rst,
    input enum_dt d1,
    output enum_dt q1 );

always_ff@(posedge clk)
begin
if(rst)
   begin
      q1 <= ParamEnum;
   end
else
   begin
      q1 <= d1;
   end
end
endmodule
```

Back

Example – Parameter of Type longint Unpacked Array

```
module sub
#(parameter longint ParamMyLongint [0:1] ='{64'd1124,64'd1785})
   (input clk,
    input rst,
    input longint d1 [0:1],
    output longint q1 [0:1] );

always_ff@(posedge clk or posedge clk)
begin
if(rst)
   begin
      q1 <= ParamMyLongint;
   end
else
   begin
      q1 <= d1;
   end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Parameter of Type longint

```
module sub
#(parameter longint ParamLongint = 64'd25)
   (input clk,
    input rst,
    input longint d1 ,
    output longint q1 );

always_ff@(posedge clk or posedge clk)
begin
if(rst)
   begin
      q1 <= ParamLongint;
   end
else
   begin
      q1 <= d1;
   end
end
endmodule
```

Back

Example – Parameter of Type structure

```
typedef byte unsigned Byte_dt;
typedef struct packed
   {shortint R;
    logic signed [4:3] G;
    bit [15:0] B;
    Byte_dt Y;
   }Struct_dt;

module sub
#(parameter Struct_dt ParamStruct ='{16'd128,2'd2,12'd24,8'd123})
   (
   //Input
    input clk,
    input rst,
    input Struct_dt d1,
   //Output
    output Struct_dt q1 );

always_ff@(posedge clk or posedge rst)
begin
if(rst)
   begin
      q1 <= ParamStruct;
   end
else
   begin
      q1 <= d1 ;
   end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Simple typedef Variable Assignment

```
module src (in1,in2,out1,out2);
input in1,in2;
output reg out1,out2;
typedef int foo;
foo a,b;

assign a = in1; assign b = in2;
always@(a,b)
   begin
      out1 = a;
      out2 = b;
   end
endmodule
```

Back

Example - Using Multiple typedef Assignments

```
module src (in1,in2,in3,in4,out1,out2,out3);
input [3:0] in1,in2;
input in3,in4;
output reg [3:0] out1;output reg out2,out3;
typedef bit signed [3:0] foo1;
typedef byte signed foo2;
typedef int foo3;
struct {
   foo1 a;
   foo2 b;
   foo3 c;
   } foo;

always@(in1,in2,in3,in4)
   begin
      foo.a = in1 & in2;
      foo.b = in3 | in4;
      foo.c = in3 ^ in4;
   end

always@(foo.a,foo.b,foo.c)
   begin
      out1 = foo.a;
      out2 = foo.b;
      out3 = foo.c;
   end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Extern Module Instantiation

```
extern module top (input logic clock, load,
  input reset, input logic [3:0] d,
  output logic [3:0] cnt);
module top ( .* );
always @(posedge clock or posedge reset)
  begin
  if (reset)
     cnt <= 0;
  else if (load)
     cnt <= d;
  else cnt <= cnt + 1;
  end
endmodule
```

Back

Example - Extern Module Reference

```
extern module counter (clock, load, reset, d, cnt);
module top (clock, load, reset, d, cnt);
input logic clock, load;
input reset;
input logic [3:0] d;
output logic [3:0] cnt;
counter cnt1 (.clock(clock), .load(load), .reset(reset),
  .d(d), .cnt(cnt) );
endmodule
module counter (clock, load, reset, d, cnt );
input logic clock, load;
input reset;
input logic [3:0] d;
output logic [3:0] cnt;
always @(posedge clock or posedge reset)
begin
  if (reset)
     cnt <= 0;
  else if (load)
     cnt <= d;
  else cnt <= cnt + 1;
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - $bits System Function

```
module top (input logic Clk,
   input logic Rst,
   input logic [7:0] LogicIn,
   output logic [$bits(LogicIn)-1:0] LogicOut,
   output logic [7:0] LogicConstSize );
logic [7:0] logic_const = 8'd0;

always@(posedge Clk, posedge Rst) begin
   if(Rst) begin
      LogicConstSize <= `d0;
      LogicOut <= logic_const;
   end
   else begin
      LogicConstSize <= $bits(logic_const);
      LogicOut <= $bits(LogicIn)-1 ^ LogicIn;
   end
end
endmodule
```

Back

Example - $bits System Function within a Function

```
module top (input logic Clk,
   input logic Rst,
   input logic [7:0] LogicIn,
   output logic [$bits(LogicIn)-1:0] LogicOut,
   output logic [7:0] LogicSize  );
function logic [$bits(LogicIn)-1:0]
   incr_logic (logic [7:0] a);
incr_logic = a + 1;
endfunction

always@(posedge Clk, posedge Rst) begin
   if(Rst) begin
      LogicSize <= ‘d0;
      LogicOut <= ‘d0;
   end
   else begin
      LogicSize <= $bits(LogicIn);
      LogicOut <= incr_logic(LogicIn);
   end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Accessing Variables Declared in a generate-case

```
module test #(
  parameter mod_sel = 1,
  mod_sel2 = 3 )
     (input [7:0] a1,
      input [7:0] b1,
      output [7:0] c1,
      input [1:0][3:1] a2,
      input [1:0][3:1] b2,
      output [1:0][3:1] c2 );
typedef logic [7:0] my_logic1_t;
typedef logic [1:0][3:1] my_logic2_t;
generate
case(mod_sel)
  0:
  begin:u1
     my_logic1_t c1;
     assign c1 = a1 + b1;
  end
  1:
  begin:u1
     my_logic2_t c2;
     assign c2 = a2 + b2;
  end
  default:
  begin:def
     my_logic1_t c1;
     assign c1 = a1 + b1;
  end
endcase
endgenerate
generate
case(mod_sel2)
  0:
  begin:u2
     my_logic1_t c1;
     assign c1 = a1 + b1;
  end
  1:
  begin:u2
     my_logic2_t c2;
```

```
            assign c2 = a2 + b2;
         end
         default:
         begin:def2
            my_logic1_t c1;
            assign c1 = a1 * b1;
         end
      endcase
      endgenerate
      assign c2 = u1.c2;
      assign c1 = def2.c1;
      endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Shift Register Using generate-for

```
module sh_r #(
  parameter width = 8,
  pipe_num = 3 )
     (input clk,
      input[width-1:0]din,
      output[width-1:0] dout );
genvar i;
generate
  for(i=0;i<pipe_num;i=i+1)
  begin:u
  reg [width-1:0] sh_r;
     if(i==0)
     begin
        always @ (posedge clk)
           sh_r <= din;
     end
     else
     begin
        always @ (posedge clk)
           sh_r <= u[i-1].sh_r;
     end
  end
endgenerate
assign dout = u[pipe_num-1].sh_r;
endmodule
```

Back

Example Accessing Variables Declared in a generate-if

```
module test #(
  parameter width = 8,
     sel = 0 )
        (input [width-1:0] a,
         input [width-1:0] b,
         input clk,
         output [(2*width)-1:0] c,
         output bit_acc,
         output [width-3:0] prt_sel );
  genvar i;
  reg [width-1:0] t_r;
generate
  if(sel == 0)
  begin:u
     wire [width-1:0] c;
     wire [width-1:0] t;
     assign {c,t} = {~t_r,a|b};
  end
  else
  begin:u
     wire [width-1:0] c;
     wire [width-1:0] t;
     assign {c,t} = {~t_r,a^b};
  end
endgenerate
always @ (posedge clk)
begin
     t_r <= u.t;
  end
assign c = u.c;
assign bit_acc = u.t[0];
  assign prt_sel = u.t[width-1:2];
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Do-while with case Statement

```
module src (out, a, b, c, d, sel);
output [3:0] out;
input [3:0] a, b, c, d;
input [3:0] sel;
reg [3:0] out;
integer i;

always @ (a or b or c or d or sel)
   begin
   i=0;
   out = 3'b000;
      do
         begin
            case (sel)
               4'b0001: out[i] = a[i];
               4'b0010: out[i] = b[i];
               4'b0100: out[i] = c[i];
               4'b1000: out[i] = d[i];
               default: out = `bx;
            endcase
         i= i+1;
         end
      while (i < 4);
   end
endmodule
```

Back

Example - Do-while with if-else Statement

```
module src (out, a, b, c, d, sel);
output [3:0] out;
input [3:0] a, b, c, d;
input [3:0] sel;
reg [3:0] out;
integer i;

always @ (a or b or c or d or sel)
   begin
   i=0;
   out = 4'b0000;
      do
         begin
            if(sel == 4'b0001) out[i] = a[i];
            else if(sel == 4'b0010) out[i] = b[i];
            else if(sel == 4'b0100) out[i] = c[i];
            else if(sel == 4'b1000) out[i] = d[i];
            else out = `bx;
         i= i+1;
         end
      while (i < 4);
   end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Simple do-while Loop

```
module src (in1,in2,out);
input [7:0] in1,in2;
output reg [7:0] out;
integer i;

always @ (in1,in2)
begin
   i = 0;
   do
      begin
         out[i] = in1[i] + in2[i];
         i = i+1;
      end
   while (i < 8 );
end
endmodule
```

Back

Example - Simple for Loop

```
module simpleloop (output reg [7:0]y, input [7:0]i, input clock);
always@(posedge clock)
begin : loop
    for (int count=0; count < 8; count=count+1) // SV code
    y[count]=i[count];
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - For Loop with Two Variables

```
module twovarinloop (in1, in2, out1, out2);
parameter p1 = 3;
input [3:0] in1;
input [3:0] in2;
output [3:0] out1;
output [3:0] out2;
reg [3:0] out1;
reg [3:0] out2;

always @*
   begin
      for (int i = 0, int j = 0; i <= p1; i++)
      begin
         out1[i] = in1[i];
         out2[j] = in2[j];
         j++;
      end
   end
endmodule
```

Back

Example - Inside operator with array of parameter at LHS operator

```
module top
(
//Input
input byte din1,
//Output
output logic dout
);

parameter byte param1[1:0] = ‘{8’d12,8’d111};
assign dout = (din1) inside {param1,121,-16};
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Inside operator with dynamic input at LHS operator

```
module top
(
//Input
input byte din,
//Output
output logic dout
);

assign dout = din inside {8'd2, -8'd3, 8'd5};
endmodule
```

Back

Example - Inside operator with dynamic input at LHS and RHS operators

```
module top
(
//Input
input byte din1,
input byte din2,
//Output
output logic dout
);

assign dout = (din1) inside {din2,105,-121,-116};
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Inside operator with expression at LHS operator

```
module top
(
//Input
input byte din1,
input byte din2,
//Output
output logic dout
);

assign dout = (din1 | din2) inside {14,17,2,20};
endmodule
```

Back

Example - Constant Declarations

```
package my_pack;
const logic foo_logic = 1'b1;
endpackage
import my_pack::*;

module test (
input logic inp,
input clk,
output logic out );

always @(posedge clk)
begin
   out <= inp ^ foo_logic;
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Direct Reference Using Scope Resolution Operator (::)

```
package mypack;
logic foo_logic = 1'b1;
endpackage
module test (
input logic data1,
input clk,
output logic out1 );

always @(posedge clk)
begin
   out1 <= data1 ^ mypack::foo_logic;
end
endmodule
```

Back

Example - Function Declarations

```
package automatic_func;
parameter fact = 2;
function automatic [63:0] factorial;
input [31:0] n;
if (n==1)
   return (1);
else
   return (n * factorial(n-1));
endfunction
endpackage

import automatic_func::*;
module src (input [1:0] a, input [1:0] b,
   output logic [2:0] out );
always_comb
begin
   out = a + b + factorial(fact);
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Importing Specific Package Items

```
package mypack;
logic foo_logic = 1'b1;
endpackage
module test (
input logic data1,
input clk,
output logic out1 );
import mypack::foo_logic;

always @(posedge clk)
begin
   out1 <= data1 ^ foo_logic;
end
endmodule
```

Back

Example - Import Statements from Other Packages

```
package param;
parameter fact = 2;
endpackage
package automatic_func;
import param::*;
function automatic [63:0] factorial;
input [31:0] n;
   if (n==1)
      return (1);
   else
      return (n * factorial(n-1));
endfunction
endpackage

import automatic_func::*;
import param::*;
module src (input [1:0] a, input [1:0] b,
   output logic [2:0] out );
always_comb
begin
   out = a + b + factorial(fact);
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Parameter Declarations

```
package mypack;
parameter a_width = 4;
parameter b_width = 4;
localparam product_width = a_width+b_width;
endpackage
import mypack::*;

module test (
input [a_width-1:0] a,
input [b_width-1:0] b,
output [product_width-1:0] c );
assign c = a * b;
endmodule
```

Back

Example - Scope Resolution

```
 //local parameter overrides package parameter value (dout <=
data[7:0];)
 package mypack;
 parameter width = 4;
 endpackage

 import mypack::*;
 module test (data,clk,dout);
 parameter width = 8; // local parameter
 input logic [width-1:0] data;
 input clk;
 output logic [width-1:0] dout;

 always @(posedge clk)
 begin
    dout <= data;
 end
 endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Task Declarations

```
package mypack;
parameter FACT_OP = 2;
   task automatic factorial(input integer operand,
      output [1:0] out1);
   integer nFuncCall = 0;
   begin
   if (operand == 0)
      begin
         out1 = 1;
      end
   else
      begin
         nFuncCall++;
         factorial((operand-1), out1);
         out1 = out1 * operand;
      end
   end
   endtask
endpackage
import mypack::*;

module src (input [1:0] a, input [1:0] b,
   output logic [2:0] out );
logic [1:0] out_tmp;

always_comb
factorial(FACT_OP,out_tmp);
assign out = a + b + out_tmp;
endmodule
```

Back

Example - User-defined Data Types (typedef)

```
package mypack;
typedef struct packed {
   int a;
   } my_struct;
endpackage
import mypack::my_struct;

module test (inp1,inp2,out);
input my_struct inp1;
input my_struct inp2;
output int out;
assign out = inp1.a * inp2.a;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Wildcard (*) Import Package Items

```
package mypack;
logic foo_logic = 1'b1;
endpackage
module test (
input logic data1,
input clk,
output logic out1 );
import mypack::*;

always @(posedge clk)
begin
   out1 <= data1 ^ foo_logic;
end
endmodule
```

Back

Example – Packed type inputs/outputs with LHS operator

```
module streaming
(
input byte a,
output byte str_rev,
output byte str
);

assign {>>{str}} = a;
assign {<<{str_rev}} = a;

endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Packed type inputs/outputs with RHS operator

```
module streaming
(
input longint a,
output longint str_rev,
output longint str
);

assign str_rev = {<< {a}};
assign str = {>> {a}};

endmodule
```

Back

Example – Slice-size streaming with LHS slice operation

```
module streaming
(
input logic a[1:8],
output logic signed [1:4] str_rev[1:2],
output logic signed [1:4] str[1:2]
);

assign {>>4{str}} = a;
assign {<<4{str_rev}} = a;

endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example – Slice-size streaming with RHS operator

```
typedef shortint shortint_dt [2:1];
typedef byte byte_dt [1:2][3:2];
typedef struct {
  logic [3:0] a [2:1];
  byte b;
  shortint c[4:2]; }
struct_dt;
module streaming (
  input shortint_dt a,
  input  byte_dt b,
  output struct_dt c_pack,
  output struct_dt c_unpack );
assign c_pack = {<< 5 {a}};
assign c_unpack = {<< 2 {b}};
endmodule
```

Back

Example – Unpacked type inputs/outputs with RHS operator

```
typedef logic [5:0]my_dt [1:0];

module streaming
(
input logic [5:0] a[1:0], //same layout - size same as the output
input logic [3:0] b[2:0], //different layout - same size as output
input logic [2:0]c[1:0], //different layout and size
output my_dt str_rev1,
output my_dt str_rev_difflay,
output my_dt str_rev_less
);

assign str_rev1 = {<<{a}};
assign str_rev_difflay = {<< {b}};
assign str_rev_less = {<< {c,2'b11}};

endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Priority case

```
module src (out, a, b, c, d, sel);
output out;
input a, b, c, d;
input [3:0] sel;reg out;

always @ (a,b,c,d,sel)
   begin
      priority case (sel)
         4'b0000: out = c;
         4'b0001: out = b;
         4'b0100: out = d;
         4'b1000: out = a;
      endcase
   end
endmodule
```

Back

Example - Unique case

```
module src (out, a, b, c, d, sel);
output out;
input a, b, c, d;
input [3:0] sel;
reg out;

always @ (a,b,c,d,sel)
   begin
      unique case (sel)
         4'b0001: out = c;
         4'b0010: out = b;
         4'b0100: out = d;
         4'b1000: out = a;
      endcase
   end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

APPENDIX H

VHDL Example Code

This appendix contains the code samples that are referenced by the corresponding chapter.

Example - Direct Instantiation Using Configuration Declaration

```
--Entity to be instantiated using the configuration
library ieee;
use ieee.std_logic_1164.all;
entity module0 is
  generic (SIZE : integer := 10);
  port (l : in std_logic_vector(SIZE-1 downto 0);
     m : in std_logic_vector(SIZE-1 downto 0);
     out1 : out std_logic_vector(SIZE-1 downto 0) );
end entity module0;
architecture behv of module0 is
begin
  out1 <= l xor m;
end behv;
-- Configuration for the entity module0
configuration conf_sub of module0 is
  for behv
  end for;
end conf_sub;
-- Module in which the entity module0 is instantiated
-- using the configuration
library ieee;
use ieee.std_logic_1164.all;
entity top is
```


Copyright © 2013 Synopsys, Inc.

```
  port (in0 : in std_logic_vector(31 downto 0);
     in1 : in std_logic_vector(31 downto 0);
     out1 : out std_logic_vector(31 downto 0) );
end entity top;
architecture behv of top is
begin
U0: configuration conf_sub
  generic map (SIZE => 32)
  port map (l => in0,
     m => in1,
     out1 => out1 );
end behv;
```

Back

SYNOPSYS®
Predictable Success

APPENDIX I

VHDL2008 Example Code

This appendix contains the code samples that are referenced by the corresponding chapter.

Copyright © 2013 Synopsys, Inc.

Example - Context declaration

```
context zcontext is
  library ieee;
  use ieee.std_logic_1164.all;
end context zcontext;

context work.zcontext;
use ieee.numeric_std.all;

entity myTopDesign is
  port (in1: in std_logic_vector(1 downto 0);
     out1: out std_logic_vector(1 downto 0) );
end myTopDesign;

architecture myarch2 of myTopDesign is
begin
  out1 <= in1;
end myarch2;
```

Back

Example - Unconstrained element types

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
package myTypes is
  type memUnc is array (natural range <>) of std_logic_vector;
  function summation(varx: memUnc) return std_logic_vector;
end package myTypes;
package body myTypes is
  function summation(varx: memUnc) return std_logic_vector is
     variable sum: varx'element;
  begin
     sum := (others => '0');
        for I in 0 to varx'length - 1 loop
           sum := sum + varx(I);
        end loop;
     return sum;
  end function summation;
end package body myTypes;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use work.myTypes.all;
entity sum is
  port (in1: memUnc(0 to 2)(3 downto 0);
        out1: out std_logic_vector(3 downto 0) );
end sum;
architecture uncbehv of sum is
begin
  out1 <= summation(in1);
end uncbehv;
```

Back

Copyright © 2013 Synopsys, Inc.

Example - all keyword

```
entity mycomp is
  port (a, c: in bit; b: out bit);
end mycomp;

architecture myarch of mycomp is
begin
  process (all)
  begin
     b <= not a or c;
  end process;
end myarch;
```

Back

Example 1: VHDL 2008 Style Conditional Operator

```
entity condOpTest is
port (
sel, in1, in2: in bit;
res: out bit
);
end condOpTest;
architecture rtlArch of condOpTest is
begin
process(in1,in2,sel)
begin
if sel then
res <= in2;
else
res <= in1;
end if;
end process;
end rtlArch;
```

Back

Copyright © 2013 Synopsys, Inc.

Example 2: VHDL 1993 Style Conditional Operator

```
entity condOpTest is
port (
sel, in1, in2: in bit;
res: out bit
);
end condOpTest;
architecture rtlArch of condOpTest is
begin
process(in1,in2,sel)
begin
if sel = '1' then
res <= in2;
else
res <= in1;
end if;
end process;
end rtlArch;
```

Back

Example 1: Logical Operators

```
entity reductionOpTest is
port (
invec: in bit_vector(2 downto 0);
nandout, xorout, xnorout, norout, orout, andout: out bit
);
end reductionOpTest;

architecture rtlArch of reductionOpTest is
begin
nandout <= nand invec;
xorout <= xor invec;
xnorout <= xnor invec;
norout <= nor invec;
orout <= or invec;
andout <= and invec;
end rtlArch;
```

Back

Copyright © 2013 Synopsys, Inc.

Example: Relational Operator

```
entity relOpTest is
port (
in1, in2: in bit;
res_eq, res_lteq: out bit
);
end relOpTest;
architecture rtlArch of relOpTest is
begin
res_eq <= in1 ?= in2;
res_lteq <= in1 ?<= in2;
end rtlArch;
```

Back

Example - Including generics in packages

```
-- Generic Package Declaration
package myTypesGeneric is
  generic
  (width: integer := 7; testVal: bit_vector(3 downto 0) := "0011";
   dfltVal: bit_vector(3 downto 0) := "1110"
  );
  subtype nvector is bit_vector(width-1 downto 0);
  constant resetVal: bit_vector(3 downto 0) := dfltVal;
  constant myVal: bit_vector(3 downto 0) := testVal;
end package myTypesGeneric;

-- Package instantiation
package myTypes is new work.myTypesGeneric
  generic map
  (width => 4, dfltVal => "0110"
  );

library IEEE;
package my_fixed_pkg is new IEEE.fixed_generic_pkg
  generic map
  (fixed_round_style    => IEEE.fixed_float_types.fixed_round,
   fixed_overflow_style => IEEE.fixed_float_types.fixed_saturate,
   fixed_guard_bits     => 3,
   no_warning           => false
  );

use work.myTypes.all;
use work.my_fixed_pkg.all;

entity myTopDesign is
  port  (in1: in nvector; out1: out nvector;
     insf: in sfixed(3 downto 0);
     outsf: out sfixed(3 downto 0);
     out2, out3, out4: out bit_vector(3 downto 0)
     );
end myTopDesign;

architecture myarch2 of myTopDesign is
begin
  out1 <= in1;
```


Copyright © 2013 Synopsys, Inc.

```
    out2 <= resetVal;
    out3 <= myVal;
    outsf <= insf;
  end myarch2;
```

Back

Example - Else/elsif clauses in if-generate statements

```
entity myTopDesign is
  generic (genval: bit_vector(1 downto 0) := "01");
  port (in1, in2, in3: in bit; out1: out bit);
end myTopDesign;

architecture myarch2 of myTopDesign is

component mycomp
  port (a: in bit;
     b: out bit );
end component;

begin
a1:
  if spec1: genval="10" generate
     inst1: mycomp port map (in1,out1);
  elsif spec2: genval="11" generate
     inst1: component mycomp port map (in2,out1);
  else spec3: generate
     inst1: component mycomp port map (in3,out1);
  end generate;
end myarch2;


library ieee;
use ieee.std_logic_1164.all;

entity mycomp is
  port ( a: in bit;
         b : out bit);
end entity mycomp;

architecture myarch1 of mycomp is
begin
  b <= '1' xor a;
end myarch1;

architecture myarch2 of mycomp is
begin
```


Copyright © 2013 Synopsys, Inc.

```
  b <= '1' xnor a;
end myarch2;

architecture myarch3 of mycomp is
signal temp : bit := '1';
begin
  b <= temp xor not(a);
end myarch3;

configuration myconfig of myTopDesign is
  for myarch2
     for a1 (spec1)
        for inst1: mycomp
           use entity mycomp(myarch1);
        end for;
     end for;
     for a1 (spec2)
        for inst1: mycomp
           use entity mycomp(myarch2);
        end for;
     end for;
     for a1 (spec3)
        for inst1: mycomp
           use entity mycomp(myarch3);
        end for;
     end for;
  end for;
end configuration myconfig;
```

Back

Example - Use of case? statement

```
library ieee;
use ieee.std_logic_1164.all;
entity myTopDesign is
  port (in1, in2: in bit;
        sel: in std_logic_vector(2 downto 0);
        out1: out bit );
end myTopDesign;
architecture myarch2 of myTopDesign is
begin
  process(all)
  begin
     a1: case? sel is
        when "1--" =>
           out1 <= in1;
        when "01-" =>
           out1 <= in2;
        when others =>
           out1 <= in1 xor in2;
     end case?;
  end process;
end myarch2;
```

Back

Copyright © 2013 Synopsys, Inc.

Example - Use of select? Statement

```
library ieee;
use ieee.std_logic_1164.all;
entity myTopDesign is
  port (in1, in2: in bit;
        sel: in std_logic_vector(2 downto 0);
        out1: out bit );
end myTopDesign;
architecture myarch2 of myTopDesign is
begin
  with sel select?
     out1 <=
     in1 when "1--",
     in2 when "01-",
     in1 xor in2 when others;
end myarch2;
```

Back

Example - extended character set

```
library ieee;
use ieee.std_logic_1164.all;
entity get_version is
  port ( ver : out string(16 downto 1));
end get_version;
architecture behv of get_version is
constant version : string (16 downto 1) := "version ©«ãëïõü»";
-- Above string includes extended ASCII characters that
-- fall between 127-255
begin
  ver <= version;
end behv;
```

Back

Copyright © 2013 Synopsys, Inc.

Copyright © 2013 Synopsys, Inc.

SYNOPSYS®
Predictable Success

APPENDIX J

Attributes Example Code

This appendix contains the code samples that are referenced by the corresponding chapter.

Copyright © 2013 Synopsys, Inc.

Example -- Verilog compiled into default library

```
//Entry in .cdc file:
// define_directive {v:sub} {syn_black_box} {1}
module top (
input clock,
input reset,
input din,
input din1,
output dout );
sub UUT (clock, reset, din, din1, dout);
endmodule
module sub (
input clock,
input reset,
input din,
input din1,
output reg dout );
always@(posedge clock)
begin
if (reset == 1'b1)
dout = 0;
else
dout = din | din1;
end
endmodule
```

Back

Example -- Verilog compiled into defined library

```
//Entry in .cdc file (compiles submodule into MyLib):
// define_directive {v:MyLib.sub} {syn_black_box} {1}
//top.v
module top (
input a,
input b,
output c,
output d );
sub inst1 (.a(a), .b(b), .c(c), .d(d) );
endmodule
//sub.v
module sub (
input a,
input b,
output c,
output d );
assign c = a & b;
assign d = top.a;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example -- VHDL compiled into defined library

```
--Entry in .cdc file (compiles submodule into MyLib):
-- define_directive {v:MyLib.sub(RTL_1)} {syn_black_box} {1}
--Top.vhd
library ieee;
use ieee.std_logic_1164.all;
library MyLib;
use MyLib.all;
entity top is
port (clk : in std_logic;
din : in std_logic;
dout : out std_logic );
end top;
architecture RTL of top is
signal inter : std_logic;
component sub
port (clk : in std_logic;
din : in std_logic;
dout : out std_logic );
end component;
for UUT1 : sub
use entity work.sub(RTL_1)
port map ( clk => clk, din => din, dout => dout);
for UUT2 : sub
use entity work.sub(RTL_2)
port map ( clk => clk, din => din, dout => dout);
begin
UUT1 : entity MyLib.sub(RTL_1)
port map (
clk => clk,
din => din,
dout => inter );
UUT2 : sub
port map (
clk => clk,
din => inter,
dout => dout );
end RTL;
--sub.vhd
library ieee;
use ieee.std_logic_1164.all;
```

Copyright © 2013 Synopsys, Inc.

```
entity sub is
port (
clk : in std_logic;
din : in std_logic;
dout : out std_logic );
end sub;
architecture RTL_1 of sub is
begin
process (clk)
begin
if rising_edge(clk) then
dout <= din;
end if;
end process;
end RTL_1;
architecture RTL_2 of sub is
begin
process (clk)
begin
if rising_edge(clk) then
dout <= not din;
end if;
end process;
end RTL_2;
```

Back

Copyright © 2013 Synopsys, Inc.

Example 2: VHDL syn_direct_reset example

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
  port (
     a : in std_logic;
     b : in std_logic;
     c : in std_logic;
     clk : in std_logic;
     din : in std_logic_vector(1 downto 0);
     dout : out std_logic_vector(1 downto 0)
);
attribute syn_direct_reset : boolean;
attribute syn_direct_reset of a : signal is true;
end entity test;
architecture beh of test is
signal rst : std_logic;
begin
rst <= a or b or c;
process(clk, rst)
begin
  if (clk='1' and clk'event) then
     if (rst = '1') then
     dout <= (others => '0');
     else
     dout <= din;
     end if;
  end if;
end process;
end beh;
```

Back

Example 1: Verilog syn_direct_reset example

```
module test (
input a /* synthesis syn_direct_reset = 1 */,
input b,
input c,
input clk,
input [1:0] din,
output reg [1:0] dout
);
always @ (posedge clk)
  if (a == 1'b1 || b == 1'b1 || c == 1'b1)
     dout = 2'b00;
  else
     dout = din;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Example 2: VHDL syn_direct_set example

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port (
  a : in std_logic;
  b : in std_logic;
  c : in std_logic;
  clk : in std_logic;
  din : in std_logic_vector(1 downto 0);
  dout : out std_logic_vector(1 downto 0)
  );
attribute syn_direct_set : boolean;
attribute syn_direct_set of a : signal is true;
end entity test;
architecture beh of test is
signal set : std_logic;
begin
set <= a or b or c;
process(clk, set)
begin
  if (clk='1' and clk'event) then
     if (set = '1') then
     dout <= (others => '1');
     else
     dout <= din;
  end if;
end if;
end process;
end beh;
```

Back

Example 1: Verilog syn_direct_set example

```
module test (
input a /* synthesis syn_direct_set = 1 */,
input b,
input c,
input clk,
input [1:0] din,
output reg [1:0] dout
);
always @ (posedge clk)
  if (a == 1'b1 || b == 1'b1 || c == 1'b1)
     dout = 2'b11;
  else
     dout = din;
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

Defining BUFR in Entity Declaration

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity bufr_ff_vhd is
port (
  clk : in std_logic;
  clk1 : in std_logic ;
  en : in std_logic;
  rst : in std_logic;
  ip1  : in std_logic;
  out1 : out std_logic_vector(1 downto 0));

attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk : signal is “BUFR”;
end entity bufr_ff_vhd;

architecture bufr_arch of bufr_ff_vhd is
signal clk1_en : std_logic;

attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk1_en : signal is “BUFR”;

begin

clk1_en <= clk1  and en;

ff_reg :process (clk , rst)
  begin
     if(rst = ‘1’) then
        out1(0) <= ‘0’;
        elsif rising_edge(clk) then
        out1(0) <=  ip1;
     end if;
  end process ff_reg;

ff_reg1 :process (clk1_en , rst)
  begin
     if(rst = ‘1’) then
```

```
            out1(1) <= '0';
            elsif rising_edge(clk1_en) then
            out1(1) <=  ip1;
         end if;
      end process ff_reg1;

   end architecture bufr_arch;
```

Back

Copyright © 2013 Synopsys, Inc.

Example – BUFR applied to net (Verilog)

```
module bufr_basic (
input rst,
input [14:0] in1,
input [14:0] in2,
output reg [29:0] out2,
input clk1 ,
input clk2/* synthesis syn_insert_buffer = "BUFR" */ );

reg [14:0] reg_in2;
reg [14:0] reg_in1;
reg [29:0] temp;

always @(posedge clk1)
begin
  if(rst)
     begin
        reg_in1 <= 0;
        reg_in2 <= 0;
        temp <= 0;
     end
  else
     begin
        reg_in1 <= in1 ;
        reg_in2 <= in2 ;
        temp <= reg_in1 + reg_in2 ;
     end
end

always @(posedge clk2)
begin
  if(rst)
     out2 <= 0;
  else
     out2 <= temp ;
end
endmodule
```

Back

Example – BUFR applied to net (VHDL)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity bufr_ff_vhd is
port (
  clk : in std_logic;
  clk1 : in std_logic ;
  en : in std_logic;
  rst : in std_logic;
  ip1  : in std_logic;
  out1 : out std_logic_vector(1 downto 0));
end entity bufr_ff_vhd;

architecture bufr_arch of bufr_ff_vhd is
signal clk1_en : std_logic;

attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk1_en : signal is "BUFR";

begin
clk1_en <= clk1  and en;

ff_reg :process (clk , rst)
  begin
  if(rst = '1') then
     out1(0) <= '0';
     elsif rising_edge(clk) then
     out1(0) <=  ip1;
  end if;
end process ff_reg;

ff_reg1 :process (clk1_en , rst)
  begin
     if(rst = '1') then
        out1(1) <= '0';
        elsif rising_edge(clk1_en) then
        out1(1) <=  ip1;
  end if;
```


Copyright © 2013 Synopsys, Inc.

```
end process ff_reg1;

end architecture bufr_arch;
```

Back

Example – BUFGMUX Inferencing (Verilog)

```
// Attribute attached to clk_mux.
`define SIZE 16

module cr_top (
input rst,
input [`SIZE:0] in1,in2,
output reg [`SIZE:0]  out,
input clk1,
input clk2,
input sel_clk
);

wire  clk_mux/*synthesis syn_insert_buffer="BUFGMUX"*/;

assign clk_mux = sel_clk ? clk1 : clk2;

always @(posedge clk_mux or posedge rst)
begin
  if(rst)
  begin
     out <= 0;
  end
  else

  begin
     out <= in1&in2 ;
  end
end
endmodule
```

Back

Copyright © 2013 Synopsys, Inc.

- BUFGMUX inferencing (VHDL)

```
-- Attribute attached to clk_out.
library ieee;
use ieee.std_logic_1164.all;

entity Top is
port( clk1: in std_logic;
  clk2: in std_logic;
  sel_clk: in std_logic;
  in1,in2: in std_logic_vector (2 downto 0);
  outp: out std_logic_vector (2 downto 0) ); end Top;

architecture behav of Top is
signal clk_out: std_logic;
attribute syn_insert_buffer : string ;
attribute syn_insert_buffer of clk_out : signal is "BUFGMUX";

begin
   process1: process(sel_clk,clk1,clk2)
   begin
     case sel_clk is
      when '0' => clk_out <= clk1;
      when '1' => clk_out <= clk2;
     end case;

   end process process1;
   process2: process(clk_out)
   begin
     if rising_edge (clk_out) then
      outp <= in1 and in2;
     end if;
   end process process2;

end behav;
```

Back

Example – syn_insert_buffer Attribute (SiliconBlue)

```
module test
  (CLK, din1, din2, din3, din4,Q1, Q2, reset, gt1, gt2);

input gt1, gt2;
input CLK;
input reset /* synthesis syn_insert_buffer = "SB_GB_IO" */;
input din1;
input din2 /* synthesis syn_insert_buffer = "SB_GB_IO" */;
input din3 /* synthesis syn_insert_buffer = "SB_GB_IO" */;
input din4;
output reg Q1, Q2;

wire gt11 /* synthesis syn_insert_buffer = "SB_GB" syn_keep = 1
*/;

assign gt11 = gt1;

wire int_clk_glob;
wire int_clk_core;

wire int_clk_glob_gt;
wire int_clk_core_gt;

reg  reg_1, reg_2, reg_3, reg_4;

assign int_clk_glob_gt = CLK & gt11;
assign int_clk_core_gt = CLK & gt2;

always @(posedge int_clk_core_gt or negedge reset)
begin
  if (!reset)
  reg_1 <= 0;
  else
  begin
  reg_1 <= din1;
  reg_2 <= din2;
  Q1 <= reg_1 + reg_2;
  end
end
```


Copyright © 2013 Synopsys, Inc.

```
always @(posedge int_clk_glob_gt)
begin
  reg_3 <= din3;
  reg_4 <= din4;
  Q2 <= reg_3 + reg_4;
end

endmodule
```

Back

Example 2: VHDL syn_reduce_controlset_size example

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port ( rst : in std_logic;

     set : in std_logic;
     en : in std_logic_vector(1 downto 0);
     clk : in std_logic;
     din : in std_logic_vector (2 downto 0);
     dout : out std_logic_vector (2 downto 0) );
end entity test;

architecture test_arch of test is
attribute syn_reduce_controlset_size : integer;
attribute syn_reduce_controlset_size of test_arch : architecture
  is 3;

begin
process(clk, rst, set, en(0))
begin
  if (clk='1' and clk'event) then
     if (rst = '1') then
        dout(0) <= '0';
        else if (set = '1') then
           dout(0) <= '1';
           else if (en(0) = '1') then
              dout(0) <= din(0);
           end if;
        end if;
     end if;
  end if;
end process;

process(clk, rst, set, en(0))
begin
  if (clk='1' and clk'event) then
     if (rst = '1') then
        dout(1) <= '0';
        else if (set = '1') then
```


Copyright © 2013 Synopsys, Inc.

```
                dout(1) <= '1';
                else if (en(0) = '1') then
                   dout(1) <= din(1);
                end if;
             end if;
          end if;
       end if;
    end process;

    process(clk, rst, set, en(1))
    begin
       if (clk='1' and clk'event) then
          if (rst = '1') then
             dout(2) <= '0';
             else if (set = '1') then
                dout(2) <= '1';
                else if (en(1) = '1') then
                   dout(2) <= din(2);
                end if;
             end if;
          end if;
       end if;
    end process;

    end test_arch;
```

Back

Example 1: Verilog syn_reduce_controlset_size example

```
 module test(i1, r1,s1, e1, clk, o1)/* synthesis
syn_reduce_controlset_size = 3 */;

 input [3:1] i1;
 input clk;
 input r1;
 input s1;
 input [2:1] e1;
 output [3:1] o1;

 reg reg1, reg2, reg3;

 //reg1 FDRSE

 always@(posedge clk)
 if (r1)
     reg1 <= 1'b0;
 else if (s1)
     reg1 <= 1'b1;
 else if (e1[1])
     reg1 <= i1[1];

 //reg2 FDRSE

 always@(posedge clk)
 if (r1)
         reg2 <= 1'b0;
 else if (s1)
         reg2 <= 1'b1;
 else if (e1[1])
         reg2 <= i1[2];

 //reg3 FDRSE

 always@(posedge clk)
 if (r1)
     reg3 <= 1'b0;
 else if (s1)
     reg3 <= 1'b1;
 else if (e1[2])
```


Copyright © 2013 Synopsys, Inc.

```
        reg3 <= i1[3];

    assign o1 = {reg3,reg2,reg1};

    endmodule
```

Back

Example 2: VHDL syn_rw_conflict_logic example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram2Kx8 is port(
data_out : out std_logic_vector(7 downto 0);
data_in : in std_logic_vector(7 downto 0);
raddr0 : in std_logic_vector(10 downto 0);
waddr0 : in std_logic_vector(10 downto 0);
clk, we : in std_logic);
end entity;

architecture beh of ram2Kx8 is
type mem_type is array (2047 downto 0) of std_logic_vector
  (7 downto 0);
signal raddr0_in : std_logic_vector(10 downto 0);
signal mem : mem_type;
attribute syn_rw_conflict_logic : boolean;
attribute syn_rw_conflict_logic  of mem : signal is false ;
begin
  data_out <= mem(conv_integer(raddr0_in));

process(clk)
begin
  if clk'event and clk='1' then
     if (we = '1') then
        mem(conv_integer(waddr0)) <= data_in;
     end if;
  end if;
end process;

process(clk)
begin
  if clk'event and clk='1' then
     raddr0_in  <= raddr0;
  end if;
end process;

end beh;
```

Back

Copyright © 2013 Synopsys, Inc.

Example 1: Verilog syn_rw_conflict_logic example

```
module ram2Kx8 (data0, waddr0,raddr0, we0, clk0, dataout);
parameter d_width = 8;
parameter addr_width = 11;
parameter mem_depth = 2048;
input [d_width-1:0] data0;
input [addr_width-1:0] waddr0, raddr0;
input we0, clk0;
output [d_width-1:0] dataout;
reg [addr_width-1:0] reg_addr0;
reg [d_width-1:0] mem [mem_depth-1:0] /* synthesis
syn_rw_conflict_logic = 0 */;
assign dataout = mem[reg_addr0];

always @(posedge clk0)
begin
  reg_addr0 <= raddr0;
     if (we0)
        mem[waddr0] <= data0;
end
endmodule
```

Back

Index

Symbols

`ifdef 753
_est.srr file 602
SEARCHFILENAMEONLY directive 218
_ta.srm file 606
! character, find command 1789
? wildcard
  Timing Analyzer 300
.* connection (SystemVerilog) 814
.adc file 594
.areasrr file 601
.cdc file
  examples 1168, 1551
.edf file 603
.est file 551, 602
.fse file 602
.info file 602
  ROM 746
.ini file 594
.lpf file 1973
.name connection (SystemVerilog) 813
.nrf file 595
.opt file
  input 595
  output 602
.prj file 595
.sap
  annotated properties for analyst 604
.sar file 604
.sdc file 595
.sfp file 595
.srd file 604
.srm file 604
.srp file 604
.srr file 608
  See log file
  watching selected information 81
.srs file 605
  initial values (Verilog) 683
  See srs file
.sv file 596
  SystemVerilog source file 596
.ta file
  See timing report file 606
.tcl file
  input 596
  output 607
tcl file

  *See also*Tcl files 607
.v file 596
.vb board file 196
.vhd file 596
.vhm file 608
.vif file 607
.vm file 608
.xnf file (Xilinx). *See* xnf file (Xilinx)
$bits system function 825

Numerics

64-bit mapping 198

A

aborting a synthesis run 243
About this program command 363
Achronix
  attributes 1861
  directives 1861
  initial values 1837
  Speedster22iHD 1856
  technologies 1829
adc file (analysis design constraint) 594
add files
  -_include tcl argument 1641
Add Implementation command 168
Add P&R Implementation command 386

Add Place & Route Options File command 388
Add Place and Route Job command 386
Add Source File command 168
add_file Tcl command 1639
add_folder Tcl command 1645
add_to_collection command 1816
adder
  SYNCore 1064
Additional Products command 362
Adjust Pin View command 547
aggregate expressions 787
Align Regions command 546
all keyword, VHDL 2008 985
Allow Docking command 82
alsloc 1182
alspin 1185
alspreserve 1187
Alt key, selecting columns in Text Editor 99
Altera
  attributes 1944
  cliquing. *See* cliquing.
  constraints 1921
  device support 1866
  device support. *See also* Altera technologies 1866
  directives 1944
  distributed TMR syntax examples 1465
  DSP block inference 1904
  edif and vqm filenames 1922
  fanout limits 1911
  forward-annotation, Max+Plus II 1921
  gated clocks 1913
  generated clocks 1914
  grey boxes *See* grey boxes
  I/O registers power-up mode 1189
  IP core support 1876
  macro library, Verilog 1876
  macro library, VHDL 1876
  macros 1879
  mapping to ATOMs 1880
  Max+Plus II forward-annotated constraints 1921
  pad cell insertion 1881
  PLLs 1880
  prepared components method 1898
  qsf2sdc utility 994
  RAM inference 1882
  RAMs
    inference (Stratix) 1882
  register inference 1880
  reports 1917
  ROM inference 1896
  SOPC Builder 998
  specifying VHDL library 1876
  syn_radhardlevel TMR 1464
  syn_romstyle attribute 1502
  syn_useioff attribute 1564
  Synplify Premier tool device support 1866
  technology keywods 1942
  translating PIN file to constraint file 242
Altera black boxes
  pad cell insertion (Altera) 1881
Altera Clearbox
  megafunctions 1924
Altera device options
  Arria 1927
  Cyclone 1927
  MAX 1935
  MAX II 1931
  Stratix 1927
Altera file format options
  MAX 1937
Altera megafunctions
  DSP blocks 1904
Altera models
  Clearbox 1706
  set_option syntax 1706
Altera RAM
  byte-wide write enable 1884
Altera RAMs
  $readmemb/$readmemh initialization 683
  inference (Arria) 1882
  inference (Cyclone) 1882
  Verilog blocking statements and dual-port RAMs 721
  Verilog for Stratix RAM with control signals 724
  Verilog types 709
  VHDL types 941
Altera technologies
  MAX 1935
altera_io_powerup attribute 1189
altera_logiclock_location attribute 1191
altera_logiclock_size attribute 1193
always blocks
  Verilog 675
    combinational logic 691

event control 692
flip-flops 696
level-sensitive latches 697
multiple event control arguments 675
always_comb (SystemVerilog) 801
always_ff (SystemVerilog) 804
always_latch (SystemVerilog) 803
analysis design constraint file (.adc) 594
annotated properties for analyst
.sap 604
.timing annotated properties (.tap) 607
object properties for filtering 1786
append_to_collection command 1818
archive file (.sar) 604
archive utility
SEARCHFILENAMEONLY directive 218
copy tcl command 1663
unarchive tcl command 1663
area estimation
design modules 551
Design Planner regions 552
warning settings 554
area estimation file (.est) 602
area reduction
syn_packer_effort_level 1439
areasrr file
hierarchical area report 623
arithmetic operators
Verilog 639
Arrange VHDL files command 239
Arria 1927
device options 1927
file format options 1930
set_option device options 1929
set_option synthesis options 1930
arrow keys, selecting objects in Hierarchy Browser 439
arrow pointers for push and pop 437
assign_to_region Tcl command 1645
assignment operators
Verilog 640
VHDL 844
assignment statement
combinational logic (Verilog) 693
level-sensitive latches (Verilog) 697
VHDL 880
asymmetric block RAM
Xilinx 2084
asynchronous clock report
description 622
generation option 292
asynchronous sets and resets
Verilog 700
VHDL 893
asynchronous state machines
Verilog 708
VHDL 904
ATOM mapping, Altera 1880
attributes
alphabetical summary 1170
Altera 1944
custom 1316
global attribute summary 1632
specifying in the SCOPE spreadsheet 1166
specifying in the source code 758
specifying, overview of methods 1166
syn_assign_to_region 1230
syn_clock_priority 1241
syn_direct_reset 1254
syn_highrel_ioconnector 1352
syntax, Verilog 758
syntax, VHDL 963
attributes (Achronix) 1861
attributes (Microsemi) 2059
attributes (Xilinx) 2136
Attributes demo 102
Attributes panel, SCOPE 488
Attributes panel, SCOPE spreadsheet 1166
auto constraints 467
Maximize option 133
Maximize option (Constraints tab) 201
automatic task declaration 659

B

Back command 162
batch mode 53
bit-stream casting 777
bit-string literals 971
black box constraints
VHDL 962
black box directives
black_box_pad_pin 1195

black_box_tri_pins 1197
syn_black_box 1235
syn_force_seq_prim 1328
syn_isclock 1380
syn_resources 1497
syn_tco 1533
syn_tpd 1536
syn_tristate 1539
syn_tsu 1543
black boxes
See also macros, macro libraries
directives. *See* black box directives
for predefined macros (Altera) 1879
instantiating, Verilog 749
instantiating, VHDL 961
internal startup box (Xilinx) 1606
Lattice 1963
Microsemi 2031
source code directives 1236
timing directives 1536
Verilog 749
VHDL 961
Xilinx 2073
Xilinx macros 2096
black_box_pad_pin directive 1195
black_box_tri_pins directive 1197
block name on end (SystemVerilog) 799
block RAMs
inferring byte enables, Verilog 741
inferring byte enables, VHDL 954
Lattice 1952
parity bus, Xilinx 2095
syn_ramstyle attribute 2061
Xilinx single port and dual-port 2094
Xilinx, Verilog 738
Xilinx, VHDL 952
blocks
packing I/O (Xilinx) 2097
board files, HAPS 196
bottom-up synthesis 1770
buffering, Altera 1912
buffers
clock. *See* clock buffers
global 1605
global. *See* global buffers
output 1603
BUFGMUX/BUFGMUX_1 inference 2097
Build Project command 141
built-in gate primitives (Verilog) 689
bus bundling 347
bus constraints
multicycle (Altera) 1920
bus_dimension_separator_style command 1134
bus_naming_style command 1133
buses
compressed display 347
enabling bit range display 346
hiding in flattened Technology views 348
buttons and options, Project view 131
By any transition command 163
By input transitions command 163
By output transitions command 163
byte enables
in block RAM, Verilog 741
in block RAM, VHDL 954
byte-wide write enable RAM
Altera 1884

C

c_diff command (collections) 481
c_intersect command (colletions) 481
c_print command (collections) 481
c_sub command (collections) 481
c_symdiff command (collections) 481
c_symdiff command, examples 1799
c_union command (collections) 481
callback functions, customizing flow 1719
camera mouse pointer 141
case sensitivity, Tcl find command 1780
case statement
default 1204
VHDL 861
casting
static 777
casting types 777
cck.rpt file (constraint checking report) 601
cell interior display, enabling/disabling 347
Change File command 168
Change Implementation Name command 375
check boxes, Project view 131
Check Resource Center Messages command 357
Clear Parameters command 366
Clearbox
features 1924
primitives 1925

set_option parameters 1706
setting options 1706
cliquing 1932
CLKDLL
xc_clockbuftype attribute 1597
clock alias 296
clock alias (Legacy) 520
clock as object 296
clock buffering report, log file (.srr) 611
clock buffers
assigning resources 1421
inserting 1353, 1360
clock constraints
clock objects (Legacy) 520
clock edges (VHDL) 884
clock enables
inferring registers (Altera) 1880
inferring registers (Xilinx) 2076
inferring with syn_direct_enable 1248
net assignment 1248
clock frequency goals, tradeoffs using different 1767
clock groups
Clock Relationships (timing report) 618
forward-annotating, Xilinx 2115
clock groups (Legacy) 522
clock groups, SCOPE 473
clock paths, ignoring 503
clock pin drivers, selecting all 94
clock priority
forward-annotation to UCF 1242
syn_clock_priority 1241
clock relationships, timing report 618
clock report
asynchronous 615
detailed 620
clock skew, Xilinx 2105
Clock to Clock panel, SCOPE (Legacy) 525
Clock Tree, HDL Analyst tool 94
CLOCK_DEDICATED_ROUTE attribute 1199
clocks
asynchrounous report 622
declared clock 617
defining 94
defining on an instance (Xilinx) 2114
derived clock 617
edges in VHDL 884
forward-annotating for Xilinx DLLs and DCMs 2116
inferred clock 617
on black boxes 1380
system clock 618
Clocks panel, SCOPE 472
Clocks panel, SCOPE (Legacy) 519
Close command 141
Close Project command 142
code
ignoring with pragma translate off/on 1214
Collapse All command 402
collection commands
c_diff 1795
c_intersect 1796
c_list 1797
c_print 1798
c_symdiff 1798
c_union 1799
SCOPE 480
collections
Synopsys standard commands 1815
Collections panel, SCOPE 480
color coding
log file (.srr) 614
Text Editor 98
combinational logic
always_comb block (SystemVerilog) 801
Verilog 690
VHDL 865
combinational loop errors in state machines 905
combined data, port types (Verilog) 643
commands
accessing 139
Add Place & Route Job 386
Hierarchy Browser 402
menu
See individual command entries
set_modules (Tcl) 1802
Tcl
See Tcl commands
Tcl collection 1794
Tcl command equivalents 140
Tcl expand 1791
Tcl find 1774
Tcl hooks 1719

VIF
commands 1726
comma-separated sensitivity list (Verilog) 644
Comment Code command 148
commenting out code (Text Editor) 99
comments
Verilog 686
VHDL 881
companion parts, Stratix, HardCopy 1910
Compile Only command 237
Compile Physical Hierarchy command 238
compile point constraints
editing 404
forward-annotating (Cyclone, Arria) 1922
Xilinx 2117
compile points
allowed resources 1221
Altera 1915
Lattice 1982
Microsemi 2053
preserving hierarchy for Xilinx Place and Route 1625
update compile point timing data 1859
updating data (Microsemi) 2048
Xilinx 2104
Xilinx flow 2120
Compile Points panel, SCOPE 491
compiler
loop iteration, loop_limit 1207
loop iteration, syn_looplimit 1390
compiler directive
SEARCHFILENAMEONLY 218
IGNORE_VERILOG_BLACKBOX_GUTS 216
compiler directives
UI option 212
Verilog 215
compiler report, log file (.srr) 610
components, VHDL. *See* VHDL components
concurrent signal assignments (VHDL) 862
condition operator
VHDL 2008 970
conditional signal assignments (VHDL) 864
configuration statement
VHDL 920
VHDL generic mapping 920
VHDL multiple entities 922
VHDL port mapping 921
configuration, VHDL
declaration 914
specification 910
Configure External Programs command 352
Configure Mapper Parallel Job command 330
Configure Revsion Control Tools
dialog box 184
Configure Verilog Compiler command 330
Configure VHDL Compiler command 330
Configure Watch command 366
congestion
syn_packer_effort_level 1439
connectivity, enabling bit range display 346
constant function (Verilog 2001) 646
constant math function 657
constants (SystemVerilog) 782
constants, VHDL 848
SNS (Selected Name Support) 868
Constraint Check command 629
constraint checking report 629
constraint file
define_compile_point 1162
define_current_design 1163
syn_connect 1723
syn_create_err_net 1724
constraint files 455
.sdc 595
Altera Max+Plus II 1921
automatic. *See* auto constraints
editing compile point files 404
fdc and sdc precedence order 459
forward-annotation, Lattice 1973
Microsemi 2042
PLL 1880
rise and fall (Legacy) 523
route option (Legacy) 529
SCOPE spreadsheet 470
SCOPE spreadsheet (Legacy) 517
syn_reference_clock (Xilinx forward-annotation) 2112
tcl script examples 1768
translating 242
Xilinx 2066, 2110
constraint files (.sdc)
creating 115
constraint_file Tcl command 1648
constraints
auto constraints. *See* auto constraints
automatic. *See* auto constraints

check constraints 238
compile point (Xilinx) 2117
forward annotating (Xilinx) 2110
FPGA timing 1100
legacy timing 1135
report file 629
styles 457
translating Altera contraints for physical synthesis 994
types 454
Constraints panel
Implementation Options dialog box 200
constructs
interface 773, 816
union (SystemVerilog) 775
context declarations
VHDL 2008 981
context help editor 99
context of filtered schematic, displaying 444
context sensitve help
using the F1 key 50
context-sensitive popup menus
See popup menus
Continue on Error
Configure Compile Point Process 333
Continue on Error compile point option 333
continuous assignments (Verilog)
combinational logic 693
continuous assignments, Verilog
level-sensitive latches 697
control panel
displaying instances 568
displaying nets 568
displaying row sites 568
displaying signal pins 568
Physical Analyst view 567, 569
control panel (Physical Analyst) 567, 569
control signals on Stratix RAMs 724
control-sets, Xilinx 2083
Copy command 147
Copy File command 374
Copy Implementation command 375
copy_colelction command 1820
copying
for pasting 124
copying image
Create Image command 141
core cells
displaying 577
core voltage switch 1910
core voltage, Stratix III 1910
counter compiler
SYNCore 1076
CPLD technology
Xilinx 2131
Create Image command 141
Create Place & Route Options file dialog box 389
Create Sub-project (Design Block) command 392, 393
create_clock timing contraint 1102
create_generated_clock timing contraint 1104
create_region Tcl command 1649
critical paths 448
analyzing 449
creating new schematics 305
custom timing reports 291
finding 311, 449
setting maximum (Microsemi) 2047
Timing Report panel, Implementation Options dialog box 206
cross probing 577
cross-clock paths, timing analysis 618
cross-hair mouse pointer 111
cross-module referencing
Verilog 663
crossprobing 429
definition 429
Ctrl key
avoiding docking 114
multiple selection 110
zooming using the mouse wheel 112
custom attributes 1316
customer support, Synopsys 59
customization
callback functions 1719
Customize command 330
customizing
project files 335
Cut command 147
cutting (for pasting) 116
Cyclone 1927
device options 1927
file format options 1930
set_option device options 1929
set_option synthesis options 1930

D

D flip-flop, active-high reset, set (VHDL)
  asynchronous 894
  synchronous 896
data objects (SystemVerilog) 781
data type conversion 777
data types (SystemVerilog) 769
data types (VHDL) 841
data types, VHDL
  guidelines 881
DCM clock priority 1241
DCMs 2116
DDR register inference
  Xilinx 2076
declared clock 617
declaring and assigning objects (VHDL) 844
default assignment (VHDL) 901
default propagation 874
define_attribute
  syntax 1167
define_clock
  defining on an instance (Xilinx) 2114, 2117
  forward-annotation, Altera Max+Plus II 1921
  forward-annotation, Microsemi 2043
  forward-annotation, Xilinx 2111
define_clock_delay
  forward-annotation, Xilinx 2111
define_clock_delay legacy timing contraint 1139
define_compile_point
  Tcl 1162
define_current_design
  Tcl 1163
define_false_path
  forward-annotation, Microsemi 2043
  forward-annotation, Xilinx 2111
  using with syn_keep 1382
define_false_path legacy timing contraint 1140
define_global_attribute
  summary 1632
  syntax 1167
define_input_delay
  forward-annotation with syn_forward_io_constraints 1330
  forward-annotation, Altera Max+Plus II 1921
  forward-annotation, Xilinx 2111
  UCF forward-annotation 2113
define_input_delay legacy timing contraint 1143
define_multicycle_path
  forward-annotation, Microsemi 2043
  forward-annotation, Xilinx 2112
  using with syn_keep 1382
define_multicycle_path legacy timing contraint 1145
define_output_delay
  forward-annotation with syn_forward_io_constraints 1330
  forward-annotation, Altera Max+Plus II 1921
  forward-annotation, Xilinx 2111
  UCF forward-annotation 2113
define_output_delay legacy timing contraint 1149
define_path_delay
  forward-annotation, Microsemi 2043
  forward-annotation, Xilinx 2112
define_path_delay constraint
  forward-annotation in Xilinx 2112
define_path_delay legacy timing contraint 1151
define_reg_output_delay legacy timing constraint
  timing constraints
    define_reg_output_delay 1155
defining I/O standards 489
delay locked loops. *See* DLLs
delay paths
  POS 510
Delay Paths panel, SCOPE 486
delays
  forward annotating 1330
  forward-annotating with syn_forward_io_constraints 1330
Delete all bookmarks command 148
Delete Region Assignments command 546
Delete Region command 546
Delete Selected Assignments command 546
deleting
  See removing
Demos & Examples 101
derived clock 617

Design Block Properties command (hierarchical project management) 395
design flow
  customizing with callback functions 1719
design flows
  synthesis (Altera) 1867
  synthesis (Xilinx) 2066
Design Hierarchy view (hierarchcal project management) 67
Design Hierarchy view (hierarchcal project management)icons 68
design parameters (Verilog)
  extracting 215
Design Plan Editor 539
Design Plan Editor view 539
design plan file (.sfp) 595
Design Plan Hierarchical view 537
Design Plan Hierarchy view 537
Design Plan view 539
design plan view file (.srp) 305, 604
Design Planner
  popup menus 558
  Preferences dialog box 553
  Properties command 548
design planner constraint files (.sfp)
  creating 115
Design Planner option
  views 536
Design Planner tools menu
  Preferences command 553
  Show Replicated Assignments command 553
Design Planner view 536
Design Planning panel, Implementation Options dialog box 550
design size, schematic sheet
  setting 432
DesignWare options
  VHDL panel 209, 213
device option
  Update Compile Point Timing Data 1859
device options 1855, 2014
  CPLDs (Xilinx) 2131
  iCE65 Lattice technology 2014
  Lattice ispGAL, ispGDX, and ispLSI 2001
  Lattice ispMACH and MACH 2011
  Lattice ispXPGA 2004
  Lattice ispXPLD5000MX and ispLSI5000VE_UPS 2007
  Lattice ORCA 1986
  LatticeECP2 and LatticeECP/EC 1989
  Spartan technologies (Xilinx) 2122
  Virtex technologies (Xilinx) 2122
device options (Microsemi) 2054
Device panel
  Conservative Register Optimization 1698
  Implementation Options dialog box 195
device view
  Physical Analyst 571
dh_module_sources Tcl command 1651
dialog boxes
  Enhanced Instance Display 577
  Find Object (Physical Analyst) 587
  Implementation Options 194
  Physical Analyst Properties 584
  Region Properties 548
  Replicated Assignments 553
  Resolve Selection 591
diff_term attribute 1201
digital clock managers. *See* DCMs
directives
  SEARCHFILENAMEONLY 218
  Altera 1944
  beta features 218
  black box instantiation (VHDL) 961
  ignore syntax check 216
  IGNORE_VERILOG_BLACKBOX_GUTS 216
  specifying 758
  specifying for the compiler (Verilog) 215
  syntax, Verilog 758
  syntax, VHDL 963
directives (Achronix) 1861
directives (Microsemi) 2059
directives (Xilinx) 2136
directory
  examples delivered with synthesis tool 57
disable I/O insertion (Lattice) 1980
Disable Sequential Optimizations (Xilinx) 2102
disabling sequential optimizations 1696
display settings
  Project view 335
Dissolve Instances command 313, 446

Dissolve to Gates command 313
dissolving instances 313
distributed TMR
  syntax and examples 1465
DLLs 2116
docking 82
  avoiding 114
docking GUI entities
  toolbar 114
do-while loops (SystemVerilog) 797
DSP (Xilinx) 1264, 1269
DSP blocks
  Altera megafunctions 1904
  formal verificaion (Altera) 1909
  inferencing 2033
  packing resets (Altera) 1908
DSP blocks (Altera) 1904
DSP48 (Xilinx) 2081
  converting asynchronous reset registers 1239, 1241
DSP48 structures (Xilinx) 2081
DSP48E, wide adder inference 2082
dual-port RAM
  true dual-port RAM
    See multi-port RAMs
dual-port RAMs
  Altera Verilog examples 720
  error message (Stratix) 721, 949
  read first, Stratix 721, 949
  Verilog blocking statements (Altera) 721
  write first, Stratix 723
duplicate modules (Verilog)
  Tcl option 1686
dynamic range assignment (VHDL) 844

E

edf file 603
edf file. *See* edif file
edif file
  bus name styles 1280
  character case 1286
  port name length 1284
  scalar and array ports 1420
  syn_noarrayports attribute 1420
edif file (.edf) 603
edif files
  Altera naming requirements 1922
  port names (Xilinx) 2117
Edit Attributes command 403
Edit Compile Point Constraints command 404
Edit Design Plan command 375
Edit menu 147
  Advanced submenu 148
Edit Module Constraints command 404
Edit Regions command 546
Edit Run Configuration command 182
Editor Options command 330
editor view
  context help 99
else/elsif clauses
  VHDL 2008 985
enable prepacking 2082
Enable Slack Margin 295
enable_io_map.txt file 1881
encoding
  enumeration, default (VHDL) 208, 881
  state machine
    FSM Compiler 105
    FSM Explorer 107, 134
    guidelines
      Verilog 705
encryption
  asymmetric 1082
  methodologies 1081
  symmetric 1081
encryption algorithms 1081
encryptIP script 1090
  command-line arguments 1091
  execution 1095
  output methods 1092
  syntax 1091
encryptP1735 script 1082
  command-line arguments 1083
  decryption envelope 1086
  output methods 1088
  public keys 1085
  syntax 1082
enhanced display mode 577
Enhanced Instance Display dialog box 577
enhanced optimization
  effect on syn_macro and syn_user_instance 1394
enumerated types
  syn_enum_encoding directive 1314
enumerated types (SystemVerilog) 771
enumerated types (VHDL) 901

enumeration encoding, default (VHDL) 208, 881
environment variables
  accessing, get_env Tcl command 1653
errors, warnings, notes, and messages report
  log file (.srr) 613
est file 551, 602
Estimate All Regions command 552
Estimate Area command 238, 551
Estimate Regions command 552
estimation
  area 551, 552
estimation (setting in Design Planner)
  area
    warning settings 554
estimation file, area (.est) 551, 602
estimation log file (_est.srr) 602
events, defining outside process (VHDL) 885
examples
  Demos & Examples 101
  Interactive Attribute Examples 102
  Tcl find command syntax 1781
examples delivered with synthesis tool, directory 57
Exit command 142
Expand command
  current level 307
  hierarchical 306
Expand Inwards command 306
Expand Paths command
  current level 307
  hierarchical 306
Expand to Register/Port command
  current level 307
  hierarchical 306
Expanded Pin View command 547
expanding
  paths between schematic objects 306
Explorer, FSM
  enabling 134
  overview 107
extra initialization state, creating (VHDL) 902
Extract Parameters 215

F

factorials
  calculating 659
failures, timing (definition) 450
fall constraints (Legacy) 523
false paths
  architectural 503
  clocks as from/to points 512
  code-introduced 503
  defined 503
  POS 510
  priority 504
fanin (Lattice) 1981
fanout
  Microsemi 2045
Fanout Guide option (Xilinx) 2100
fanout limits
  overriding default 1397
  syn_maxfan attribute 1397
fanouts (Altera) 1911
fanouts (Lattice) 1980
fanouts (Xilinx) 2099
Fast Synthesis mode (Synplify Premier) 198
FDC
  create_clock constraint 1102
  create_generated_clock 1104
  reset_path 1106
  set_clock_groups 1108
  set_clock_latency 1111
  set_clock_route_delay 1113
  set_clock_uncertainty 1114
  set_datapathonly_delay 1116
  set_false_path 1118
  set_input_delay 1120
  set_max_delay 1122
  set_multicycle_path 1125
  set_output_delay 1128
  set_reg_input_delay 1131
  set_reg_output_delay 1132
  standard collection commands 1815
fdc
  precedence over sdc 459
fdc constraints 460
  generation process 458
fdc file
  relationship with other constraint files 455
FIFO compiler
  SYNCore 1026
FIFO flags
  empty/almost empty 1034
  full/almost full 1033

handshaking 1035
programmable 1036
programmable empty 1040
programmable full 1037
file format options
project command (Altera) 1943
File menu
Recent Projects submenu 142
File Options command 381
files
_est.srr 602
.adc 594
.areasrr 601
.cdc 594
.edf 603
.edf. *See* edif file
.est 551, 602
.fdc 594
.fse 602
.info 602, 746
.ini 594
.lpf 1973
.nrf 595
.opt 595, 602
.prj 51, 595
.sar 604
.sdc 595
.sfp 595
.srm 604, 606
.srp 305, 604
.srr 608
watching selected information 81
.srs 605
.ta 606
.ta *See also* timing report file 293
.tcl for Xilinx xtclsh 596, 607
.v 596
.vhd 596
.vhm 608
.vif 607
.vm 608
.xnf (Xilinx). *See* xnf file (Xilinx)
adding to project 170, 1639
area estimation (.est) 551, 602
compiler output (.srs) 605
constraint 1648
constraint (.adc) 594
constraint (.sdc) 595
copying 374
creating 115
customized timing report (.ta) 606
design component info (.info) 602
design plan (.sfp) 595
design plan view (.srp) 305, 604
edif (.edf) 603
estimation log (_est.srr) 602
include 1641
initialization (.ini) 594
log (.srr) 608
watching selected information 81
log. *See* log file
mapper output (.srm) 604, 606
Netlist Restructure (.nrf) 595
opening recent project 142
organization into folders 336
output
See output files
partitioned RTL view (.srp) 305, 604
physical synthesis netlist (.srp) 305, 604
project 1670
project (.prj) 51, 595
removing from project 168
replacing in project 171
RTL view (.srs) 605
srr 608
watching selected information 81
srs *See* srs file
stand-alone timing report (.ta) 284
state machine encoding (.fse) 602
Synplicity archive file (.sar) 604
synthesis output 601
Technology view (.srm) 604, 606
temporary 314
timing report. *See also* timing report file 293
verification (.vif) 607
verification interface format (.vif) 607
Verilog (.v) 596
VHDL (.vhd) 596
Xilinx (.opt) 595, 602
files for synthesis 594
Filter Schematic command 308
popup menu 367
filtered schematic
compared with unfiltered 416
filtering 442
commands 442
compared with flattening 447
critical paths 311
FSM states and transitions 96, 163

paths from pins or ports 312, 450
selected objects 308
timing reports 292
filtering critical paths 449
filtering design objects
Physical Analyst 589
Find again command 148
Find command
HDL Analyst 152
Physical Analyst view 574
Text Editor 148
find command
batch mode 1659
filter properties 1786
Find command (Physical Analyst) 587
Find Object dialog box
filtering design objects 589
Find Object dialog box (Physical Analsyt) 587
finding
critical paths 311, 449
information on synthesis tool 59
GUI 50
finding objects (Physical Analyst) 587
finite state machines
See state machines
Fix gated clock conversion report
log file (.srr) 613
Flatten Current Schematic command 446
filtered schematic 309
unfiltered schematic 308
Flatten Schematic command 446
Flattened Critical Path command 306
flattened schematic, creating 305
Flattened to Gates View command 305
Flattened View command 305
flattening
commands 444
compared with filtering 447
instances 313
schematics 308
selected instances 446
flip-flops
Verilog 696
flip-flops (VHDL) 883
Float command
Watch window popup menu 82
floating
toolbar 114
Floating License Usage command 363
floating toolbar popup menu 114
Floorplan Editor
selecting objects 580
floorplan objects 577
Floorplanned View command 305
folders
adding to project 1645
folders for project files 336
foreach_in_collection command 1821
forgotten assignment to next state, detecting (VHDL) 904
formal verification
Altera DSP blocks 1909
forward annotation
initial values 683
Forward Annotation of Initial Values
Verilog 683
Forward command 162
forward-annotation
Altera RAM init values 683
compile point constraints (Cyclone, Arria) 1922
ispLEVER 1973
Lattice constraints 1973
Microsemi 2042
RLOC 1618
RLOC_ORIGIN 1618
syn_forward_io_constraints attribute 1330
xc_props attribute 1613
Xilinx 2110
Xilinx clock groups 2115
Xilinx constraints 2110
Xilinx DLLs and DCMs 2116
forward-annotation (Lattice) 1973
FPGA Implementation Tools command 357
FPGA timing constraints 1100
frequency
cross-clock paths 619
Frequency (Mhz) option, Project view 132
from points
clocks 511
multiple 507
object search order (Timing Analyzer) 296
objects 506
timing analyzer 296
fse file 602

FSM coding style
  Verilog 706
  VHDL 899
FSM Compiler option, Project view 134
FSM Compiler, enabling and disabling
  globally
    with GUI 134, 1983
  locally, for specific register 106
FSM default state assignment (Verilog) 706
FSM encoding file (.sfe) 602
FSM Explorer
  enabling 134
  overview 107
FSM Explorer command 238
FSM Explorer option, Project view 134
FSM Table command 163
FSM toolbar 119, 120
FSM Viewer 95
  popup menu 369
  popup menu commands 369
FSMs
  optimizing with FSM Compiler 1701
  syn_encoding attribute 1305
FSMs (finite state machines)
  See state machines
Full View command 161
full_case directive 1203
functions
  Verilog constant math 657
  Verilog signed 656
  Verilog unsigned 656
  VHDL 2008 predefined 976
functions, selected name support (VHDL) 869

G

gate primitives, Verilog 689
gated clocks
  syn_force_seq_prim directive 1328
gated clocks (Altera) 1914
gated clocks (Lattice iCE60/iCE40) 1985
gated clocks (Xilinx) 2102
gated-clock conversion 221, 222
GCC & Prototyping Tools panel 222
generate statement
  VHDL 938
generate_instance_constraints Tcl command 1652
generated clocks (Altera) 1914, 2102
generated clocks (Lattice) 1985
Generated Clocks panel, SCOPE 478
generated-clock optimization 221, 222
generic technology library 600
generics
  VHDL 2008 packages 981
get_env Tcl command 1653
get_object_name command 1823
get_option Tcl command 1653
global attributes summary 1632
global buffers 1605
  defining 1339
  xc_global_buffers attribute 1605
global set/reset. *See* GSR
Go to SolvNet command 357
Goto command 148
Goto Net Driver command
  current level 307
  hierarchical 307
graphical user interface (GUI), overview 61
grey boxes
  using 1926
GSR
  xc_isgsr 1606
GSR resource (Lattice) 1980
GTECH library. *See* generic technology library
gtech.v library 600
gui
  synthesis software 45
GUI (graphical user interface), overview 61

H

HAPS board file 196
HardCopy
  companion parts 1910
HDL Analyst
  Find command 152
  Visual Properties 162
HDL Analyst menu 304, 579
  Current Level submenu 307
  Hierarchical submenu 306
  Physical Analyst submenu 579
  RTL submenu 305
  Select All Schematic submenu 315
  Select All Sheet submenu 315

Technology submenu 305
HDL Analyst Options command 331
HDL Analyst tool 415
accessing commands 417
analyzing critical paths 448
Clock Tree 94
crossprobing 429
displaying timing information 311
filtering designs 442
finding objects 427
hierarchical instances. *See* hierarchical instances
object information 418
preferences 432
push/pop mode 435
ROM table viewer 746, 958
schematic sheet size 432
schematics, filtering 442
schematics, multiple-sheet 432
status bar information 418
Synplify license 415
title bar information 432
HDL Analyst toolbar
See Analyst toolbar
HDL Analyst views 416
See also RTL view, Technology view
HDL files, creating 115
HDL parameter overrides 1654
hdl_define Tcl command 1654
hdl_param Tcl command 1654
header, timing report 616
help
online
accessing 50
Help command 362
Help menu 362
hidden hierarchical instances 423
are not flattened 446
Hide command
floating toolbar popup menu 114
Log Watch window popup menu 82
Tcl Window popup menu 85
View menu, Rats Nest submenu 547
Hide Instances command 312
hiding instances 312
hierarchical area report 623
.areasrr file 623
Hierarchical Critical Path command 305
hierarchical design, creating
Verilog 751
VHDL 906
hierarchical instances 421
compared with primitive 420
display in HDL Analyst 421
hidden 423
opaque 421
transparent 421
Hierarchical Project Management
configure revision control tools 184
hierarchical project management
Source Code Management (SCM) tools 391
hierarchical project management views 66
hierarchical schematic sheet, definition 432
Hierarchical View command 305
hierarchy
flattening 308
compared with filtering 447
flattening with syn_hier 1344
preserving for Xilinx Place and Route 1625
pushing and popping 435
schematic sheets 432
Verilog 751
hierarchy (VHDL) 906
Hierarchy Browser 438
changing size in view 90
Clock Tree 94
commands 402
finding schematic objects 427
moving between objects 94
popup menu 402
refreshing 402
RTL view 90
symbols (legend) 94
Technology view 92
trees of objects 93
hierarchy browser
enabling/disabling display 347
hierarchy separator 1133
high reliability
syn_radhardlevel 1462
High Reliability panel
Implementation Options dialog box 206
How to Use Help command 362

I

I/O buffers
  inserting 1376
  specifying I/O standards 1440
I/O constraints
  multiple on same port (Legacy) 530
I/O delays
  forward-annotating, Xilinx 2113
I/O insertion 1859
  Altera Cyclone 1928
  Altera Stratix 1928
  Arria 1928
I/O insertion (Altera) 1912
I/O insertion (Microsemi) 2047
I/O insertion (Xilinx) 2096
I/O packing
  disabling with syn_replicate 1494
I/O pads
  instantiating (Xilinx) 2096
I/O registers, power-up mode (Altera) 1189
I/O standards
  QSF constraints 996
I/O Standards panel, SCOPE 489
I/Os
  See also ports
  inferring Xilinx buffers with syn_diff_io 1245
iCE65 Lattice device options 2014
IEEE 1364 Verilog 95 standard 598
ieee library (VHDL) 850
if-then-else statement (VHDL) 860
ignored language constructs (Verilog) 641
ignored language constructs (VHDL) 840
impl Tcl command 1656
Implementation Directory 76
implementation options
  Options Panel 197
Implementation Options command 168, 194
Implementation Options dialog box 172, 194
  Constraints panel 200
  Design Planning panel 550
  Device panel 195
  High Reliability panel 206
  Netlist Restructure panel 550
  Options panel 197
  Place and Route panel 223
  Timing Report panel 205
  Verilog panel 211
  VHDL panel 207
implementation options, device
  partdata tcl command 1660
Implementation Results panel
  Options for implementation dialog box 203
implementations
  creating 168
  naming 375
Import IP
  commands 224
  Launch System Designer 236
Import IP menu 224
Import IP Package command 235
Import Xilinx ISE Project command 225
import_vivado_ip 1018
include command
  verilog library directories 213
include files 1641
indenting a block of text 98
indenting text (Text Editor) 98
index_collection command 1824
inference
  Xilinx BUFGMUX/BUFGMUX_1 2097
  Xilinx I/O buffers 2097
inferencing
  DSP blocks 2033
inferred clock 617
inferring RAMs
  VHDL 940
inferring RAMs (Xilinx) 2084
inferring ROMs
  VHDL 958
info file 746
info file (design component info) 602
ini file 594
INIT values
  syntax for forward-annotation 1617
init values
  in RAMs 877
initial value data file
  Verilog 680
Initial Values
  forward annotation 683
initial values
  $readmemb 678
  $readmemh 678
  Achronix 1837

Lattice ICE65/iCE40 1966
registers (Verilog) 676
initial values (Verilog)
netlist file (.srs) 683
initialization file (.ini) 594
input files 594
.adc 594
.ini 594
.nrf 595
.opt 595, 602
.sdc 595
.sfp 595
.sv 596
.v 596
.vhd 596
vhd 596
Inputs/Outputs panel, SCOPE 482
Inputs/Outputs panel, SCOPE (Legacy) 527
Insert Sub-project command 183, 397
inserting
bookmarks (Text Editor) 98
level-sensitive latches in design, warning 691, 880
Instance Properties command (hierarchical project management) 395
instances
controlling optimization 1585
dissolving 313
expanding paths between 306
expansion maximum limit 348
expansion maximum limit (per filtered sheet) 350
expansion maximum limit (per unfiltered sheet) 350
finding by name 148, 574
hiding and unhiding 312
hierarchical
dissolving 440
making transparent 440
hierarchical. *See* hierarchical instances
isolating paths through 312
making transparent 313
name display 346
naming syntax, Verilog constraints 1157
naming syntax, VHDL constraints 1159
preserving with syn_noprune 1431
primitive. *See* primitive instances
properties (Physical Analyst) 585
selecting all in schematic 315
instances (Physical Analyst) 568
Instances command
schematic selection 315
sheet selection 315
instantiating black boxes (Verilog) 749
instantiating black boxes (VHDL) 961
instantiating components (VHDL) 852, 866
instantiating gate primitives, Verilog 689
integer data type (VHDL) 843
Interactive Attribute Examples 102
interface construct 773, 816
interface information, timing report 619
IOB statements, Xilinx 1579
IP cores (SYNCore)
building ram models 245
IP encryption
Lattice 1951
IP encryption (Lattice) 1951
ise2syn utility
GUI command 225
Tcl syntax 1017
Isolate Paths command 312
isolating paths from pins or ports 450
ispGAL, ispGDX, and ispLSI
set_option synthesis options 2003
ispMACH and MACH
set_option synthesis options 2013
ispXPDL5000MX and ispLSI5000VE
set_option synthesis options 2010
ispXPGA
set_option synthesis options 2006

J

Job Status command 239, 243
job tcl command 1657

K

keyboard shortcuts 123, 542, 572
arrow keys (Hierarchy Browser) 439
keyword completion, Text Editor 98
keywords
all (VHDL 2008) 985
completing in Text Editor 98
SystemVerilog 836

L

labels, displaying 346

language
  guidelines (Verilog) 675
language constructs (Verilog) 639
language constructs (VHDL) 838, 840
language guidelines (VHDL) 880
latches
  always blocks (Verilog) 697
  concurrent signal assignment (VHDL) 887
  continuous assignments (Verilog) 697
  error message (VHDL) 889
  in timing analysis 448
  level-sensitive
    Verilog 697
  mapping (Xilinx) 2080
  process blocks (VHDL) 888
  SystemVerilog always_latch 803
Lattice
  attributes and directives 2021
  black boxes 1963
  Block RAM Support 1952
  disable I/O insertion 1980
  fanin limit 1981
  forward annotation 1973
  forward-annotation constraints 1973
  generated clocks 1985
  GSR resource 1980
  IP encryption 1951
  loc attribute 1206
  macros 1963
  map logic to macrocells 1981
  maximum cell fanin 1981
  maximum terms/macrocell 1981
  orca_padtype attribute 1208
  orca_props attribute 1210
  pad types (ORCA) 1208
  percentage of design to optimize for timing 1982
  pipelining 1984
  product families 1950
  reports 1974
  retiming 1984
  syn_useioff attribute 1570
  updating compile points 1982
Lattice device options
  ispMACH and MACH 2011
  ispXPGA 2004
  ispXPLD5000MX and ispLSI5000VE_UPS 2007
  LatticeSC/SCM and LatticeXP 1993
  MachXO 1997
  ORCA 1986
Lattice iCE65/iCE40
  gated clocks 1985
  initial values 1966
  set_option synthesis options 2016
Lattice iCE65/iCE40 options 2015
Lattice options
  LatticeECP2 and LatticeECP/EC 1990
  MachXO 1998
  ORCA 1986
Lattice RAM types (Verilog) 709
Lattice RAM types (vhdl) 941
LatticeECP/EC and Later
  set_option synthesis options 1992
LatticeSC/SCM and LatticeXP2/XP
  set_option synthesis options 1996
Launch Identify Instrumentor command 239, 243, 245
launch_system_designer command 1658
LCELLs 1935
legacy sdc file. *See* sdc files, difference between legacy and Synopsys standard
legacy timing constraints 1135
  define clock 1136
  define_clock_delay 1139
  define_false_path 1140
  define_input_delay 1143
  define_multicycle_path 1145
  define_output_delay 1149
  define_path_delay 1151
  define_reg_input_delay 1154
levels
  See hierarchy
level-sensitive latches
  Verilog 697
  VHDL
    unwanted 889
level-sensitive latches (VHDL)
  using concurrent signal assignments 887
  using processes 888
libraries
  general technology 599
  macro, built-in 597, 850
  technology-independent 599
  Verilog
    macro 749
  VHDL

attributes and constraints 597, 850
IEEE, supported 838
libraries (VHDL) 849
library and package rules, VHDL 851
library packages (VHDL), accessing 851
library statement (VHDL) 851
license
floating 363
saving 364
specifying in batch mode 53
License Agreement command 362
license queuing 55
License Request command 362
Limit Number of Paths 295
limitations
SystemVerilog 767
limited RAM resources
Verilog 710
VHDL 942
Linux, 64-bit mapping 198
literal
bit string 971
literals
SystemVerilog 769
loc attribute (Lattice) 1206
localparams
Verilog 2001 656
location constraints
RLOCs with synthesis attribute 2081
Log File
HTML 165
text 165
log file
displaying 161
Tcl commands for filtering 1721
log file (.srr) 608
watching selected information 81
Log File command
View menu 165
log file report 608
clock buffering 611
compiler 610
errors, warnings, notes, and messages 613
fix gated clock conversion 613
mapper 611
net buffering 611
resource usage 612
retiming 613
summary of compile points 612
timing 612
Log Watch Configuration dialog box 83
Log Watch window 81
Output Windows 89
popup menu 366
positioning commands 82
Log Watch Window command 161
log_filter Tcl command
syntax 1721
log_report Tcl command 1722
logical operators
VHDL 2008 970
loop_limit directive 1207
Lowercase command 148
LPMs
prepared components (Altera) 1898
LUT6_2 primitives 2082
LUTRAM
examples 717
LutRAMs
Stratix RAMs with control signals 724

M

MachXO and Platform
set_option synthesis options 2000
macro libraries
Altera Verilog 1876
Altera VHDL 1876
macromodule 639
macros
libraries 597, 850
MATH18X18 block 2032
Microsemi 2031
preserving with syn_macro 1392
SIMBUF 2032
macros (Lattice) 1963
map logic to macrocells (Lattice) 1981
map primitive 1610
mapper output file (.srm) 604, 606
mapper report
log file (.srr) 611
Mapping of Multipliers During Formal Verification
Altera Stratix 1915
margin, slack 449
MAX 1935
device options 1935

file format options 1937
set_option device options 1936
set_option synthesis options 1937
MAX II
file format options 1933
set_option device options 1933
set_option synthesis options 1933
maximum parallel jobs 334
maximum terms/macrocell (Lattice) 1981
measurement tool
Physical Analyst 581
megafunctions
Clearbox 1924
grey boxes 1926
memory compiler 245
memory, saving 314
menubar 139
Menus
Import IP 224
menus
context-sensitive
See popup menus
Edit 147
HDL Analyst 304, 579
Help 362
Options 330, 579
popup
See popup menus
Project 168
Run 237, 551
Tools, Synplify Premier DP Design Planner 329, 552
View 160
message viewer
description 85
Messages
Tcl Window command 161
messages
Resource Center 357
Messages Tab 85
Messages window 84
Microsemi
alsloc attribute 1182
alspin attribute 1185
alspreserve attribute 1187
assigning I/O ports 1185
attributes 2059
black boxes 2031
compile point synthesis 2053
compile point timing data 2048
device options 2054
directives 2059
features 2029
forward-annotation, constraints 2042
I/O insertion 2047
macros 2031
MATH18X18 block 2032
Operating Condition Device Option 2049
preserving relative placement 1182
product families 2028
reports 2043
retiming 2047
SIMBUF macro 2032
syn_radhardlevel attribute 1462
Tcl implementation options 2056
Microsemi implementing RAM 2036
model template, VHDL 881
modules
naming syntax, Verilog constraints 1157
naming syntax, VHDL constraints 1158
renaming 1493, 1551
modules, Verilog 685
mouse button operations 110
mouse operations 107
Mouse Stroke Tutor 108
Mouse Stroke Tutor command 362
mouse wheel operations 112
Move command
floating toolbar window 114
Log Watch window popup menu 82
Tcl window popup menu 85
moving between objects in the Hierarchy Browser 94
moving GUI entities
toolbar 114
multicycle constraints
forward-annotation (Altera) 1920
multicycle paths
clocks as from/to points 511
examples 501
forward-annotating to UCF 2112
POS 510
syn_reference_clock 1491
using different start/end clocks 500
multidimensional array
Verilog 2001 660
multiple drivers
resolving 1707

Multiple File Compilation Unit
  Verilog panel 212
multiple projects
  displaying project files 337
multiple target technologies, synthesizing with Tcl script 1766
multiple-sheet schematics 432
multiplexer (Verilog) 692
multipliers
  DSP blocks 2033
multi-port RAM examples
  Verilog 714
Multi-port RAM extraction
  Verilog 711, 942
  VHDL 711, 942
multi-port RAM extraction
  VHDL 946
multi-port RAM VHDL example 947
multi-port RAMs
  nram inference, VHDL 711, 942, 946
  Xilinx implementations 2095
MultiProcessing
  Continue on Error mode 333
multiprocessing
  maximum parallel jobs 334
multisheet schematics
  transparent hierarchical instances 434

N

naming
  objects (VHDL) 844
  region 548
naming rules 1132
naming syntax
  Verilog 1157
  VHDL 1159
navigating
  among hierarchical levels
    by pushing and popping 435
    with the Hierarchy Browser 438
  among the sheets of a schematic 432
ncf file
  forward-annotation 2110
nesting design details (display) 440
net buffering report, log file 611
net drivers
  displaying and selecting 307
netlist file 608
  initial values (Verilog) 683
netlist formats
  Implementation Options dialog box, Implementation Results panel 205
Netlist Restructure file (.nrf) 595
Netlist Restructure panel, Implementation Options dialog box 550
nets
  expanding hierarchically from pins and ports 306
  finding by name 148, 574
  naming syntax, Verilog constraints 1157
  naming syntax, VHDL constraints 1159
  preserving (Microsemi) 1187
  preserving with alspreserve (Microsemi) 1187
  preserving with syn_keep 1382
  properties (Physical Analyst) 586
  selecting instances on 307
nets (SystemVerilog) 783
New command 141, 543
New Implementation command 172
New Project command 142
Next Bookmark command 148
Next Error command 240
Next Sheet command 162
Normal View command 161
nram
  VHDL 711, 942, 946
  Xilinx implementations 2095
nrf file 595
numeric_bit IEEE package (VHDL) 850
numeric_std IEEE package (VHDL) 850

O

object information
  status bar, HDL Analyst tool 418
  viewing in HDL Analyst tool 418
object prefixes
  Tcl find command 1778
object properties
  annotated properties for analyst 1786
object search order (Timing Analyzer) 296
object types
  Tcl find command 1778
objects
  crossprobing 429
  displaying compactly 347

dissolving 440
expanding paths between 306
filtering 308
finding 587
making transparent 440
selecting in Floorplan Editor 580
selecting overlapping in Physical Analyst view 591
unselecting
all in schematic 315
objects (VHDL)
naming 844
objects, schematic
See schematic objects
OFFSET constraints
unsupported 1016
OFFSET I/O constraints (Xilinx) 1628
Online Documents command 362
Online help
F1 key 50
online help
accessing 50
opaque hierarchical instances 421
are not flattened 446
Open command
File menu 141
Open Project command 141
open_design command 1659
open_file command 1660
opening
project 141
openIP encryption 1098
operators
set membership (SystemVerilog) 790
streaming (SystemVerilog) 789
Tcl collection 1794
type (SystemVerilog) 794
Verilog 639
VHDL
assignment 844
Selected Name Support (SNS) 870
sharing in case statements 865
SNS 870
VHDL 2008 condition 970
VHDL 2008 logical 970
VHDL 2008 relational 971
operators (SystemVerilog) 786
operators (VHDL) 853
opt file 595, 602
optimization
state machines 105
option settings
reporting 1653
options
Achronix Speedster22iHD 1856
Altera Cyclone 1927
Altera Stratix 1930
CPLDs (Xilinx) 2132
Lattice iCE65/iCE40 2015
LatticeSC/SCM and LatticeXP 1994
Project view 131
Frequency (Mhz) 132
FSM Compiler 134
FSM Explorer 134
Pipelining 134
Resource Sharing 134
Retiming 135
prototyping 222
setting 1679
setting with set_option Tcl command 1768
Spartan (Xilinx) 2124
Virtex (Xilinx) 2124
options (Microsemi) 2056
Options command
Xilinx submenu 331
Options for implementation dialog box
Implementation Results panel 203
Options menu 330, 579
Quartus submenu (Altera) 331
Options panel
Implementation Options dialog box 197
ORCA
set_option synthesis options 1988
orca_padtype attribute (Lattice) 1208
orca_props attribute 1210
Other panel, SCOPE (Legacy) 534
output buffers
selection 1603
output drivers
slowing transition times 1624
transition time 1599
output files 601
_est.srr 602
.areasrr 601
.edf 603
.edf. *See* edif file
.est 551, 602

.info 602, 746
.sar 604
.srm 604, 606
.srp 305, 604
.srr 608
watching selected information 81
.srs 605
.ta 606
.vhm 608
.vif 607
.vm 608
.xnf (Xilinx). *See* xnf files (Xilinx). 2064
log. *See* log file
netlist 608
srs
See srs file
See also files
Output Windows 89
overlapping objects
selecting in Physical Analyst view 591
overriding FSM Compiler 1696, 1928, 1932
overriding parameter value, Verilog 688
Overview of the Synopsys FPGA Synthesis Tools 42

P

packages 811
VHDL 2008 978
VHDL 2008 generics 981
packages, VHDL 849
pad cell insertion file, Altera 1881
pad locations
See also pin locations
pad types
Lattice ORCA 1208
Xilinx 1611
pads
disabling I/O insertion 1859
disabling I/O insertion (Xilinx) 2096
Pan command 161
panels
Implementation Options dialog box
Design Planning 550
Netlist Restructure 550
parallel_case directive 1212
parameter data types
SystemVerilog 784
parameters
overriding HDL 1654
SYNCore adder/subtractor 268
SYNCore byte-enable RAM 261
SYNCore counter 272
SYNCore FIFO 247
SYNCore RAM 257
SYNCore ROM 264
parity bus, Xilinx block RAM 2095
partdata tcl command 1660
partitioned RTL view file (.srp) 305, 604
partitioning of schematics into sheets 432
Paste - Replicate command 546
Paste command 147
pasting 116
path delays
clocks as from/to points 512
path filtering 295
paths
expanding hierarchically from pins and ports 306
percentage of design to optimize for timing
Lattice 1982
performance summary, timing report 616
Physical Analyst
command summary 580
control panel 567, 569
cross probing 577
device view 571
Enhanced Instance Display 577
input files 569
instance commands 582
net commands 584
properties 584
routing nets 568
Physical Analyst view
Configure Enhanced Instance Display command 575
Cross Probing command 575
filter command 575
Find command 587
finding objects 587
Go to Location command 576
instance properties 585
net properties 586
Physical Analyst Properties command 575
Selection Transcription command 575
Show Critical Path command 575

unfilter command 575, 581
physical constraints
translating QSF constraints 994
physical optimization
description 44
physical synthesis
translating QSF constraints 994
physical synthesis netlist file (.srp) 305, 604
pin location constrants
QSF 995
pin locations
forward annotating 1388
Lattice 1206
Microsemi 1185
specifying (Xilinx) 2097
pins
displaying
on transparent instances 425
displaying names 346
displaying on technology-specific primitives 426
displaying on transparent instances 313
expanding hierarchically from 306
expanding paths between 306
isolating paths from 312, 450
maximum on schematic sheet 350
pipelining
option 134
syn_pipeline attribute 1442
pipelining (Altera) 1913
pipelining (Lattice) 1984
pipelining (Xilinx) 2101
Pipelining option, Project view 134
Place & Route
creating a new options file for Xilinx 389
place & route
run from the synthesis tool 388
Place & Route options file 388
adding for Xilinx 388
place & route options file
specifying 388
Place and Route constraint file (Microsemi) 2042
Place and Route panel
Implementation Options dialog box 223
place and route tcl commands
job 1657
PLL constraints 1880
pointers, mouse
cross-hairs 111
push/pop arrows 437
zoom 161
popping up design hierarchy 435
popup menus
floating toolbar 114
FSM Viewer 369
Hierarchy Browser 402
Log Watch window 82, 83, 366
Log Watch window positioning 82
Project view 372
RTL view 402
Tcl window 85, 366, 367
Technology view 402
port names
in EDIF files (Xilinx) 2117
port placement 1608
ports
displaying names 346
expanding hierarchically from 306
expanding paths between 306
finding by name 148
isolating paths from 312
naming syntax, Verilog constraints 1157
naming syntax, VHDL constraints 1159
selecting all in schematic 315
ports (VHDL) 846
Ports command
schematic 315
sheet 315
ports connections (SystemVerilog) 813
POS
interface 509
pragma translate_off directive 1214
pragma translate_on directive 1214
precedence of constraint files 459
predefined enumeration types (VHDL) 841
predefined functions
VHDL 2008 976
predefined packages (VHDL) 850
preferences
HDL Analyst tool 432
project file display 335
Project view display 112
Preferences command
Design Planner tools menu 553
Preferences dialog box
Design Planner 553
Preferred License Selection command 363

prefixes
  Timing Analyzer points 296
PREP benchmarks
  Verilog 750
  VHDL 967
prepared components
  Altera VHDL 1898
Previous bookmark command 148
Previous Error/Warning command 240
Previous Sheet command 162
primitive instances 420
primitives
  dividing 1714
  internal logic, displaying 347
  pin names in Technology view 426
primitives, Verilog 689
Print command 141
Print Setup command 141
printing
  view 141
printing image
  Create Image command 141
priority encoding 1212
private key 1082
prj file 51, 595
probes
  inserting 1454
process keyword (VHDL) 858
process template (VHDL)
  modeling combinational logic 858
process template, VHDL 883
Process View 77
processes, VHDL 880
Product of Sums
  See POS
program_terminate command 1661
program_version command 1662
project command
  Altera file formats 1943
  CPLDs (Xilinx) 2135
  Spartan (Xilinx) 2130
  Virtex (Xilinx) 2130
project files
  organization into folders 336
project files (.prj) 51, 595
Project Files view 67
Project menu 168
  commands 168
Project Options command 376
project results
  Implementation Directory 76
  Process View 77
  Project Status View 69
Project Results View 69
Project Status View 69
project Tcl command 1662
Project view 62
  buttons and options 131
  display settings 335
  options 131
  popup menu 372
  setting up 335
  Synplify Premier/DP 62
  Synplify Pro 62
  Synplify tool 62
Project View Options command 330
Project window 62
project_data Tcl command 1669
project_file Tcl command 1670
project_name_cck.rpt file 629
projects
  adding files 170
  closing 142
  creating (Build Project) 141
  creating (New) 142
  displaying multiple 337
  opening 141
Promote Global Buffer Threshold (Microsemi) 2045
properties
  find command 1786
  Physical Analyst 584
  project 1669
Properties command
  Design Planner 548
  instances in Physical Analyst 585
  nets in Physical Analyst 586
  View menu
    Synplify Premier DP Design Planner 547
prototyping options 222
public key 1082
Push Tristates
  Verilog panel 212
Push/Pop Hierarchy command 162
push/pop mode, HDL Analyst tool 435

Q

QSF constraints
  example 996
  supported 995
qsf2sdc 994
  syntax 994
qsf2syn utility 1003
  syntax 1003
Quartus
  .vqm netlist 1866
  converting constraints to sdc 994
  edif and vqm file names 1922
  importing projects from, qsf2syn syntax 1003
  library versions 1876
  qsf2sdc utility. *See* qsf2sdc 994
Quartus megafunctions
  pad cell insertion 1881
quitting a synthesis run 243

R

radiation effects. *See* high reliability
RAM
  byte-wide write enable RAM (Altera) 1884
RAM implementations
  Microsemi 2036
RAM inference
  additional logic (VHDL) 942
  Altera Stratix 1882
  Arria 1882
  Cyclone 1882
  multi-port RAM, VHDL 711, 942, 946
  synchronous RAM (VHDL) 944
  working with limited RAM resources (VHDL) 942
RAM inference (Altera) 1882
RAM inference (Verilog) 709
RAM inference (VHDL) 940
RAM inference (Xilinx) 2084
RAM mapping
  Altera Stratix 1883
RAM types (Verilog)
  Altera 709
  Lattice 709
  Xilinx 709
RAM types (vhdl)
  Altera 941
  Lattice 941
  Xilinx 941
RAMs
  additional components 710
  implementation styles 1467
  initial values (Verilog) 678
  limited resources, Verilog 709
  multi-port, Xilinx 2095
  technology support 1469
  Virtex-5 743
recent projects, opening 142
recording command 1673
Redo command 147
Refresh command 366
Region Properties dialog box 548
regions
  assigning critical paths 1645
  creating 1649
regions, naming 548
register inference (Xilinx) 2076
register packing
  syn_useioff attribute 1576
  syn_useioff attribute, Altera 1564
  syn_useioff attribute, Lattice 1570
  syn_useioff attribute, Xilinx 1579
registers
  DDR (Xilinx) 2076
  inferring clock enables (Xilinx) 2076
  preserving with syn_preserve 1448
registers (VHDL) 883
Registers panel, SCOPE 484
Registers panel, SCOPE (Legacy) 531
regular expressions
  Tcl find command 1778
relational operators
  VHDL 2008 971
relative location
  alsloc (Microsemi) 1182
  xc_map (Xilinx) 1610
  xc_rloc 1622
  xc_uset 1630
relative placement. *See* RLOCs
relative position, Xilinx components 2081
Reload command 402
Remove Files From Project command 168
Remove Implementation command 375
remove_from_collection command 1826

removing
  bookmark (Text Editor) 98
  window (view) 114
Replace command
  Text Editor 148
replacing
  text 158
Replicated Assignments dialog box 553
replication
  Altera 1912
  disabling 1494
report_clocks command 1673
reports
  constraint checking (cck.rpt) 629
  hierarchical area report 623
  resource usage (Xilinx) 2107
  resource usage, Altera 1917
  resource usage, Lattice 1974
  timing report (.ta file) 284
reports (Lattice) 1974
reset_path timing contraint 1106
resets
  packing in Altera DSP blocks 1908
  Verilog 699
  VHDL 893
    detecting problems 902
Resolve Multiple Drivers option 1707
Resolve Selection dialog box 591
resolving conflicting timing constraints 513
Resource Center
  See Technical Resource Center
resource library (VHDL), creating 852
resource sharing
  Resource Sharing option 200
  syn_sharing directive 1518
  VHDL 865
Resource Sharing option, Project view 134
resource usage report, log file 612
Resynthesize All command 237
retiming
  report, log file 613
  syn_allow_retiming attribute 1216
retiming (Altera) 1913
retiming (Lattice) 1984
retiming (Microsemi) 2047
retiming (Xilinx) 2101
Retiming option, Project view 135
RLOCs 2081
  specifying with synthesis attribute 2081
ROM compiler
  SYNCore 1058
ROM inference
  VHDL 958
ROM inference (Altera) 1896
ROM inference (Verilog) 746
ROM inference (VHDL) 958
ROM initialization
  with Verilog generate block 748
ROM initializtion
  with rom.info file 747
ROM tables
  Verilog 746
  VHDL 958
rom.info file 746
ROMs
  Altera implementations 1897
  syn_romstyle attribute 1500, 1509
  syn_romstyle attribute, Altera 1502
  syn_romstyle attribute, Lattice 1507
  syn_romstyle attribute, Xilinx 1511
routing nets (Physical Analyst) 568
RTL view 90
  displaying 117, 1660
  file (.srs) 605
  opening hierarchical view 305
  popup menu 402
  popup menu commands 402
  primitives
    Verilog 702
    VHDL 894
  printing 141
rules
  library and package, VHDL 851
Run All Implementations command 239
Run Background Compile command 331
Run Foreground Compile command 331
Run menu 237, 551
Run Tcl Script command 239, 240
running place & route 388

S

s2o utility 1098
sar file
  Archive Project command 175
Save All command 141
Save As command 141

Save command 141
scalable adder, creating (Verilog) 687
scalable architecture, using (VHDL) 936
scalable designs (VHDL) 934
scaling by overriding parameter value, Verilog
  with # 688
  with defparam 688
scan chains
  Altera DSP blocks 1906
schematic objects
  crossprobing 429
  definition 418
  displaying compactly 347
  dissolving 440
  expanding paths between 306
  filtering 308
  finding 427
  making transparent 440
  status bar information 418
  unselecting all 315
schematic sheets 432
  hierarchical (definition) 432
  navigating among 432
  setting size 432
schematics
  configuring amount of logic on a sheet 432
  crossprobing 429
  displaying labels 346
  filtered 416
  filtering commands 442
  flattening 308
  flattening compared with filtering 447
  flattening selectively 446
  hierarchical (definition) 432
  multiple-sheet 432
  multiple-sheet. *See also* schematic sheets
  navigating sheets 162
  object information 418
  opening hierarchical RTL 305
  partitioning into sheets 432
  sheet connectors 347, 419
  sheets
    navigating among 432
    size, setting 432
  size in view, changing 90
  unfiltered 416
  unfiltering 444
  unselecting objects 315
SCM tools
  configure 184
SCOPE
  Attributes panel 488
  clock groups 473
  Clocks panel 472
  Collections panel 480
  Compile Points panel 491
  Delay Paths panel 486
  for legacy sdc 463
  Generated Clocks panel 478
  I/O Standards panel 489
  Inputs/Outputs panel 482
  Registers panel 484
  specifying RLOCs 2081
  TCL View 494
SCOPE spreadsheet
  Attributes panel 1166
  popup menu commands 366
  starting 470
SCOPE timing constraints summary 471
SCOPE timing constraints summary (Legacy) 518
sdc
  fdc precedence 459
  SCOPE for legacy files 463
  standard sdc collection commands 1815
SDC constraints
  from Altera QSF 995
sdc file
  difference between legacy and Synopsys standard 457
  translated UCF constraint examples 1006
sdc2fdc utility 988
Search SolvNet
  using 103, 363
Select All command 148
Select All States command 163
Select in Analyst command 367
Select Net Driver command
  current level 307
  hierarchical 307
Select Net Instances command
  current level 307
  hierarchical 307
Select Place & Route option file dialog box 388
Selected command 163
Selected Name Support (SNS), VHDL 868

selecting
  text column (Text Editor) 99
selecting multiple objects using the Ctrl key 110
sensitivity list (VHDL) 859
sequential elements
  naming 1133
sequential logic
  SystemVerilog
    sequential logic 804
  VHDL
    examples 966
sequential logic (Verilog) 695
sequential logic (VHDL) 865
sequential optimization, preventing with syn_preserve 1448
sequential optimizations
  disabling 1696, 1928, 1932
  disabling (Altera) 1914
  disabling (Xilinx) 2102
sequential shift components
  syn_srlstyle attribute 1524
set and reset signals (VHDL) 893
Set Library command 168
set modules command (collections) 481
set modules_copy command (collections) 481
Set Slack Margin command 311
Set VHDL Library command 168
set_clock_groups timing contraint 1108
set_clock_latency timing contraint 1111
set_clock_route_delay timing contraint 1113
set_clock_uncertainty timing contraint 1114
set_datapathonly_delay legacy timing constraint 1156
set_datapathonly_delay timing constraint 1116
set_false_path timing contraint 1118
set_hierarchy_separator command 1133
set_input_delay timing contraint 1120
set_max_delay timing contraint 1122
set_multicycle_path timing contraint 1125
set_option
  Resolve Multiple Drivers 1707
set_option command
  Altera description 1937
  Altera MAX device syntax 1936
  Altera MAX II and APEX device syntax 1933
  Altera MAX II and APEX synthesis syntax 1933
  Altera MAX synthesis syntax 1937
  Altera Stratix, Cyclone, and Arria device syntax 1929
  Altera Stratix, Cyclone, and Arria synthesis syntax 1930
set_option Tcl command 1679
set_output_delay timing contraint 1128
set_reg_input_delay timing constraint 1131
set_reg_output_delay timing constraint 1132
set_rtl_ff_names command 1133
set/reset priority (Microsemi) 1452
sets and resets
  VHDL 893
sets and resets (Verilog) 699
settings
  reporting option 1653
sheet connectors 347, 419
Shift key 114
shift registers
  Altera definition 1881
shortcuts
  keyboard
    See keyboard shortcuts
Show All Hier Pins command 313
Show All Regions command 547
Show command 547
Show Compile Points command 376
Show Context command 312
Show Critical Path command 311
Show Replicated Assignments command 552, 553
Show Selected command 547
Show Timing Information command 311
sign casting 777
signal assignments
  Verilog, always blocks 696
  VHDL
    conditional 864
    simple and selected 863
signal assignments (VHDL) 844
  concurrent 862
signal pins (Physical Analyst)
  displaying 568
signed arithmetic (VHDL) 843

signed functions 656
signed multipliers (Verilog) 674
signed signals, Verilog 2001 645, 646, 658
SIMBUF macro 2032
SIMD mode
  using syn_dspstyle attribute 1275
simple component instantiation (VHDL) 866
simple gates, Verilog 689
simple signal assignments, VHDL 863
simulation
  using enumerated types, VHDL 902
simulation mismatches
  full_case directive 1205
sites (Physical Analyst) 568
size casting 777
sizeof_collection command 1828
slack
  cross-clock paths 619
  defined 617
  margin
    definition 450
    setting 311, 449
  slack margin 295
slice_primitive Tcl command 1714
SLR (Super Logic Regions) 2065
SNS (Selected Name Support), VHDL 870
  constants 868
  demand loading 873
  functions and operators 869
  user-defined function support 871
SolvNet
  search 103, 363
SolvNet support 353
SolvNet Support command 353
SOPC2Syn 998
source code
  specifying RLOCs 2081
source files
  See also files
  adding to VHDL design library 849
  creating 115
srd file 604
SRL tables (Xilinx) 2079
srm file 604, 606
  hidden logic not saved 314
srp file 305, 604
srr file 608
  See log file
  watching selected information 81
srs file 605
  hidden logic not saved
  initial values (Verilog) 683
Stacked Silicon Interconnect
  Xilinx technology 2065
standard IEEE package (VHDL) 850
standards, supported
  Verilog 598
  VHDL 597
Start Design Manager command 331
Start Floorplanner command 331
Start ISE Project Navigator command 331
start/end clock period
  multicycle constraints (Legacy) 533
start/end points
  Timing Report panel, Implementation Options dialog box 206
starting Synplify 52
starting Synplify Premier 52
starting Synplify Premier DP 52
starting Synplify Pro 52
startup blocks
  Xilinx 1606
state machines
  See also FSM Compiler, FSM viewer, FSMs.
  asynchronous
    Verilog 708
    VHDL 893
  displaying in FSM viewer 315
  encoding
    displaying 96
    FSM Compiler 105
    FSM Explorer 107, 134
    syn_encoding attribute
      Verilog 706
      VHDL 898
  encoding file (.fse) 602
  enumerated type, VHDL 901
  enumerated types 1314
  extracting 1529
  filtering states and transitions 96, 163
  optimization 105
  state encoding, displaying 96

SystemVerilog example with enumerated types 772
Verilog 707
state machines (Verilog) 705
state machines (VHDL) 897
state values (FSM), Verilog 707
static casting 777
Status Bar command 160
status bar information, HDL Analyst tool 418
std IEEE library (VHDL) 850
std_logic_1164 IEEE package (VHDL) 850
std_logic_arith IEEE package (VHDL) 850
std_logic_signed IEEE package (VHDL) 850
std_logic_unsigned IEEE package (VHDL) 850
stopping a synthesis run 243
Stratix 1927
companion parts 1910
core voltage for Stratix III 1910
device options 1927
file format options 1930
set_option device options 1929
set_option synthesis options 1930
streaming operator
SystemVerilog 789
structural designs, Verilog 751
structural netlist file (.vhm) 608
structural netlist file (.vm) 608
Submit Support Request command 353
subprojects
importing Vivado IP 1020
subtractor
SYNCore 1064
summary of compile points report
log file (.srr) 612
Super Logic Regions 2065
supported language constructs (Verilog) 639
supported language constructs (VHDL) 838
supported standards
Verilog 598
VHDL 597
sxml2pxml Tcl command
using the standalone XML converter 1716
symbols
enabling name display 346
finding by name 148, 574
Hierarchy Browser (legend) 94
syn_allow_retiming attribute 1216
syn_assign_to_region attribute 1230
syn_assign_to_slr attribute 1231
syn_auto_insert_bufg attribute 1233
syn_auto_partition attribute 1234
syn_black_box directive 1235
syn_clean_reset attribute 1239
syn_clock_priority attribute 1241
syn_connect 1723
syn_cp_use_fast_synthesis attribute 1244
syn_create_err_net 1724
syn_diff_io attribute 1245
syn_direct_enable attribute 1248
syn_direct_reset Attribute 1254
syn_disable_purifyclock attribute 1262
syn_dspstyle attribute 1264, 1269
using SIMD mode 1275
syn_edif_bit_format attribute 1280
syn_edif_name_length attribute 1284
syn_edif_scalar_format attribute 1286
syn_encoding
compared with syn_enum_encoding directive 1317
using with enum_encoding 1317
syn_encoding attribute 1305
FSM encoding style
Verilog 706
VHDL 898
syn_enum_encoding
using with enum_encoding 1317
syn_enum_encoding directive 1314
compared with syn_encoding attribute 1317
not for FSM encoding 899
syn_expand_coreloc attribute 1319
syn_fast_auto attribute 1320
syn_force_pads attribute 1327
syn_force_seq_prim directive 1328
syn_forward_io_constraints
effect on forward-annotation (Altera) 1920
syn_forward_io_constraints attribute 1330
syn_fsm_correction directive 1332
syn_gatedclk_clock_en_polarity directive 1335
syn_global_buffers
and xc_global_buffers 1339
syn_global_buffers attribute 1339

syn_hier
using with fanout guides 1398
syn_hier attribute 1344
and xc_use_keep_hierarchy 1625
syn_highrel_ioconnector attribute 1352
syn_insert_buffer attribute 1353, 1358, 1360
syn_insert_pad attribute 1376
syn_isclock directive 1380
syn_keep
compared with syn_preserve and syn_noprune directives 1384
DSP block inference, Altera 1907
syn_keep directive 1382
syn_loc attribute 1388
translating Altera QSF constraints 997
syn_looplimit directive 1390
syn_macro directive 1392
syn_map_dffrs attribute 1395
syn_max_memsize_reg attribute 1403
syn_maxfan attribute 1397
Altera 1911
Xilinx designs 2100
syn_multstyle
DSP block inference, Altera 1907
syn_multstyle attribute 1404
syn_netlist_hierarchy attribute 1413
syn_no_compile_point attribute 1429
syn_noarrayports attribute 1420
syn_noclockbuf attribute 1421
using with fanout guides 1398
syn_noclockpad attribute 1427
syn_noprune directive 1431
compared with syn_keep and syn_preserve 1432
syn_packer_effort_level attribute 1439
syn_pad_type attribute 1440
syn_pipeline attribute 1442
syn_placer_effort_level attribute 1447
syn_preserve
compared with syn_keep and syn_noprune 1449
DSP block inference, Altera 1907
syn_preserve directive 1448
syn_preserve_sr_priority attribute (Microsemi) 1452
syn_probe attribute 1454
syn_radhardlevel 1462
Altera and Xilinx 1464
Altera and Xilinx options 1465
Altera and Xilinx syntax 1465
Microsemi options 1463
Microsemi syntax 1463
TMR. *See* TMR, distributed TMR
syn_ramstyle attribute 1467
See also RAM inference
syn_reduce_controlset_size attribute 1484
syn_reference_clock attribute 1490
effect on multiple I/O constraints (Legacy) 530
forward-annotation, Xilinx 2112
syn_rename_module directive 1493
syn_replicate
using with fanout guides 1398
syn_replicate attribute 1494
Altera fanouts 1912
syn_resources attribute 1497
syn_romstyle attribute 1509
Altera 1502
syn_romstyle attribute (Lattice) 1500, 1507
syn_romstyle attribute (Xilinx) 1511
syn_safe_case directive 1517
syn_sharing directive 1518
syn_slow attribute 1520
syn_srlstyle attribute 1524
syn_state_machine directive 1529
syn_sxml2pxml Tcl command 1716
SYN_TCL_HOOKS variable 1719
syn_tco directive 1533
syn_tpd directive 1536
black-box timing 1536, 1543
syn_tristate directive 1539
syn_tristatetomux attribute 1540
effect of tristate pushing 221
syn_tsu directive 1543
black-box timing 1543
syn_unconnected_inputs attribute 1546
syn_unique_inst_module directive 1551
syn_use_carry_chain attribute 1553
syn_use_xilinx_placement attribute 1558
syn_useenables attribute 1559
syn_useioff
DSP block inference, Altera 1907
syn_useioff attribute
Altera 1564
Lattice 1570
Xilinx 1579

syn_user_instance
  comparison with syn_macro 1393
syn_user_instance attribute 1585
syn_vote_loops Attribute 1591
synchronous FSM from concurrent assignment statement (VHDL) 905
synchronous read RAMs
  Verilog 711
  VHDL 944
synchronous sets and resets
  Verilog 701
synchronous sets and resets (VHDL) 894
synchronous sets/resets
  inferring registers with (Altera) 1880
  Xilinx 2076
SYNCore
  adder/subtractor 1064
  adder/subtractor parameters 268
  byte-enable RAM compiler
    byte-enable RAM compiler
      SYNCore 1053
  byte-enable RAM parameters 261
  counter compiler 1076
  counter parameters 272
  FIFO compiler 1026
  FIFO parameters 247
  RAM compiler
    RAM compiler
      SYNCore 1043
  RAM parameters 257
  ROM compiler 1058
  ROM parameters 264
SYNCore adder/subtractor
  adders 1065
  dynamic adder/subtractor 1071
  subtractors 1068
SYNCore FIFOs
  definition 1026
  parameter definitions 1031
  port list 1029
  read operations 1028
  status flags 1033
  write operations 1027
SYNCore ROMs
  clock latency 1062
  dual-port read 1060
  parameter list 1061
  single-port read 1059
SYNCore wizard 245
synhooks.tcl file 1719
Synopsys customer support, contacting 59
Synopsys FPGA implementation tools
  product information 357
Synopsys FPGA products 357
Synopsys FPGA Synthesis Tools
  overview 42
Synopsys Home Page command 357
Synopsys standard sdc file. *See* sdc files, difference between legacy and Synopsys standard
Synopsys Training Page command 357
Synplicity directory (ini file) 594
synplicity.ucf file 2110
synplify command-line command 53
synplify premier command-line command 53
synplify premier dp command-line command 53
Synplify Premier synthesis tool
  overview 38
Synplify Premier tool
  Altera device support 1866
Synplify Premier/DP
  Project view 62
Synplify Premier/DP tools
  user interface 45
Synplify Pro synthesis tool
  overview 38
Synplify Pro tool
  Project view 62
  user interface 45
Synplify tool
  Project view 62
  user interface 45
synplify_pro command-line command 53
syntax
  bus dimension separator 1134
  bus naming 1133
Syntax Check command 238
synthesis
  attributes and directives (VHDL) 963
  attributes and directives, Verilog 758
  examples, VHDL 965
  guidelines
    Verilog 667
  guidelines (VHDL) 879
  log file (.srr) 608
    watching selected information 81

stopping 243
Synthesis Check command 238
Synthesis Constraints Optimization Environment (Legacy SCOPE)
See Legacy SCOPE spreadsheet
synthesis jobs
monitoring 243
synthesis macro, Verilog 753
synthesis software
flow 56
gui 45
synthesis_off directive 1593
synthesis_off directive, handling 209
synthesis_on directive 1593
synthesis_on directive, handling 209
Synthesize command 237
system clock 618
System Designer
HAPS board file 196
Tcl batch command 1658
system designer 236
SystemVerilog 212, 773, 811, 816
.* connection 814
.name connection 813
$bits system function 825
always_comb 801
always_ff 804
always_latch 803
block name on end 799
constants 782
data objects 781
data types 769
do-while loops 797
enumerated types 771
ignoring code with synthesis_off/on 1593
interface construct 773, 816
keywords 836
limitations 767
literals 769
nets 783
operators 786
packages 811
procedural blocks 800
type casting 772
typedef 770
unnamed blocks 798
variables 782
SystemVerilog data types
assignment for syn_keep 1383
SystemVerilog keywords
context help 99

T

ta file (customized timing report) 606
task declaration
automatic 659
Tcl
c_diff collection command 1795
c_intersect collection command 1796
c_list collection command 1797
c_print collection command 1798
c_symdiff collection command 1798
c_union collection command 1799
collection commands 1794
set_modules collection command 1802
-verilog argument 1640
-vhdl argument 1640
Tcl (Tool Command Language) 1636
tcl argument
-_include 1641
Tcl collection commands 1794
c_diff 1795
c_intersect 1796
c_list 1797
c_print 1798
c_symdiff 1798
c_union 1799
set_modules 1802
Tcl collection operators 1794
Tcl commands
add_file 1639
add_folder 1645
assign_to_region 1645
collections 480
constraint files 461
constraint_file 1648
create_region 1649
dh_module_sources 1651
get_env 1653
get_option 1653
hdl_param 1654
impl 1656
log file commands 1721
pasting 85
project 1662

project_data 1669
project_file 1670
set_option 1679
slice_primitive 1714
syn_sxml2pxml 1716
syntax for Tcl hooks 1719
synthesis 1637
Tcl conventions 1636
Tcl expand command 1791
Tcl find command 1774
case sensitivity 1780
examples 1781
object prefixes 1778
object types 1778
regular expression syntax 1778
special characters 1778
syntax 1775
wildcards 1778
TCL Help command 362
Tcl Script
Tcl Window command 161
Tcl Script and Messages windows 84
Tcl Script window 84
Output Windows 89
Tcl scripts
examples 1766
running 239, 240
Tcl shell command
sdc2fdc 988
TCL View, SCOPE 494
Tcl VIF commands 1726
Tcl window
popup menu 366, 367
popup menu commands 85
popup menus 85
Tcl Script and Messages windows 84
Tcl Window command 161
Technical Resource Center
accessing 357
description 135
setting options 360
setting type of update 361
specifying PDF reader (UNIX) 352
specifying web browser (UNIX) 352
update frequency 361
technologies
CPLD (Xilinx) 2134
Spartan (Xilinx) 2129
Virtex (Xilinx) 2129
Technology view 91
creating 305
displaying 117
file (.srm) 604, 606
popup menu 402
popup menu commands 402
printing 141
technology view
displaying 1660
template, module (Verilog) 685
text
copying, cutting and pasting 147
replacing 158
Text Editor
features 98
indenting a block of text 98
opening 97
popup menu commands 367
printing 141
selecting text column 99
view 97
text editor
completing keywords 98
Text Editor view 97
text macro
Verilog 754
the 1017
through constraints
point-to-point delays 486
point-to-point delays (Legacy) 533
through points
clocks 512
lists, multiple 509
lists, single 508
multiple 509
product of sums UI 509
single 508
specifying for timing exceptions 508
specifying for timing report 293
time borrowing 1859
TIMESPEC I/O constraint (Xilinx)
xc_use_timespec_for_io attribute 1628
timing
syn_tco directive 1533
syn_tpd directive 1536

syn_tsu directive 1543
timing analysis of critical paths (HDL Analyst tool) 448
timing analyst
cross-clock paths 618
generating report 284
timing analyzer
wildcards 299
timing annotated properties (.tap) 607
timing constraints
checking 238
conflict resolution 513
constraint priority 513
create_clock 1102
create_generated_clock 1104
FPGA 1100
reset_path 1106
See also FPGA timing constraints
See constraints
set_clock_groups 1108
set_clock_latency 1111
set_clock_route_delay 1113
set_clock_uncertainty 1114
set_datapathonly_delay 1116
set_false_path 1118
set_input_delay 1120
set_max_delay 1122
set_multicycle_path 1125
set_output_delay 1128
set_reg_input_delay 1131
set_reg_output_delay 1132
timing exceptions
False Paths panel 503
multicycle paths 500
priority 513
specifying paths/points 503
timing failures, definition 450
timing information, displaying (HDL Analyst tool) 311
timing report 615
asynchronous clock report 292
clock relationships 618
customized (.ta file) 606
defining through points 293
detailed clock report 620
file (.ta) 284, 606
header 616
interface information 619
performance summary 616
specifying slack margin 295
using path filtering 295
timing report file
generating custom 291
stand-alone 293
Timing Report panel
Implementation Options dialog box 205
Number of Critical Paths 206
Start/End Points 206
Timing Report View 285
options 285
timing reports
asynchronous clocks 622
file. *See* timing report file
filtering 292
log file (.srr) 612
parameters 284
stand-alone 284
stand-alone (.ta file) 284
Tip of the Day command 363
title bar information, HDL Analyst tool 432
TMR
Altera and Xilinx syn_radhardlevel 1464
Microsemi syn_radhardlevel 1462
to points 296
clocks 511
multiple 507
objects 506
Timing Analyzer 296
Toggle bookmark command 148
toolbars 114
FSM 119, 120
moving and docking 114
Toolbars command 160
Tools menu 329, 552
commands 552
tooltips
displaying 164
top-level designs
importing Vivado IP 1019
Translate Vendor IO command 238
translate_off directive 1593
translate_on directive 1593
translating constraints 242
transparent hierarchical instances 422
lower-level logic on multiple sheets 434
operations resulting in 441
pins and pin names 425
transparent instances
displaying pins 313

trees of objects, Hierarchy Browser 93
trees, browser, collapsing and expanding 94
TriMatrix memory 1883
tristates
  black_box_tri_pins directive 1197
  converting drivers to muxes 1540
  pushing tristates, description 220
  pushing tristates, example 220
  pushing tristates, pros and cons 221
  syn_tristate directive 1539
tristates, Verilog 689
true dual-port block RAM 2095
true dual-port RAM
  See also multi-port RAMs
type casting 777
  SystemVerilog 772
typedef (SystemVerilog) 770

U

UCF
  clock priority with syn_clock_priority 1241
  forward-annotation of clock priority 1242
UCF constraints
  supported constraints 1005
  translation examples 1006
UCF file
  forward-annotating delay constraints 2113
  forward-annotating multicycle paths 2112
ucf2sdc
  syntax 1004
  translation examples 1006
ucftosdc Tcl command 1004
Uncomment Code 148
Undo command 147
Unfilter command 163
unfiltered schematic, compared with filtered 416
unfiltering 312
  FSM diagram 163
  schematic 312
unfiltering schematic 444
Unflatten Current Schematic command 309
Unhide Instances command 312
unhiding hidden instance 312
union construct (SystemVerilog) 775
UNIX
  configure external programs 331
unnamed blocks (SystemVerilog) 798
Unselect All command 315
  View menu (FSM Viewer) 163
unsigned arithmetic (VHDL) 843
unsigned functions 656
unsupported language constructs
  VHDL
    configuration declaration 919
    configuration specification 913
unsupported language constructs (VHDL) 839
updates from the Resource Center 357
Uppercase command 148
Use OFFSET for I/O constraints option 1628
use statement (VHDL) 851
user interface
  Synplify Premier/DP tools 45
  Synplify Pro tool 45
  Synplify tool 45
user interface, overview 61
user-defined enumeration data types (VHDL) 842
user-defined functions, SNS (VHDL) 871
using the mouse 107
utilies
  sdc2fdc 988
utilities
  qsf2sdc 994
  s2o 1098
utilization
  syn_packer_effort_level 1439

V

v file 596
variables
  accessing, get_env Tcl command 1653
  reporting 1653
  SystemVerilog 782
variables (VHDL) 848
VCS Simulator command 239
Vendor Constraints
  Implementation Results panel, Implementation Options dialog box 205
  writing 205

vendor technologies
  Achronix 1829
  Altera 1865
  Lattice 1949
  Microsemi 2027
  Xilinx 2063
vendor-specific Tcl commands 1717
verification mode (Xilinx) 1914, 2103
Verilog
  `ifdef and `define statements 215
  'ifdef 753
  allow duplicate modules (Tcl option) 1686
  always blocks 675
    combinational logic 691
    event control 692
    level-sensitive latches 697
    multiple event control arguments 675
  assignment operators 640
  asynchronous sets and resets 700
  asynchronous state machines 708
  attribute syntax 758
  beta features 218
  black boxes 749
  blocking statements and mapping Altera RAMs 721
  built-in gate primitives 689
  byte enable in block RAM 741
  combinational logic 690
  combined data, port types 643
  comma-separated sensitivity list 644
  comments, syntax 686
  compiler, configuring 330
  constant function (Verilog 2001) 646
  continuous assignments 693, 697
  cross-module referencing 663
  directive syntax 758
  extract design parameters 215
  flip-flops using always blocks 696
  Forward Annotation of Initial Values 683
  gate primitives 689
  generic technology library 600
  hierarchical design 751
  hierarchy 751
  ignored language constructs 641
  ignoring code with 'ifdef 753
  ignoring code with translate off/on 1593
  initial value data file 680
  initial values for RAMs 678
  initial values for registers 676
  instantiating
    black boxes 749
    gate primitives 689
  language
    constructs 639
  language guidelines 675
  level-sensitive latches 697
  library directories 213
  limited RAM resources 710
  localparams (Verilog 2001) 656
  module template 685
  multidimensional array (Verilog 2001) 660
  multiplexer 692
  multi-port RAM examples 714
  Multi-port RAM extraction 711, 942
  naming syntax for constraints 1157
  netlist file 608
  operators 639
  overriding parameter value
    with # 688
    with defparam 688
  PREP benchmarks 750
  primitives 689
  RAM inference 709
  ROM inference 746
  scalable adder, creating 687
  scalable modules 686
  scaling by overriding parameter value
    with # (example) 688
    with defparam (example) 688
  sequential logic 695, 697
  sets and resets 699
  signal assignments always blocks 696
  signed multipliers 674
  signed signals (Verilog 2001) 645, 646, 656, 658
  simple gates 689
  source files (.v) 596
  specifying compiler directives 215
  state machines 705
  state values (FSM) 707
  Stratix RAM with control signals 724
  structural netlist file (.vm) 608
  structural Verilog 751
  supported language constructs 639
  supported standards 598
  syn_keep on multiple nets 1383
  synchronous read RAMs 711

synchronous sets and resets 701
synthesis macro 753
synthesis text macro 753
text macro 754
tristate gates 689
wildcard (*) in sensitivity list 642, 644
Verilog 2001 598
constant statement 646
localparams 656
multidimensional array 660
signed signals 645, 646, 656, 658
Verilog panel 212
Verilog 2001 support 642
Verilog 95 598
-verilog argument
Tcl 1640
Verilog include files
using _SEARCHFILENAMEONLY_ directive 218
Verilog language support 637, 763
Verilog panel 212
Implementation Options dialog box 211
Multiple File Compilation Unit 212
options 212
Push Tristates 212
SystemVerilog 212
Verilog source file (.v) 596
Verilog synthesis guidelines 667
version information 363
vhd file 596
vhd source file 596
VHDL
accessing packages 851
adding source files to design library 849
assignment operators 844
assignments 880
asynchronous FSM created with process 905
asynchronous sets and resets 893
asynchronous state machines 904
attribute syntax 963
attributes package 963
black boxes 961
byte enable in block RAM 954
case statement 861
clock edges 884
clock edges, wait statements 886
combinational logic
definition 865
examples 965
comments, syntax 881
compiler, configuring 330
compiling design units into libraries 849
component instantiation 866
concurrent signal assignments 862
conditional signal assignments 864
configuration
declaration 914
specification 910
configuration statement 920
constants 848
SNS (Selected Name Support) 868
D flip-flop with active-high reset, set
asynchronous 894
synchronous 896
data types 841
guidelines 881
declaring and assigning objects 844
default assignment 901
demand loading 873
design libraries 849
detecting reset problems 902
directive syntax 963
dynamic range assignment 844
enumerated types as state values 901
enumeration encoding, default 208, 881
events, defining outside process 885
flip-flops 883
forgotten assignment to next state, detecting 904
FSM coding style 899
generics for scalable designs 935
hierarchical designs 906
if-then-else statement 860
ignored language constructs 840
ignoring code with synthesis off/on 209
inferring RAMs 940
inferring ROMs 958
initialization state, extra 902
instantiating
black boxes 961
components 852, 866
instantiating components 852
integer data type 843
language
constructs 838, 840
guidelines 880
support 837

latch error, example 889
level-sensitive latches
  concurrent signal assignment 887
  process blocks 888
  unwanted 889
libraries 849
  attributes, supplied with synthesis tool 597, 850
library and package rules 851
library packages
  accessing 851
  attributes package 963
  IEEE support 838
  predefined 850
library statement 851
limited RAM resources 942
model template 881
multi-port RAM extraction 946
naming objects 844
naming syntax for constraints 1158
object naming syntax 844
operators 853
packages 849
ports 846
predefined enumeration types 841
predefined packages 850
PREP benchmarks 967
process keyword 858
process template 883
  modeling combinational logic 858
processes 880
  creating flip-flops and registers 883
RAM inference 940
registers 883
reset signals 893
resource library, creating 852
resource sharing 865
ROM inference 958
RTL view primitives 894
scalable architecture, using 936
scalable design
  creating using generate statements 938
  creating using generics 935
  creating using unconstrained vector ports 934
scalable designs 934
  generate statement 938
  generics 935
  unconstrained vector ports 934
Selected Name Support (SNS) 868
selected signal assignments 863
sensitivity list 859
sequential logic 865
  examples 966
sequential statements 860
set signals 893
sharing operators in case statements 865
signal assignments 844
  concurrent 862
  conditional 864
  selected 863
  simple 863
signals 846
simple component instantiation 866
simple signal assignments 863
simulation using enumerated types 902
SNS 868
  constants 868
  demand loading 873
  functions and operators 869
  user-defined function support 871
source files (.vhd) 596
state machines 897
statements
  case 861
  generate 938
  if-then-else 860
  library 851
  use 851
  wait 886
structural netlist file (.vhm) 608
supported language constructs 838
supported standards 597
synchronous FSM from concurrent assignment statement 905
synchronous read RAMs 944
synchronous sets and resets 894
synthesis
  attributes and directives 963
  examples 965
  guidelines 879
unsupported language constructs 839
  configuration declaration 919
  configuration specification 913
use statement 851
user-defined enumeration data types 842

variables 848
wait statement inside process 886
VHDL 2008 969
enabling 980
operators 970
packages 978
-vhdl argument
Tcl 1640
VHDL assignment
dynamic range 844
VHDL components
creating resource library 853
instantiating 852, 866
specifying configurations 911
vendor macro libraries 853
VHDL componentsL
configuration declarations 914
VHDL generic mapping
configuration statement 920
VHDL libraries
compiling design units 849
setting up 172
VHDL library
specifying non-default (Altera) 1876
VHDL multiple entities
configuration statement 922
VHDL panel
DesignWare options 209, 213
Implementation Options dialog box 207
VHDL port mapping
configuration statement 921
VHDL source file (.vhd) 596
vhm file 608
View FSM command 315
View FSM Info File command 315
View Log File command 161
View menu 160
Filter submenu 163
Log File command 165
Rats Nest submenu 547
RTL and Technology view commands 162
View Result File command 161
View Sheets command 162
views 80, 536
FSM 95
managing 113
Project 62
removing 114
RTL 90
Technology 91
VIF
mapping multipliers (Altera Stratix) 1915
vif_add_file command 1726
vif_add_library command 1727
vif_set_constant command 1727
vif_set_equiv command 1728
vif_set_fsm command 1729
vif_set_fsmreg command 1729
vif_set_map_point command 1729
vif_set_merge command 1730
vif_set_port_dir command 1730
vif_set_result_file command 1731
vif_set_state_map command 1731
vif_set_top_module command 1732
vif_set_transparent command 1732
VIF commands
vif_add_file (Tcl) 1726
vif_add_library (Tcl) 1727
vif_set_constant (Tcl) 1727
vif_set_equiv (Tcl) 1728
vif_set_fsm (Tcl) 1729
vif_set_fsmreg (Tcl) 1729
vif_set_map_point (Tcl) 1729
vif_set_merge (Tcl) 1730
vif_set_port_dir (Tcl) 1730
vif_set_result_file (Tcl) 1731
vif_set_state_map (Tcl) 1731
vif_set_top_module (Tcl) 1732
vif_set_transparent (Tcl) 1732
vif file 607
vif_add_file command 1726
vif_add_library command 1727
vif_set_constant command 1727
vif_set_equiv command 1728
vif_set_fsm command 1729
vif_set_fsmreg command 1729
vif_set_map_point command 1729
vif_set_merge command 1730
vif_set_port_dir command 1730
vif_set_result_file
command 1731
vif_set_state_map
command 1731
vif_set_top_module
command 1732

vif_set_transparent
  command 1732
Virtex 5, RAM inference 743
virtual clocks (Legacy) 521
Visual Properties command 162
Vivado IP 1018
vm file 608
vqm files
  naming requirements 1922

W

wait statement, inside process (VHDL) 886
Watch Window. *See* Log Watch window
web browser,specifying for UNIX 352
web updates 357
What's Cool 101
What's New 101
white boxes
  ise2syn 227
wide adder/subtractor 2082
wildcards
  naming syntax, Verilog 1157
  naming syntax, VHDL constraints 1158
  Tcl find command 1777, 1778
  text Find 150
  text replacement 158
  timing analyzer 299
  Verilog sensitivity list 642, 644
window
  Project 62
windows 80, 536
  closing 125
  log watch 81
  removing 114
Windows, 64-bit mapping 198
wires, preserving with syn_keep directive 1382
Write Output Netlist Only command 238

X

xc_area_group (Xilinx) 1595
xc_clockbuftype attribute 1597
xc_fast attribute 1599
xc_fast_auto attribute 1603
xc_global_buffers
  and syn_global_buffers 1605
xc_global_buffers attribute 1605
xc_isgsr directive 1606
xc_loc attribute 1608
xc_map attribute 1610
xc_padtype attribute 1611
xc_props attribute 1613
xc_pulldown attribute 1620
xc_pullup attribute 1620
xc_rloc attribute 1622
xc_slow attribute 1624
xc_use_keep_hierarchy attribute 1625
xc_use_rpms attribute 1627
xc_use_timespec_for_io attribute 1628
xc_uset attribute 1630
xdc2fdc
  examples 1022
  syntax 1020
xflow and xtclsh 2119
Xilinx
  asymmetric block RAM 2084
  attributes 2136
  black boxes 2073, 2096
  block RAMs 738, 952, 2094
  byte enable in block RAM, Verilog 741
  byte enable in block RAM, VHDL 954
  clock groups, forward-annotating 2115
  clock skew 2105
  compile point constraints 2117
  compile point timing data 2104
  compile points 2120
  constraints 2066, 2110
  CPLD device options 2131
  CPLD options 2132
  CPLD project command 2135
  CPLD technologies 2131, 2134
  DCMs 2116
  DDR register inference 2076
  defining clock on instance 2114, 2117
  directives 2136
  distributed TMR syntax examples 1465
  DLLs 2116
  DSP component 1264, 1269
  dynamic SRL tables 2079
  fanout limits 2099
  forward-annotating I/O delays 2113
  forward-annotating multicycle paths 2112
  forward-annotating properties 1613
  forward-annotation 2110
  gated clocks 2102

generated clocks 2102
I/O locations 2097
inferring dynamic SRL 2079
inferring I/O buffers with syn_diff_io 1245
instantiating I/O pads 2096
IOB statements, writing 1579
IP cores 2071
mapping latches 2080
MMCMs 2116
multi-port RAMs 2095
OFFSET constraints for I/O delays 2113
OFFSET for I/O constraints 1628
packing IOBs 2097
packing registers, syn_useioff attribute 1579
pad types 1611
pipelining 2101
place & route options file 388
port names 2117
RAMs 2084
register balancing 2101
register inference 2076
relative locations attributes 2081
relaxing constraints 2117
resource usage report 2107
retiming 2101
running from synthesis tool 331
SLR 2065
Spartan device options 2122
Spartan options 2124
Spartan project command 2130
Spartan technologies 2129
specifying pin location 2097
syn_clock_priority 1241
syn_radhardlevel TMR 1464
syn_romstyle attribute 1511
syn_useioff attribute 1579
synthesis design flows 2066
technologies 2063
TIMESPEC for I/O constraints 1628
translating PAD file to constraint file 242
true dual-port RAM. *See* multi-port RAMs
updating compile points 2104
verification mode 2103
Virtex device options 2122
Virtex options 2124
Virtex project command 2130
Virtex technologies 2129
Xilinx differential I/O buffer inference 2097
Xilinx DSP48 register conversion 1239, 1241
Xilinx Partition Flow
  using the standalone XML converter 1716
Xilinx Partition flow
  syn_sxml2pxml Tcl command 1716
Xilinx projects
  converting with ise2syn 225
Xilinx RAM types (Verilog) 709
Xilinx RAM types (vhdl) 941
Xilinx SSI technology 2065
Xilnx forward-annotation
  datapath 1156
  datapath only 1116
XML converter
  using Xilinx Partition Flow 1716
xnf file (Xilinx) 2064
xtclsh executable 2119

Z

zoom
  using the mouse wheel and Ctrl key 112
zoom mouse pointer 161
Zoom Out command 161
Zoom Selected command
  Physical Analyst view 576